CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

13: Merge Sort, Quick Sort

Tom Yeh
he/him/his

Basics

input	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
sort left half	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
sort right half	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge results	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Mergesort overview

- Invented by John von Neumann in 1945
- Algorithm sketch:
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge the two halves

https://en.wikipedia.org/wiki/File:JohnvonNeumann-LosAlamos.gif

Merging two already sorted halves into one sorted array

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
    for (int k = lo; k <= hi; k++)
        aux[k] = a[k];

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        if (i > mid) //ran out of elements in the left subarray
           a[k] = aux[j++];
        else if (j > hi) //ran out of elements in the right subarray
            a[k] = aux[i++];
        else if (less(aux[j], aux[i]))
            a[k] = aux[j++];
        else
            a[k] = aux[i++];
    }
}
```

Merging Example - 1st step - copying to auxiliary array

Array aux

A	G	L	O	R	H	I	M	S	T
0	1	2	3	4	5	6	7	8	9

lo mid hi

Array a

A	G	L	O	R	H	I	M	S	T
0	1	2	3	4	5	6	7	8	9

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
    for (int k = lo; k <= hi; k++)
        aux[k] = a[k];   // Copy a to aux array

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        if (i > mid) //ran out of elements in the left subarray
           a[k] = aux[j++];
        else if (j > hi) //ran out of elements in the right subarray
            a[k] = aux[i++];
        else if (less(aux[j], aux[i]))
            a[k] = aux[j++];
        else
            a[k] = aux[i++];
    }
}
```

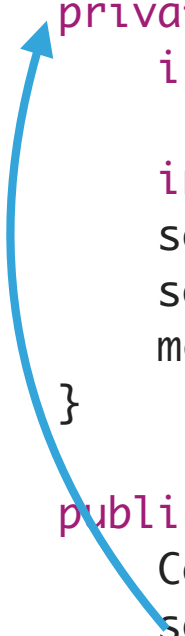
http://algs4.cs.princeton.edu

2.2 MERGING DEMO

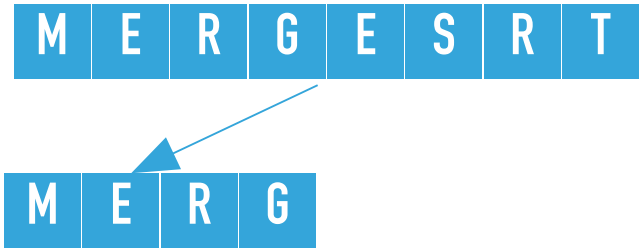
Mergesort - the quintessential example of divide-and-conquer

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}

public static void sort(Comparable[] a) {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length - 1);
}
```

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}

public static void sort(Comparable[] a) {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length - 1);
}
```

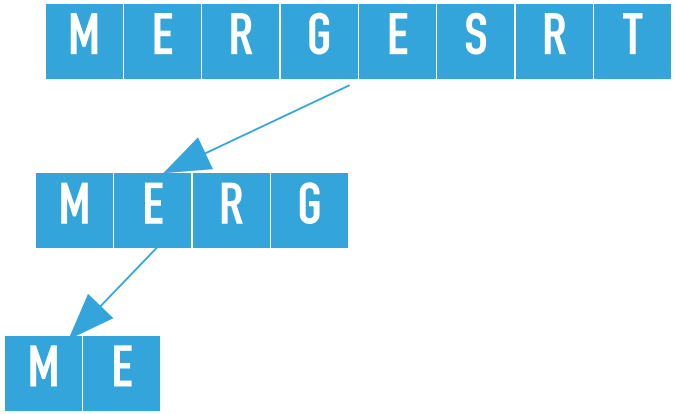


`sort([M, E, R, G, E, S, R, T])` calls
`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 7)` where the array of `nulls` is the auxiliary array, `lo = 0` and `hi = 7`.



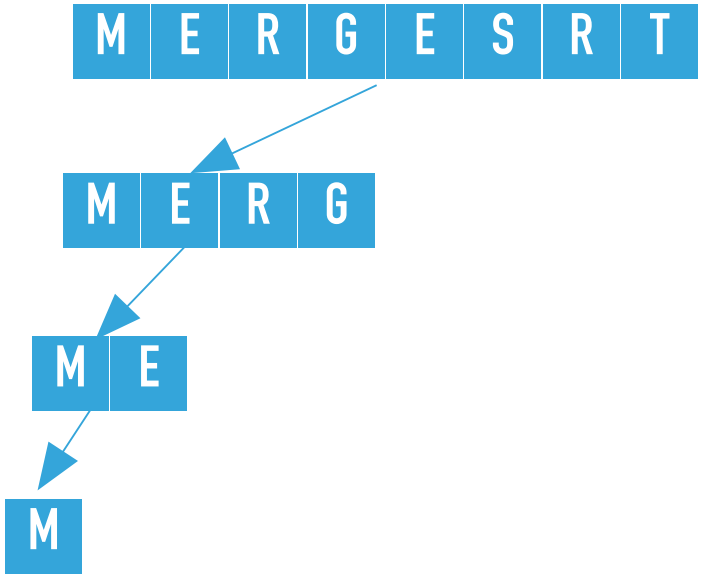

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 7)` calculates the `mid = 3` and calls recursively `sort` on the left subarray, that is `sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 3)`, where `lo = 0`, `hi = 3`



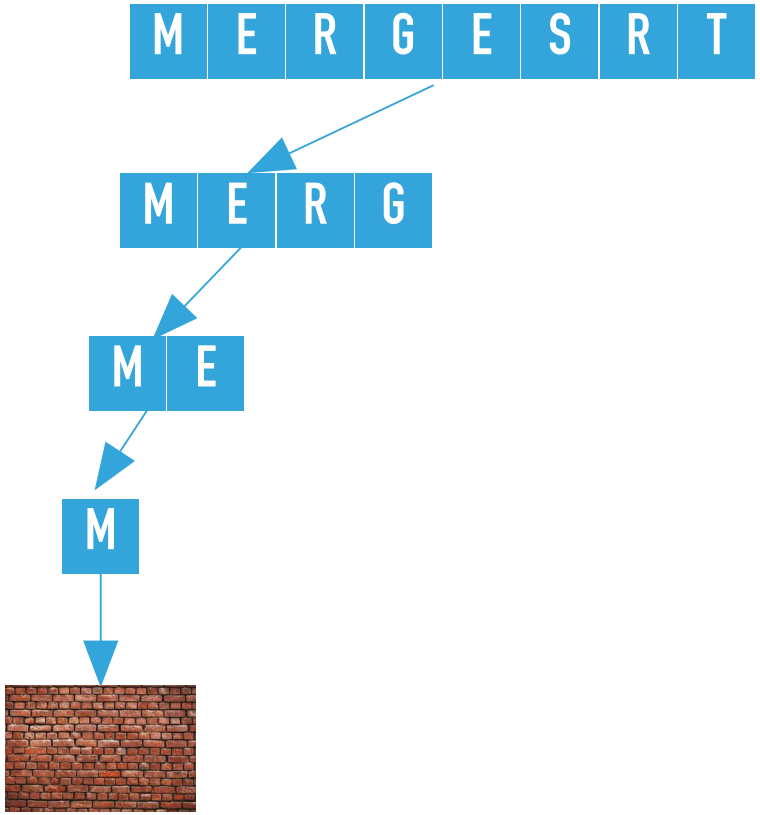

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 3)` calculates the `mid = 1` and calls recursively `sort` on the left subarray, that is `sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 1)`, where `lo = 0`, `hi = 1`



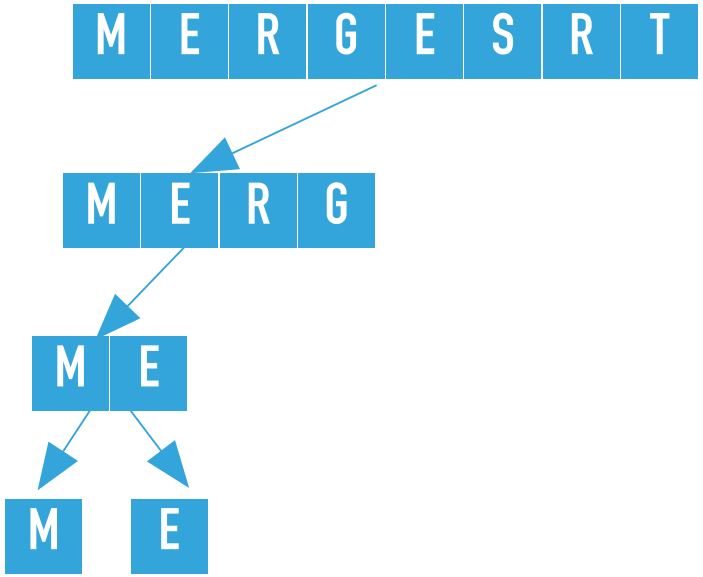

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 1)` calculates the `mid = 0` and calls recursively `sort` on the left subarray, that is `sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 0)`, where `lo = 0`, `hi = 0`



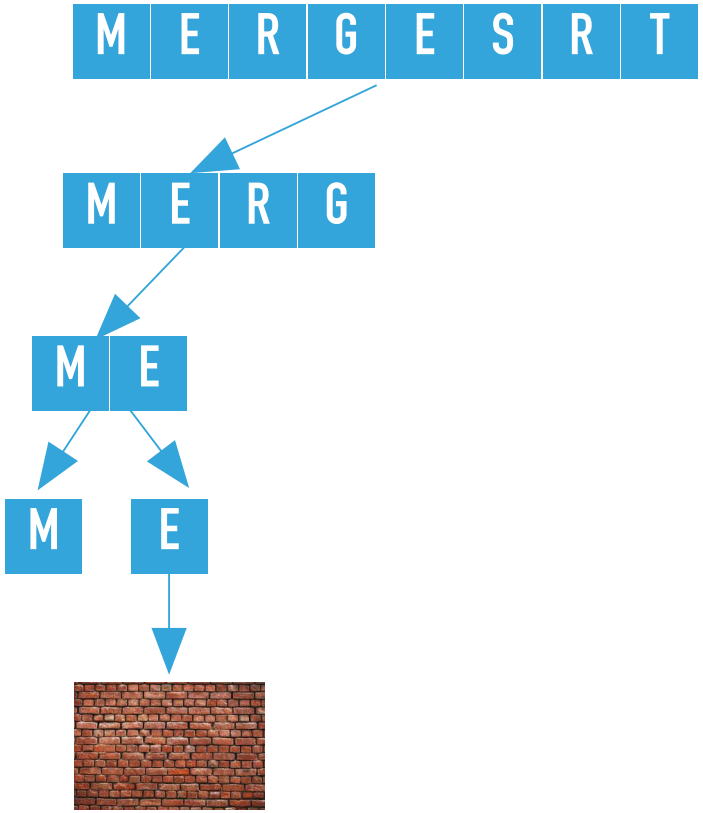

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 0)` finds `hi <= lo` and returns.



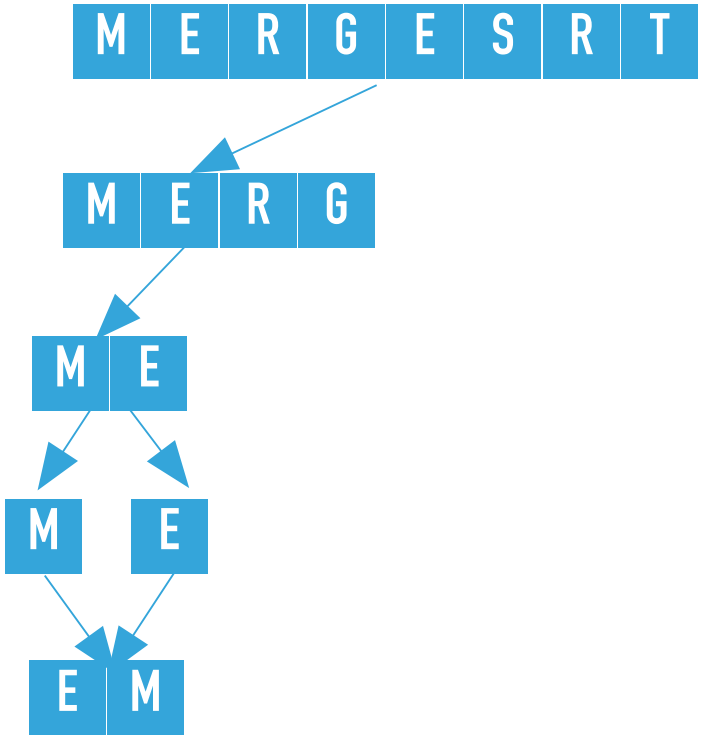

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 1)` calls recursively `sort` on the right subarray, that is `sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 1, 1)`, where `lo = 1`, `hi = 1`



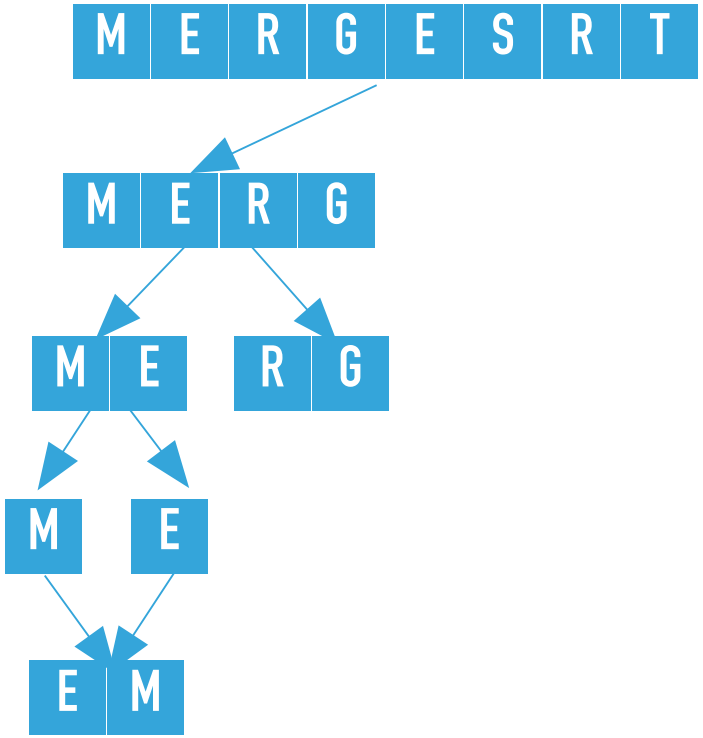

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 1, 1)` finds `hi <= lo` and returns.



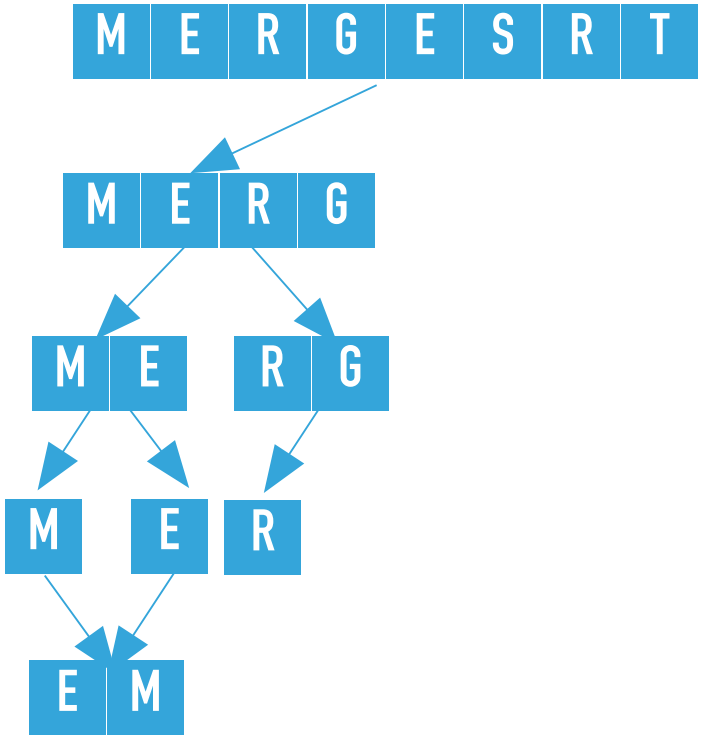

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 1)` merges the two subarrays that is calls `merge([M, E, R, G, E, S, R, T], [null, null, null, null, null, null, null, null], 0, 0, 1)`, where `lo = 0`, `mid = 0`, and `hi = 1`. The resulting partially sorted array is `[E, M, R, G, E, S, R T]`.



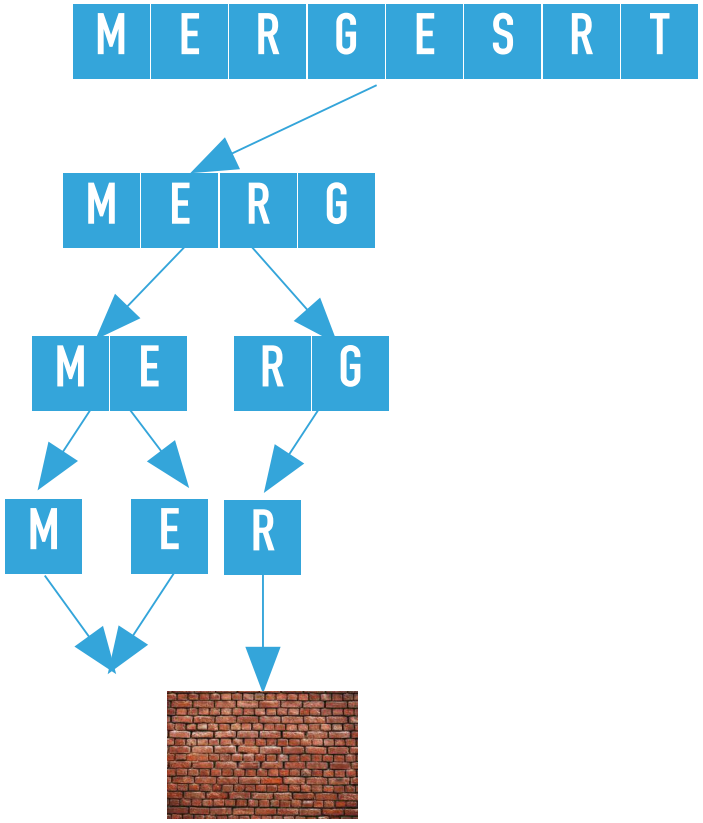

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 0, 3)` calls recursively `sort` on the right subarray, that is `sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 2, 3)`, where `lo = 2`, `hi = 3`



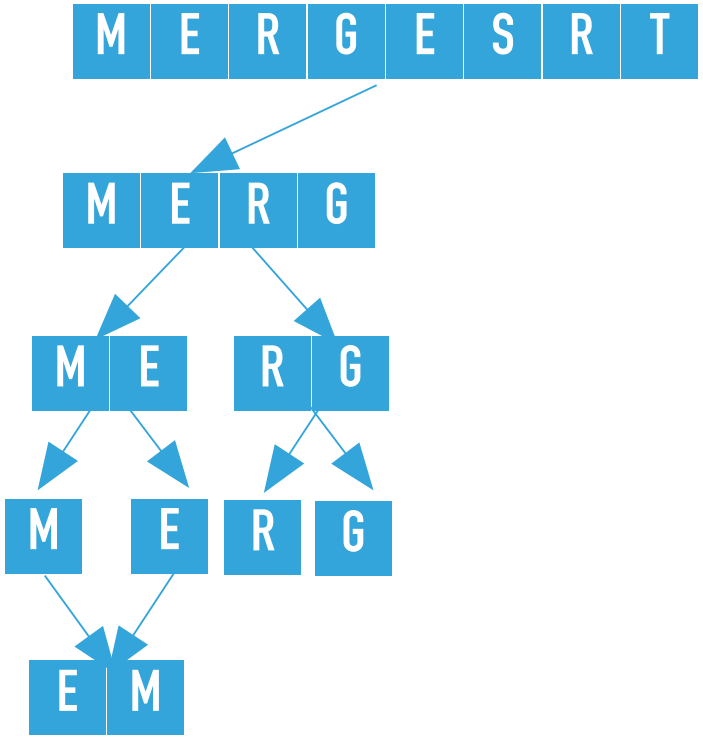

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 2, 3)` calculates the `mid = 2` and calls recursively `sort` on the left subarray, that is `sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 2, 2)`, where `lo = 2`, `hi = 2`



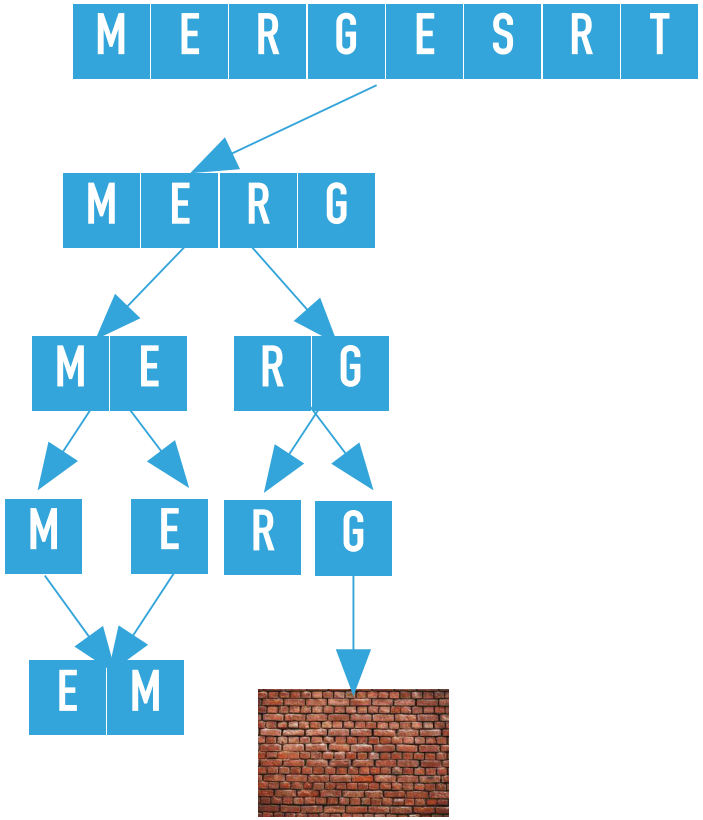

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 2, 2)` finds `hi <= lo` and returns.



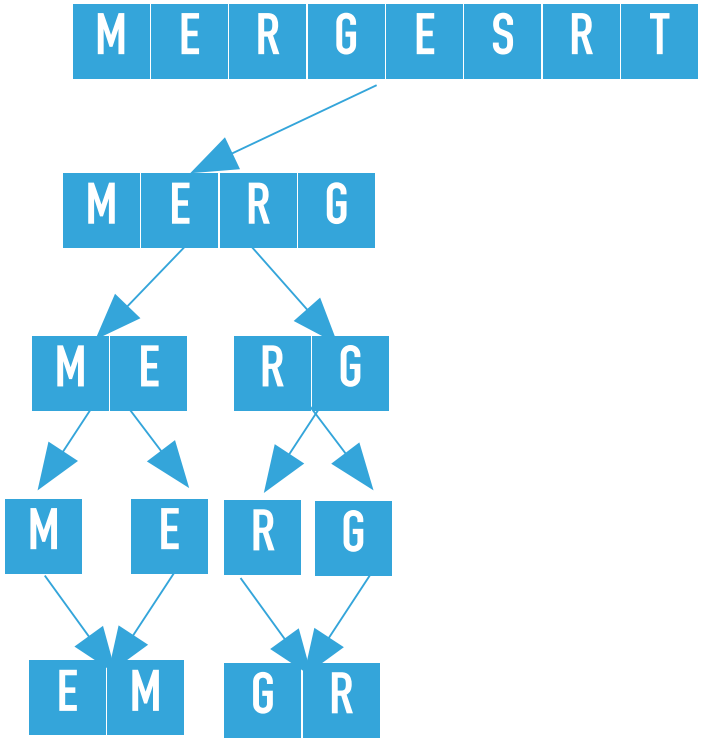

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 2, 3)` calls recursively `sort` on the right subarray, that is `sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 3, 3)`, where `lo = 3`, `hi = 3`



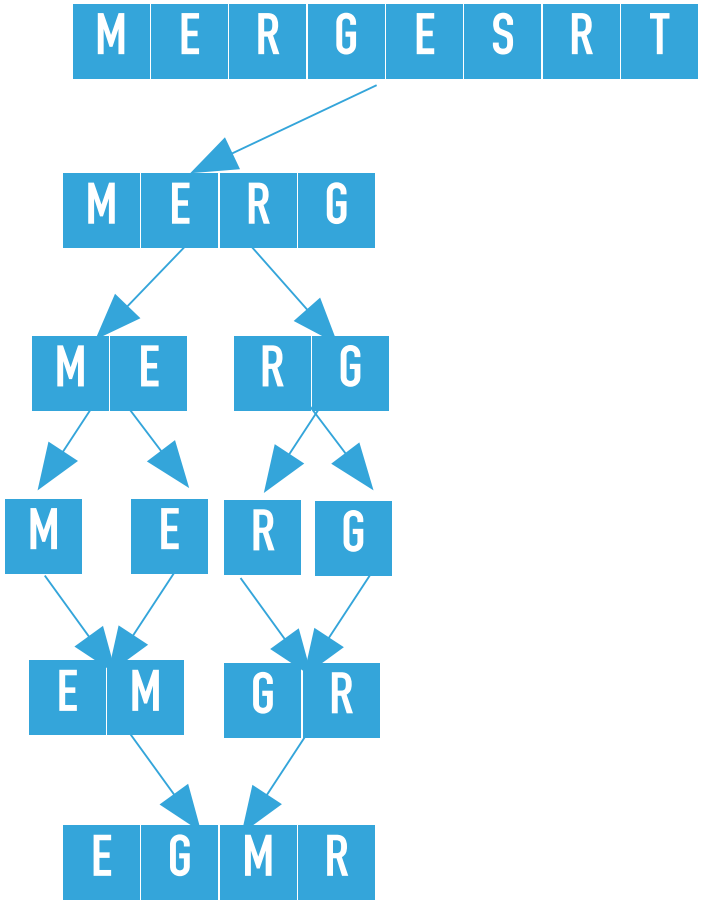

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 3, 3)` finds `hi <= lo` and returns.



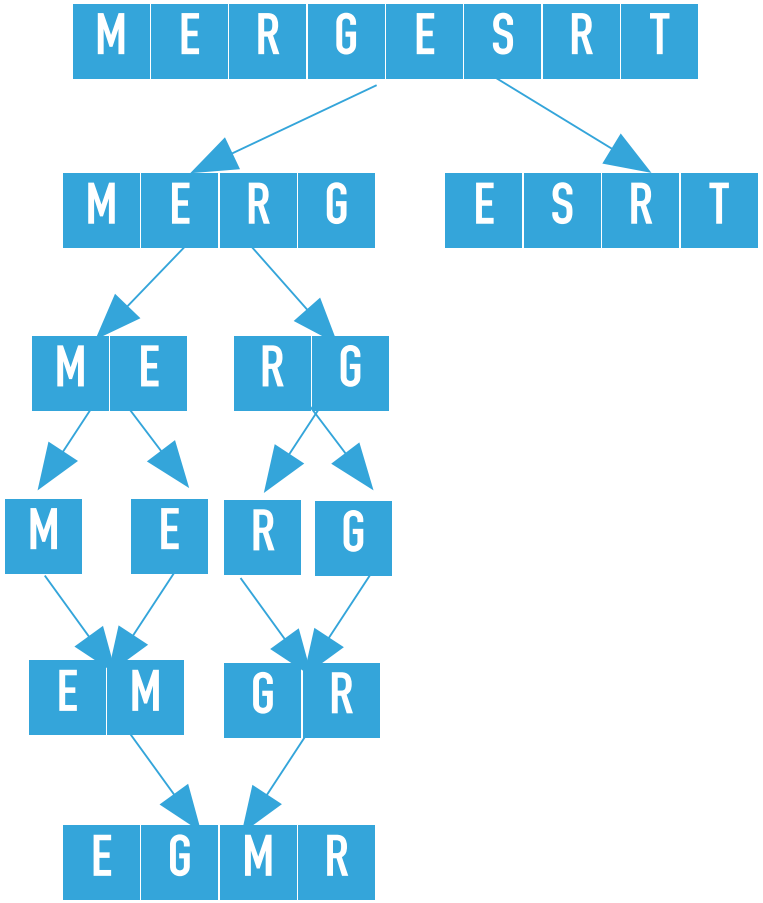

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 2, 3)` merges the two subarrays that is calls `merge([E, M, R, G, E, S, R, T], [M, E, null, null, null, null, null, null], 2, 2, 3)`, where `lo = 2`, `mid = 2`, and `hi = 3`. The resulting partially sorted array is `[E, M, G, R, E, S, R T]`.



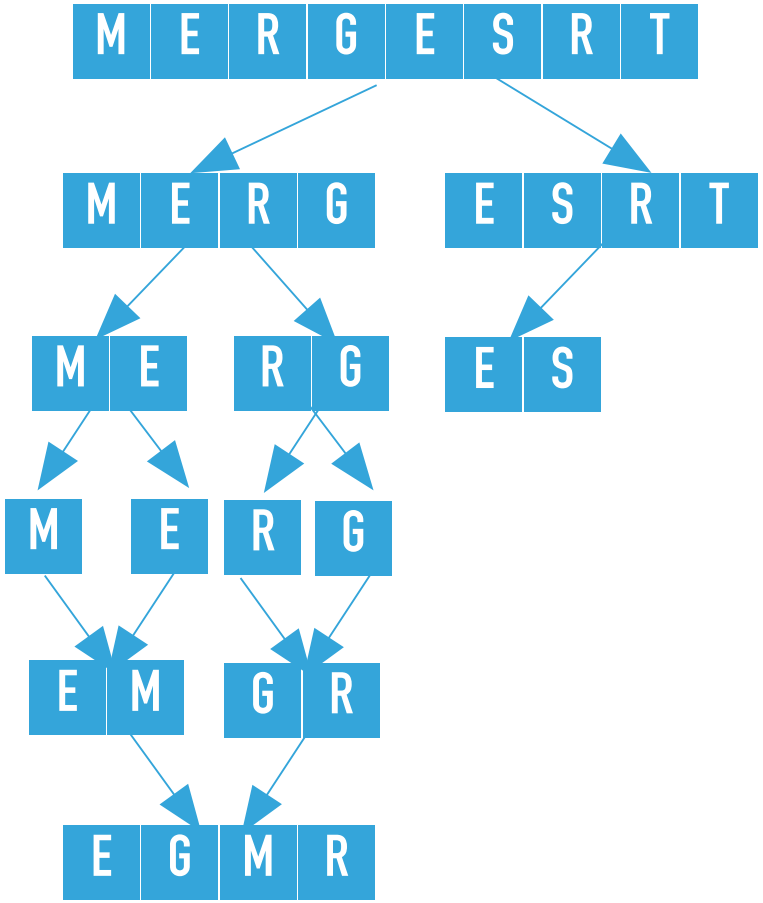

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, M, G, R, E, S, R, T], [M, E, R, G, null, null, null, null], 0, 3)` merges the two subarrays that is calls `merge([E, M, G, R, E, S, R, T], [M, E, R, G, null, null, null, null], 0, 1, 3)`, where `lo = 0`, `mid = 1`, and `hi = 3`. The resulting partially sorted array is `[E, G, M, R, E, S, R T]`.



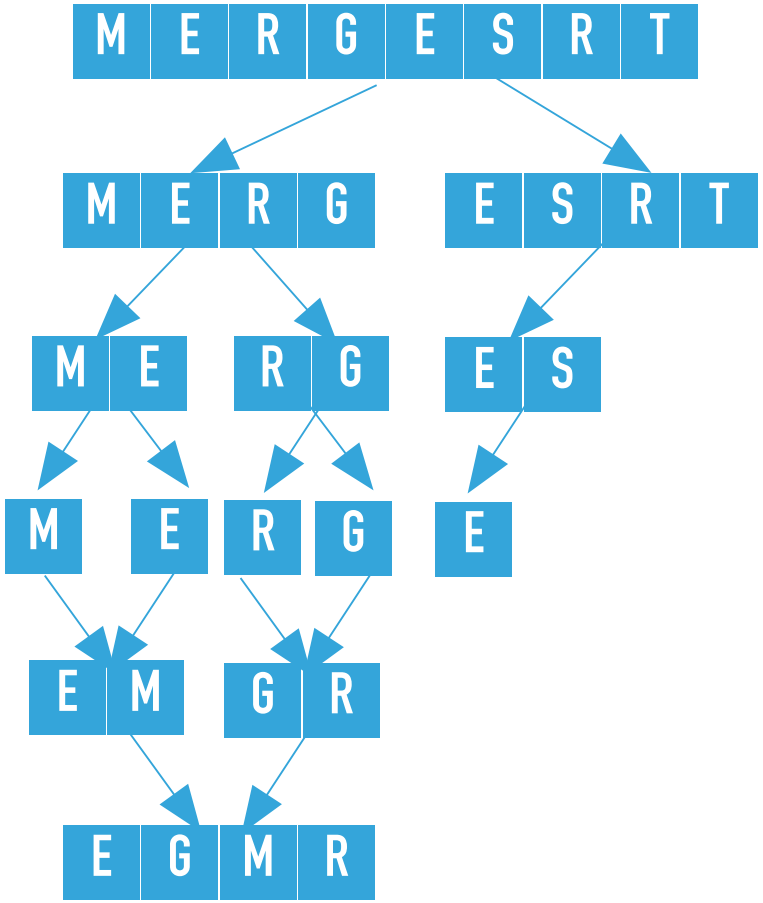

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 0, 7)` calls recursively `sort` on the right subarray, that is `sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 7)`, where `lo = 4`, `hi = 7`



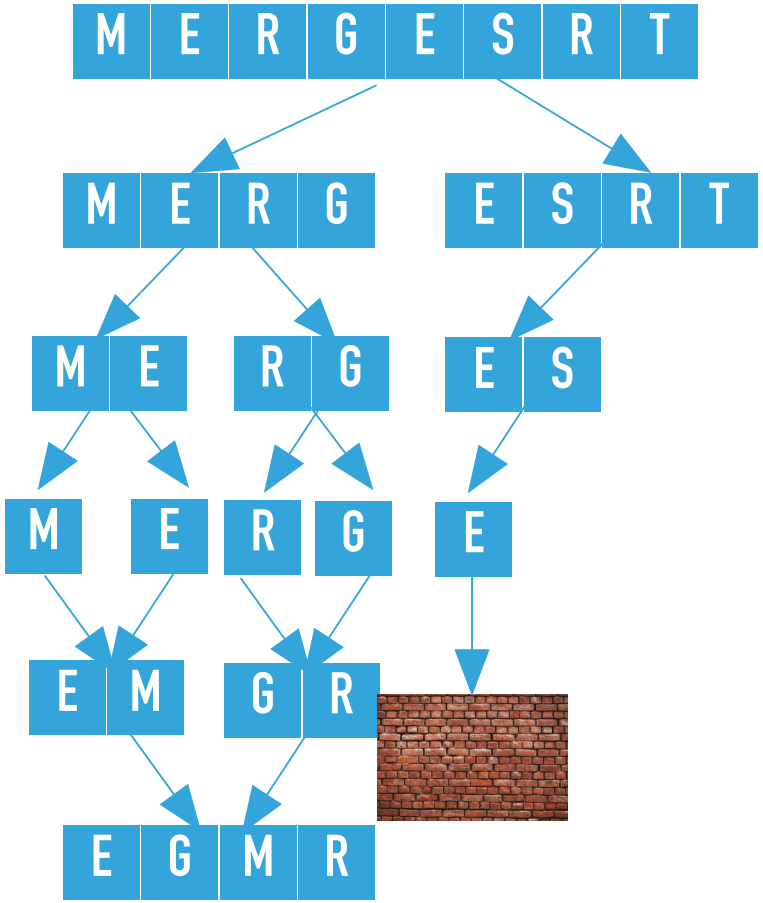

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 7)` calculates the `mid = 5` and calls recursively `sort` on the left subarray, that is `sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 5)`, where `lo = 4`, `hi = 5`.



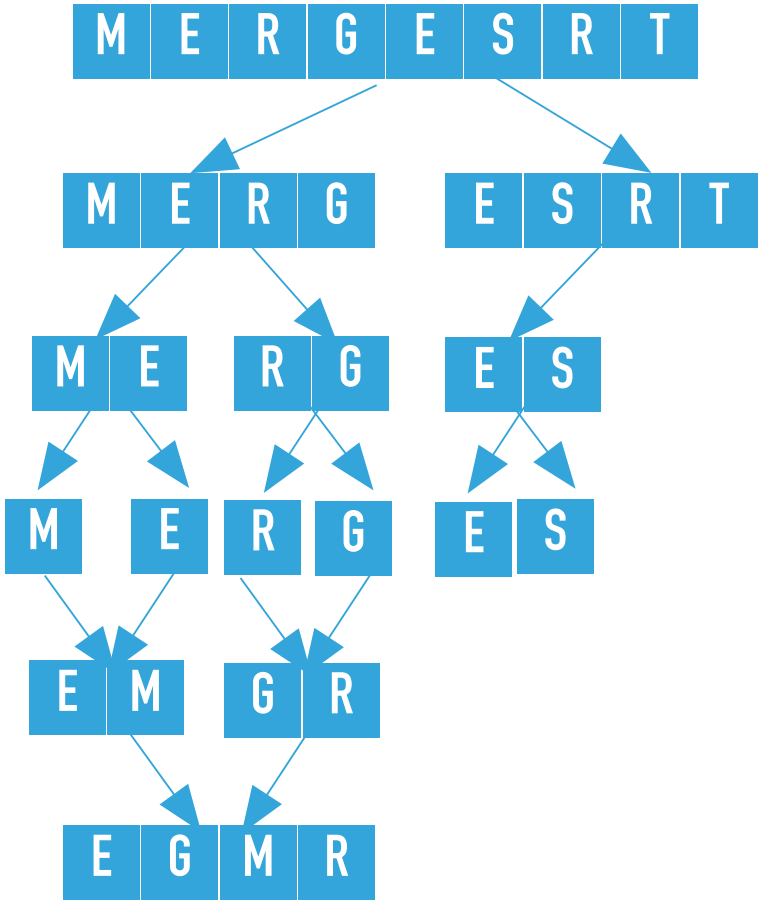

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 5)` calculates the `mid = 4` and calls recursively `sort` on the left subarray, that is `sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 4)`, where `lo = 4`, `hi = 4`.



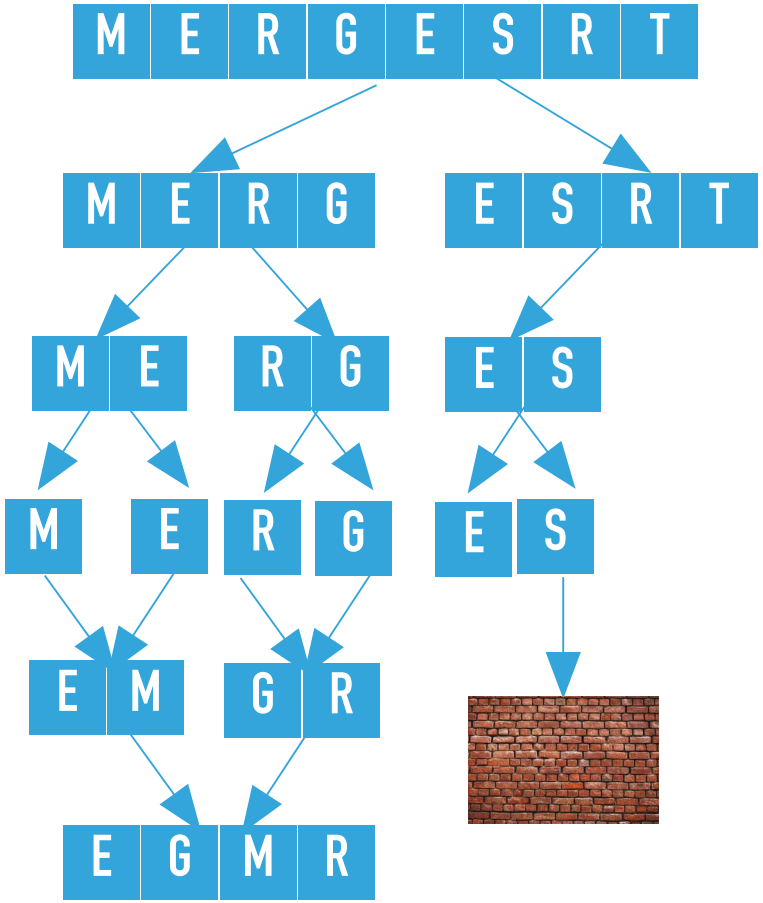

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 4)` finds `hi <= lo` and returns.



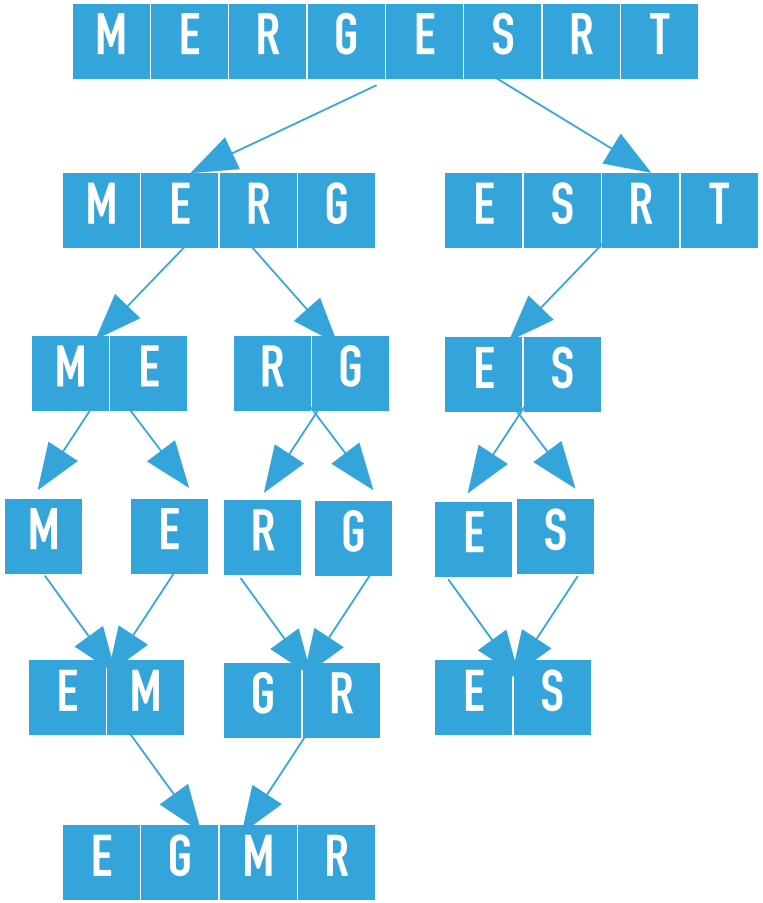

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 5)` calls recursively `sort` on the right subarray, that is `sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 5, 5)`, where `lo = 5`, `hi = 5`



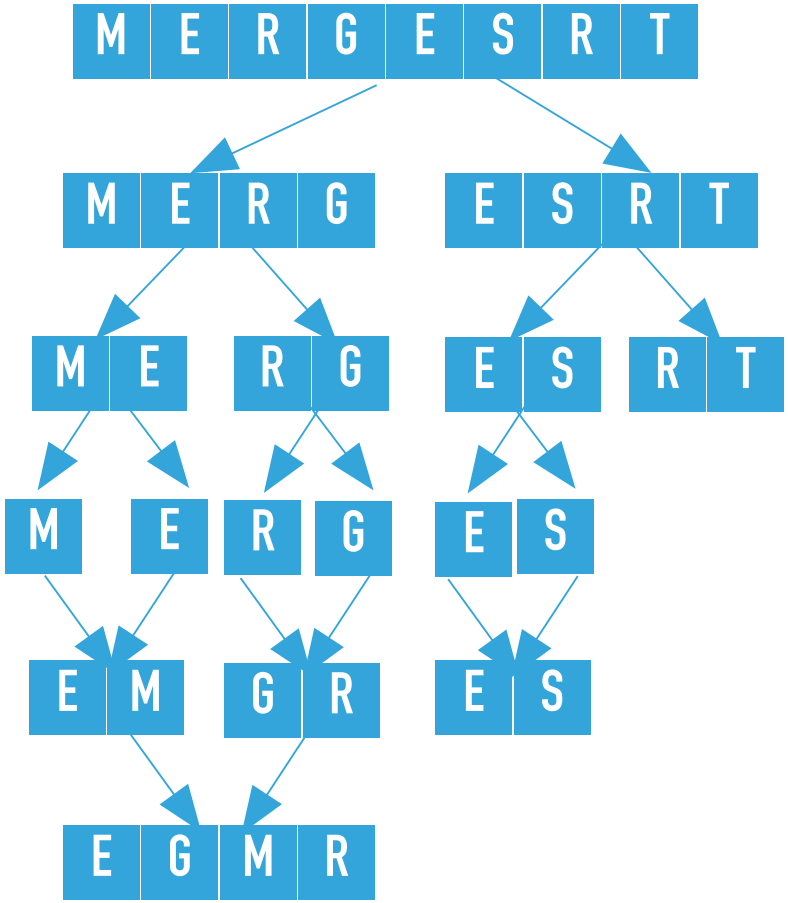

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 5, 5)` finds `hi <= lo` and returns.



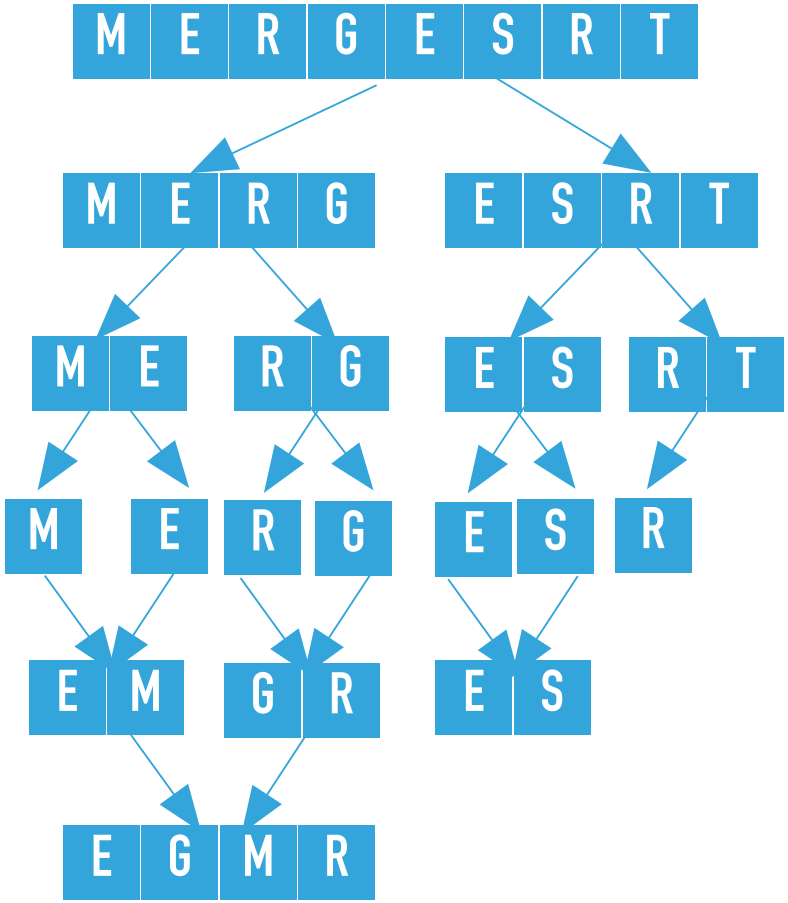

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 5)` merges the two subarrays that is calls `merge([E, G, M, R, E, S, R, T], [E, M, G, R, null, null, null, null], 4, 4, 5)`, where `lo = 4`, `mid = 4`, and `hi = 5`. The resulting partially sorted array is `[E, G, M, R, E, S, R, T]`.



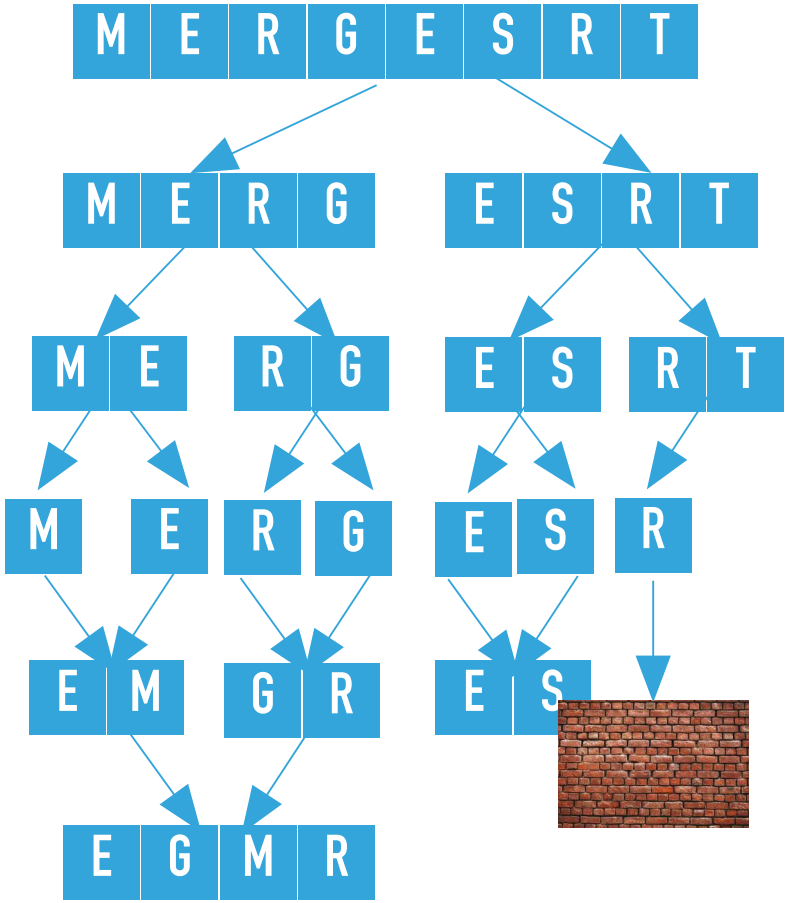

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 4, 7)` calls recursively `sort` on the right subarray, that is `sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 6, 7)`, where `lo = 6`, `hi = 7`



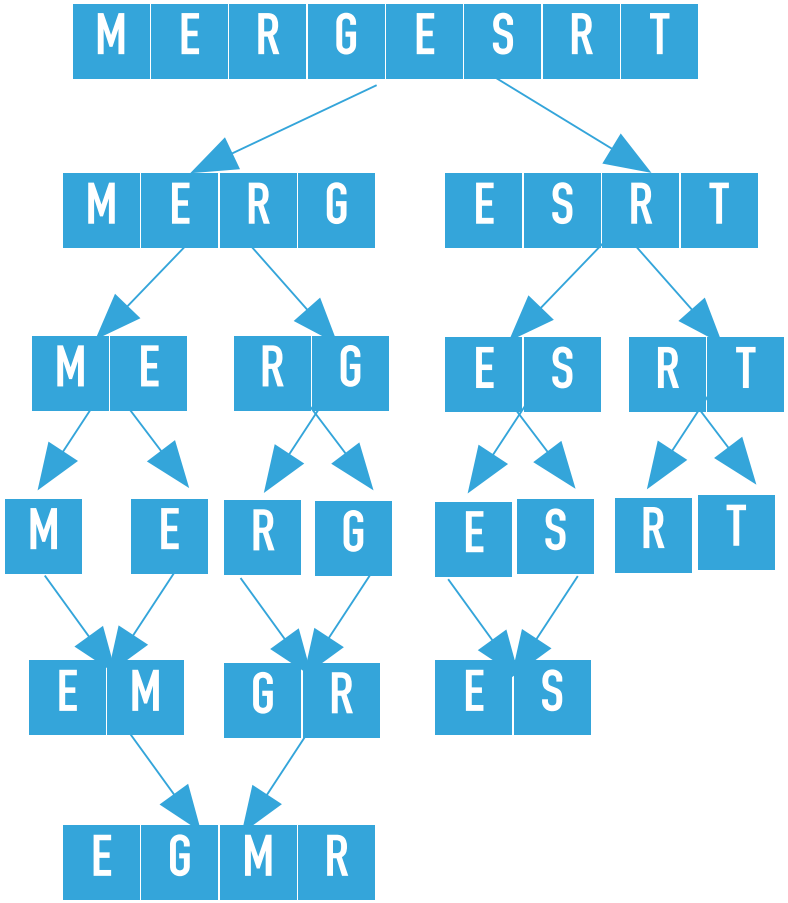

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 6, 7)` calculates the `mid = 6` and calls recursively `sort` on the left subarray, that is `sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 6, 6)`, where `lo = 6`, `hi = 6`.



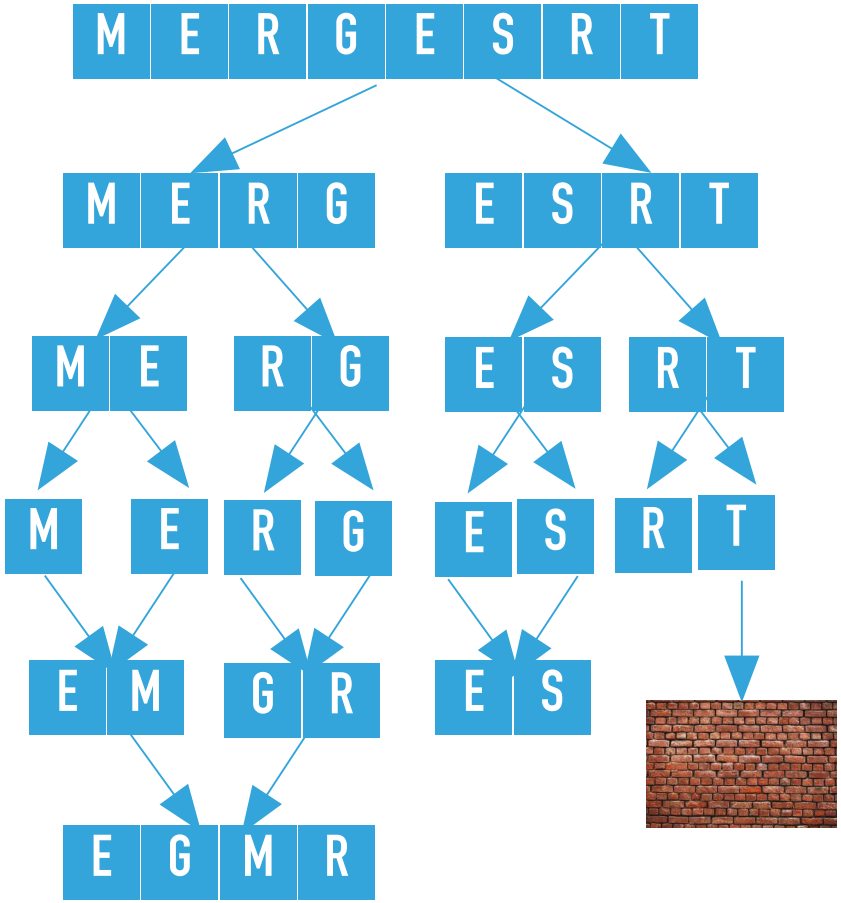

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 6, 6)` finds `hi <= lo` and returns.



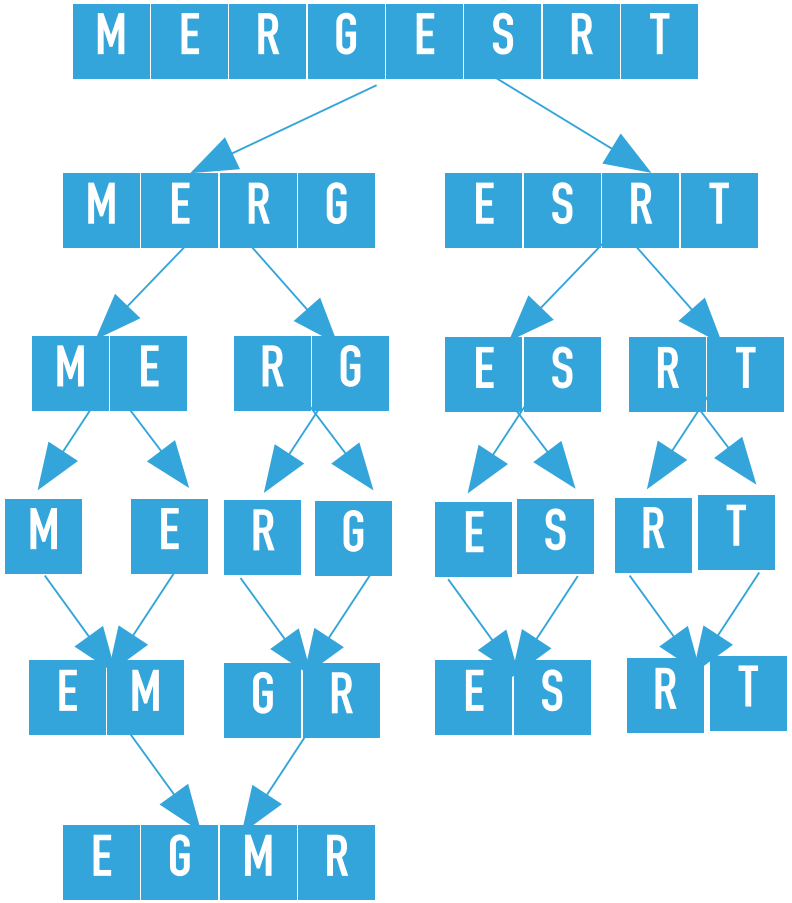

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 6, 7)` calls recursively `sort` on the right subarray, that is `sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 7, 7)`, where `lo = 7`, `hi = 7`



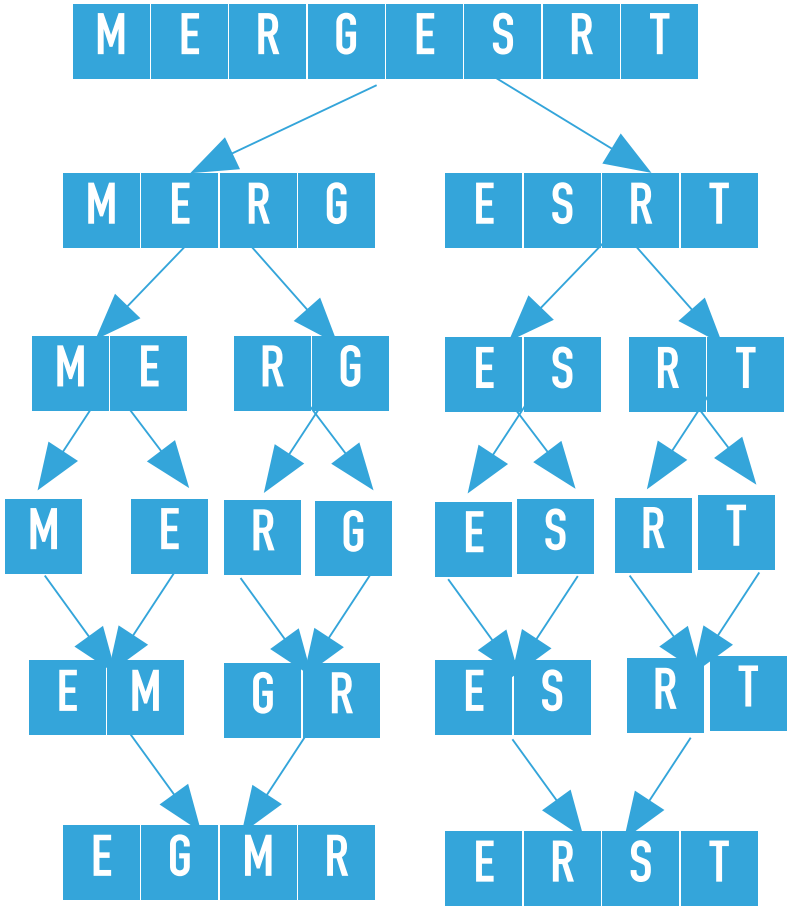

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 7, 7)` finds `hi <= lo` and returns.



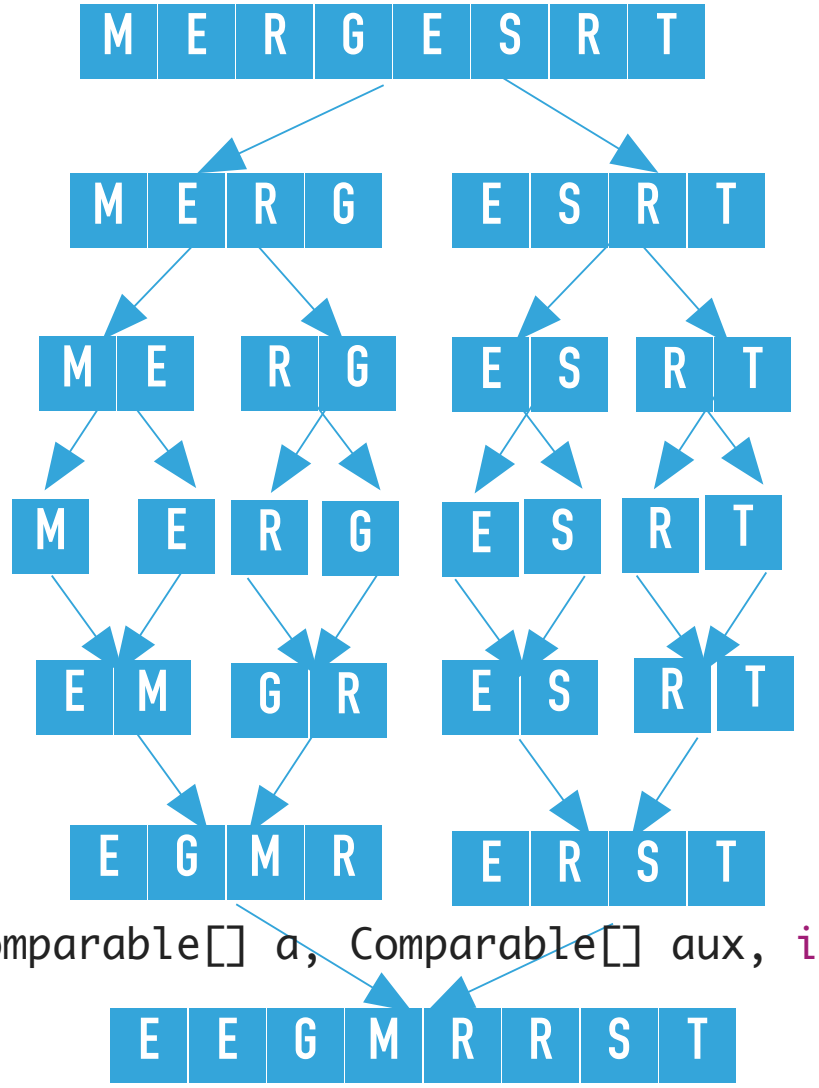

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 6, 7)` merges the two subarrays that is calls `merge([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, null, null], 6, 6, 7)`, where `lo = 6`, `mid = 6`, and `hi = 7`. The resulting partially sorted array is `[E, G, M, R, E, S, R, T]`.




```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, R, T], 4, 7)` merges the two subarrays that is calls `merge([E, G, M, R, E, S, R, T], [E, M, G, R, E, S, R, T], 4, 5, 7)`, where `lo = 4`, `mid = 5`, and `hi = 7`. The resulting partially sorted array is `[E, G, M, R, E, R, S, T]`.




```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

`sort([E, G, M, R, E, R, S, T], [E, M, G, R, E, S, R, T], 0, 7)` merges the two subarrays that is calls `merge([E, G, M, R, E, R, S, T], [E, M, G, R, E, S, R, T], 0, 3, 7)`, where `lo = 0`, `mid = 3`, and `hi = 7`. The resulting sorted array is `[E, E, G, M, R, R, S, T]`.

Practice time

Which of the following subarray lengths will occur when running mergesort on an array of length 10?

A. { 1, 2, 3, 5, 10 }
B. { 2, 4, 6, 8, 10 }
C. { 1, 2, 5, 10 }
D. { 1,2,3,4,5,10}

Answer

Which of the following subarray lengths will occur when running mergesort on an array of length 10?

A. { 1, 2, 3, 5, 10 }

Good algorithms are better than supercomputers

- Your laptop executes 10^8 comparisons per second
- A supercomputer executes 10^{12} comparisons per second

	Insertion sort			Mergesort		
Computer	**Thousand inputs**	**Million inputs**	**Billion inputs**	**Thousand inputs**	**Million inputs**	**Billion inputs**
Home	Instant	2 hours	300 years	instant	1 sec	15 min
Supercomputer	Instant	1 second	1 week	instant	instant	instant

Analysis of comparisons

- We will assume that that n is a power of 2 ($n = 2^k$, where $k = log_2 n$) and the number of comparisons $T(n)$ to sort an array of length n with merge sort satisfies the recurrence:
  - $T(n) = T(n/2) + T(n/2) + (n - 1) = O(n \log n)$

- Number of array accesses (rather than exchanges, here) is also $O(n \log n)$.

Mergesort uses $\leq n \log n$ compares to sort an array of length n

If $n = 4$, 2 levels

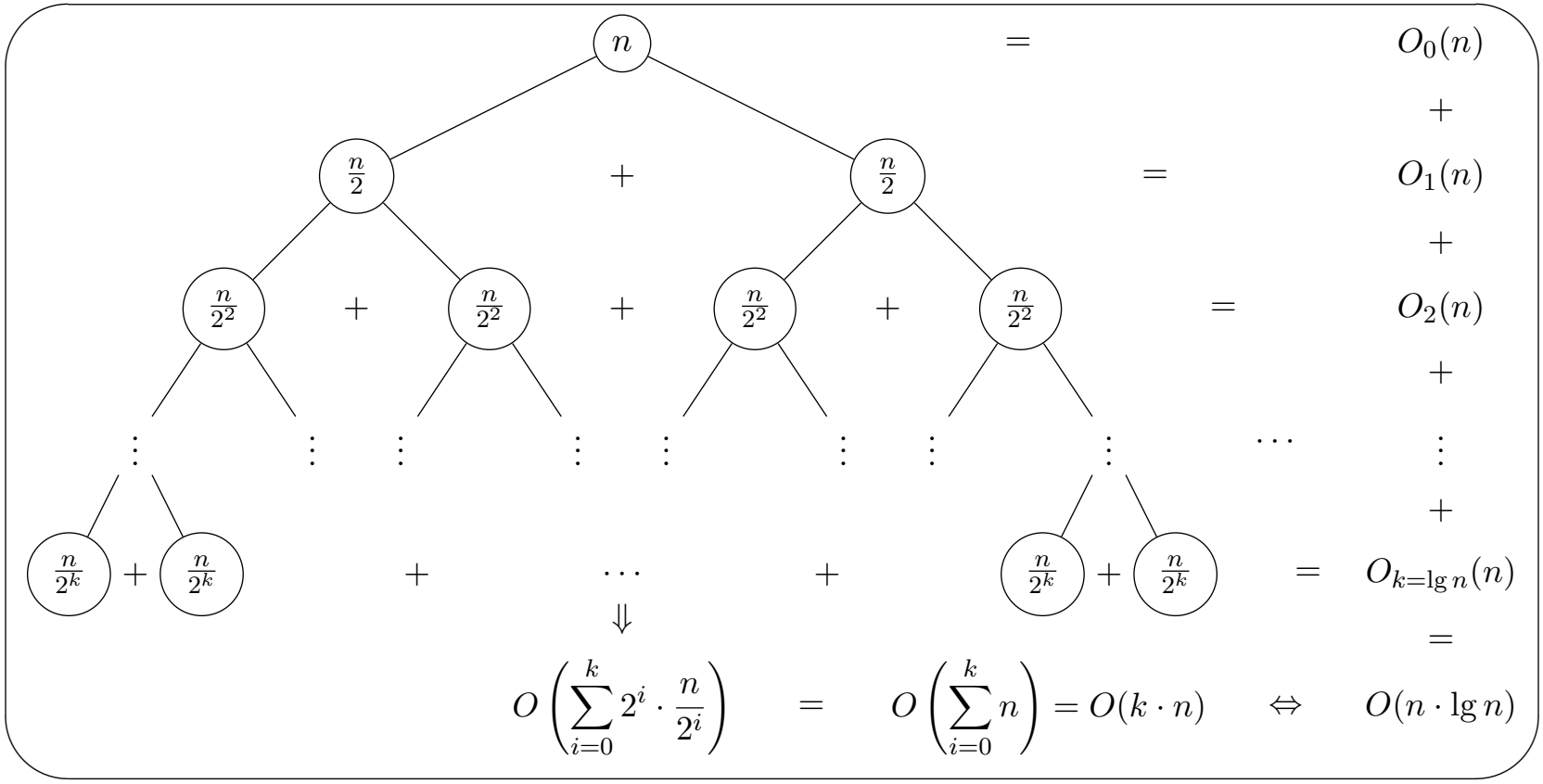
If $n = 8$, 3 levels

If $n = 16$, 4 levels

...

If $n = 2^k$, k levels,

or $k = log_2 n$



http://www.texample.net/media/tikz/examples/PDF/merge-sort-recursion-tree.pdf

Any algorithm with the same structure takes $n \log n$ time

```
public static void f(int n) {
     if (n == 0)
         return;
     f(n/2);
     f(n/2);
     linear(n);
}
```

Mergesort basics

- Auxiliary memory is proportional to n, as `aux[]` needs to be of length n for the last merge.
- At its simplest form, merge sort is not an in-place algorithm.
- There are modifications for halting the size of the auxiliary array but in-place merge is very hard.
- Stable: Look into `merge()`, if equal keys, it takes them from the left subarray.
  - So is insertion sort, but not selection sort.

Practical improvements for Mergesort

- Use insertion sort for small subarrays.
- Stop if already sorted.
- Eliminate the copy to the auxiliary array by saving time (not space).

```
private static void sort(Comparable[] src, Comparable[] dst, int lo, int hi) {
   if (hi <= lo + 7) {
        insertionSort(dst, lo, hi);
        return;
    }

    int mid = lo + (hi - lo) / 2;
    sort(dst, src, lo, mid);
    sort(dst, src, mid+1, hi);

    if (!less(src[mid+1], src[mid])) {
      for (int i = lo; i <= hi; i++) dst[i] = src[i];
       return;
    }

    merge(src, dst, lo, mid, hi);
}
```

- For years, Java used this version to sort Collections of objects.

Sorting: the story so far

	In place	Stable	Best	Average	Worst	Remarks
Selection	X		$O(n^2)$	$O(n^2)$	$O(n^2)$	n exchanges
Insertion	X	X	$O(n)$	$O(n^2)$	$O(n^2)$	Use for small arrays or partially
Merge sort		X	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Guaranteed performance; stable

Lecture 13: Mergesort

- Mergesort

Readings:

- Textbook:
  - Chapter 2.2 (pages 270–277)
- Website:
  - Mergesort: https://algs4.cs.princeton.edu/22mergesort/
  - Code: https://algs4.cs.princeton.edu/22mergesort/Merge.java.html

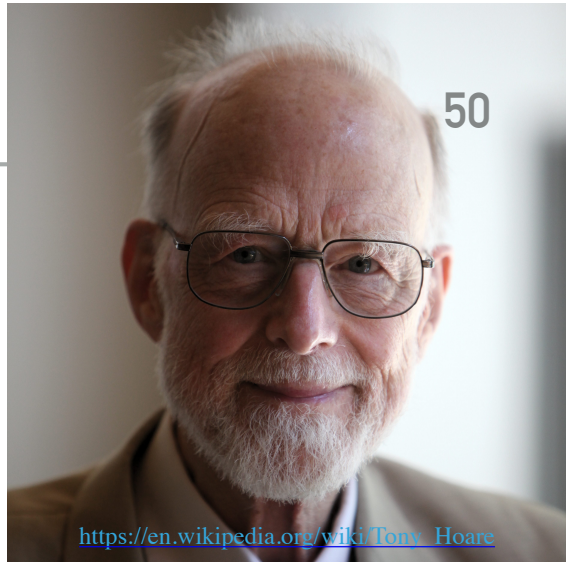
Practice Problems:

- 2.2.1-2.2.2, 2.2.11

TEXT

Lecture 14: Quicksort

- This week's assignment
- Quicksort

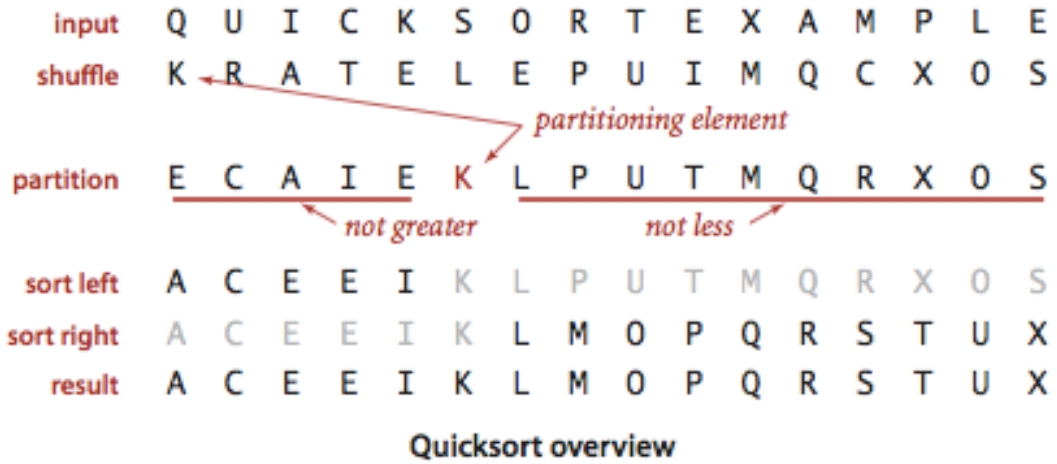


Basics

- Divide-and-conquer method
- Invented by Sir Tony Hoare in 1959.
  - Wanted to sort words before looking them up in dictionary.
  - Came up with quicksort but did not know how to implement it.
  - Learned Algol 60 and recursion and implemented it.
  - Won the 1980 Turing Award (also invented null - and regretted it).
- Bob Sedgewick (author of our textbook) refined and analyzed many versions of quicksort.

Algorithm sketch

- Shuffle the array.
- Partition so that, for some pivot `j`:
  - Entry `a[j]` is in place.
  - There is no larger entry to the left of `j`.
  - No smaller entry to the right of `j`.
- Sort each subarray recursively.



Quicksort overview

Quicksort Code

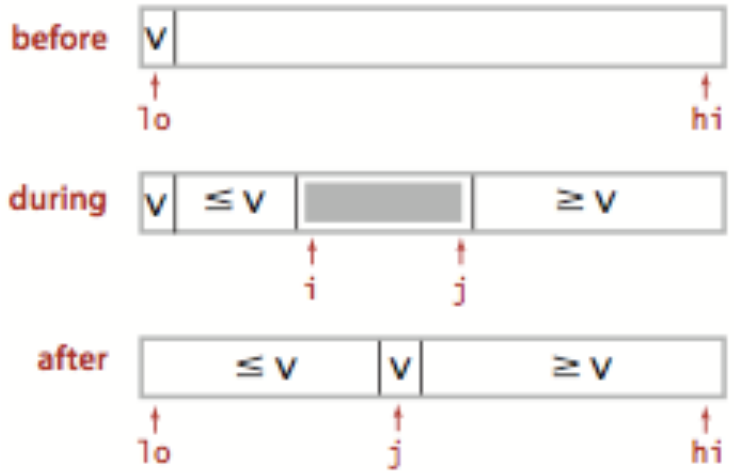
```
// quicksort the subarray from a[lo] to a[hi]
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}

/**
* Rearranges the array in ascending order, using the natural order.
* @param a the array to be sorted
*/
public static void sort(Comparable[] a) {
    StdRandom.shuffle(a);
    sort(a, 0, a.length - 1);
}
```

Hoare's partition scheme (faster than Lomuto's you will see in CS140)

- Partition the subarray `a[lo…hi]` so that `a[lo…j-1]<=a[j]<=a[j+1…hi]`
- Start with pivot at `lo`, pointer `i` at `lo` and pointer `j` at `hi+1`.
- Repeat the following until pointers `i` and `j` cross:
  - Scan `i` from left to right as long as `a[lo]>a[i]`.
    - (i.e. stop when you find a key that is greater than (or equal to) the pivot)
  - Scan `j` from right to left as long as `a[lo]<a[j]`.
    - (i.e. stop when you find a key that is less than (or equal to) the pivot)
  - Exchange `a[i]` with `a[j]`.
- Exchange `a[lo]` and `a[j]`. Return `j`.

Partition Code

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
    while (true) {

        // find item greater than (or equal to) v to swap
        while (less(a[++i], v)) {
            if (i == hi) break;
        }

        // find item smaller than (or equal to) v to swap
        while (less(v, a[--j])) {
            if (j == lo) break; // redundant since a[lo] acts as sentinel
        }

        // check if pointers cross
        if (i >= j) break;

        exch(a, i, j);
    }

    // put partitioning item v at a[j]
    exch(a, lo, j);

    // now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
    return j;
}
```



Quicksort partitioning overview

Quicksort Example - Sort [Q,U,I,C,K,S,O,R,T,E,X,A,M,P,L,E]

```
public static void sort(Comparable[] a) {
    StdRandom.shuffle(a);
    sort(a, 0, a.length - 1);
}

// quicksort the subarray from a[lo] to a[hi]
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Shuffling resulted to:

K,R,A,T,E,L,E,P,U,I,M,Q,C,X,O,S

Call partition with lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

lo hi

Partition Example, lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
i																j

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
	i															j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
	i														j	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo															hi
	i													j	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
	i												j			

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
	i											j				

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
	i											j				

```
exch(a, i, j);
```

Partition Example, lo=0, hi=15

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
	i											j				

```
exch(a, i, j);
```

Partition Example, lo=0, hi=15

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
		i										j				

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=15

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
			i									j				

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=15

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
			i								j					

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
			i							j						

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
			i						j							

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
			i						j							

```
exch(a, i, j);
```

Partition Example, lo=0, hi=15

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
			i						j							

```
exch(a, i, j);
```

Partition Example, lo=0, hi=15

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
				i					j							

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=15

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
					i				j							

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=15

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
					i			j								

```
// find greater than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
					i		j									

```
// find greater than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
					i	j										

```
// find greater than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
					i	j										

```
exch(a, i, j);
```

Partition Example, lo=0, hi=15

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
					i	j										

```
exch(a, i, j);
```

Partition Example, lo=0, hi=15

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo															hi
						i,j									

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=15

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo															hi
					j	i									

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=15

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo															hi	
					j	i										

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=0, hi=15

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo															hi
					j	i									

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=0, hi=15

E	C	A	I	E	**K**	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo															hi
					j	i									

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

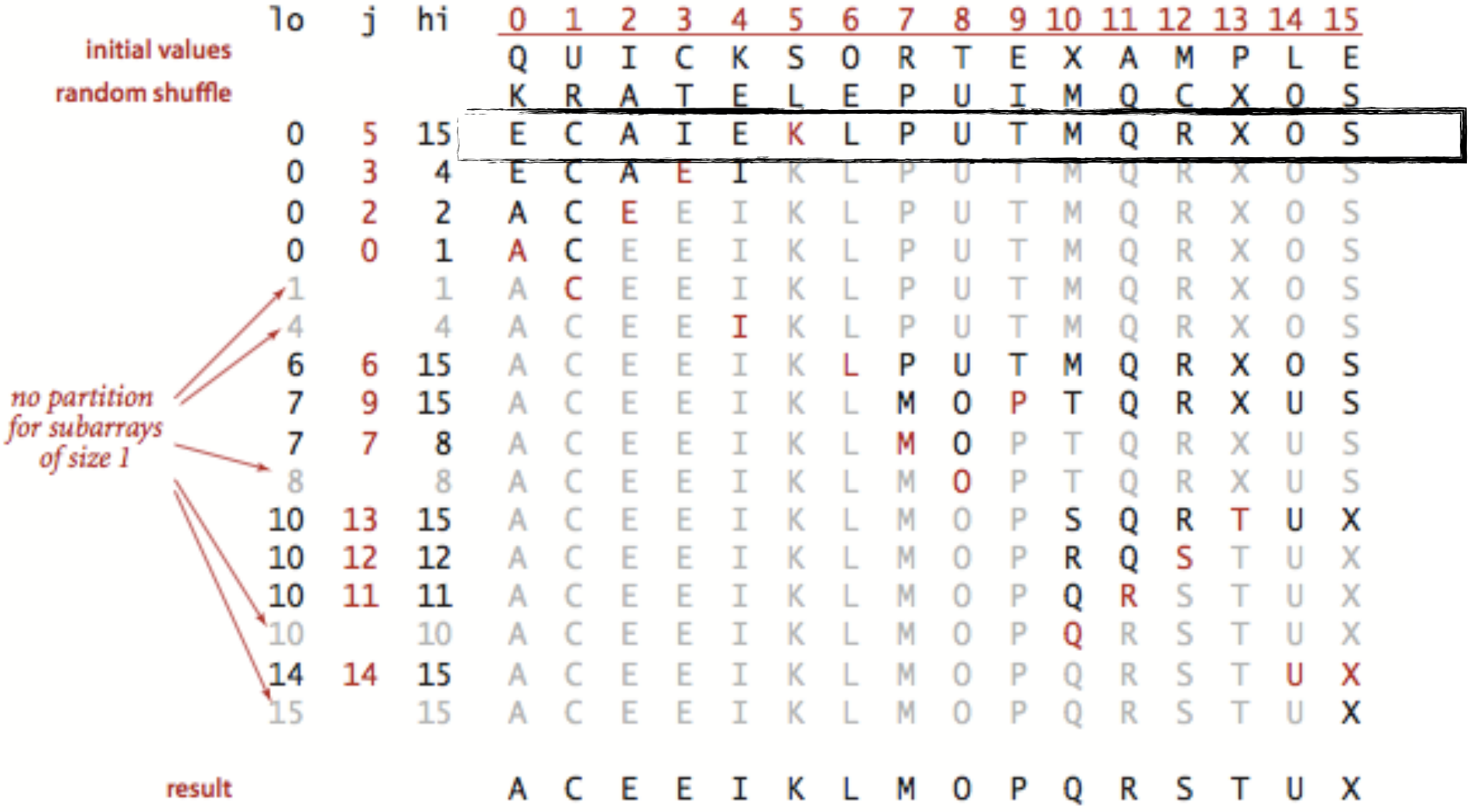
Partition for lo=0, hi=15 is complete. j=5

Call sort recursively on left subarray with lo=0, hi =4

```
private static void sort(Comparable[] a,
int lo, int hi) {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=0, hi=4

E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
i					j											

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=0, hi=4

E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
	i				j											

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=4

E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo				hi											
		i			j										

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=4

E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo				hi											
			i		j										

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=4

E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
			i	j												

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=4

E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
			i	j												

```
exch(a, i, j);
```

Partition Example, lo=0, hi=4

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo				hi											
			i	j											

```
exch(a, i, j);
```

Partition Example, lo=0, hi=4

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
				i,j												

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=4

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
			j	i												

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=4

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
			j	i												

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=0, hi=4

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
			j	i												

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=0, hi=4

E	C	A	**E**	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo				hi												
			j	i												

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

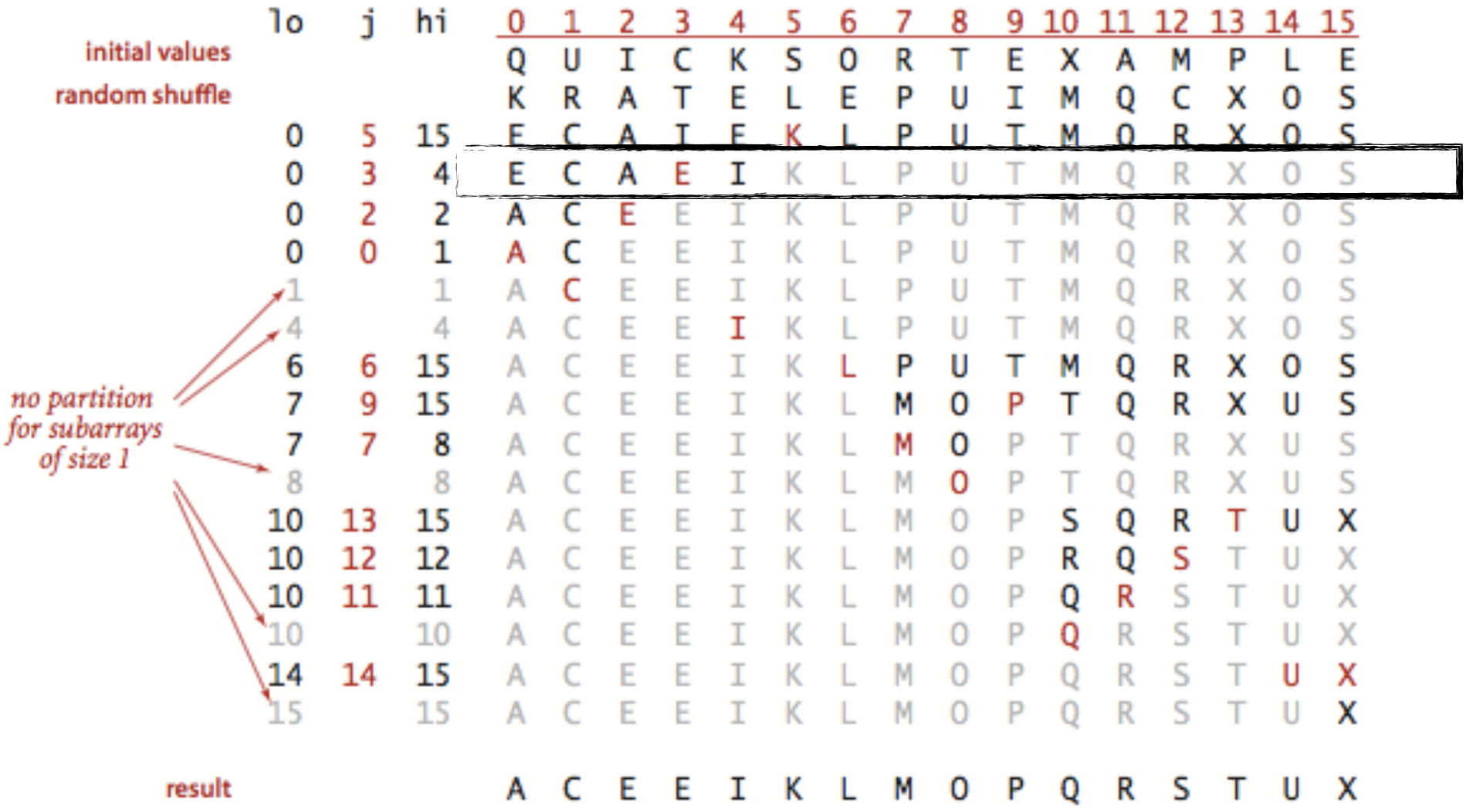
Partition for lo=0, hi=4 is complete. j=3

Call sort recursively on left subarray with lo=0, hi =2

```
private static void sort(Comparable[] a,
int lo, int hi) {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=0, hi=2

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo		hi														
i			j													

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=0, hi=2

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo		hi													
	i		j												

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=2

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo		hi													
		i	j												

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=2

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo		hi														
		i,j														

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=2

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo		hi														
		i,j														

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=0, hi=2

E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo		hi														
		i,j														

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=0, hi=2

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo		hi													
		i,j													

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

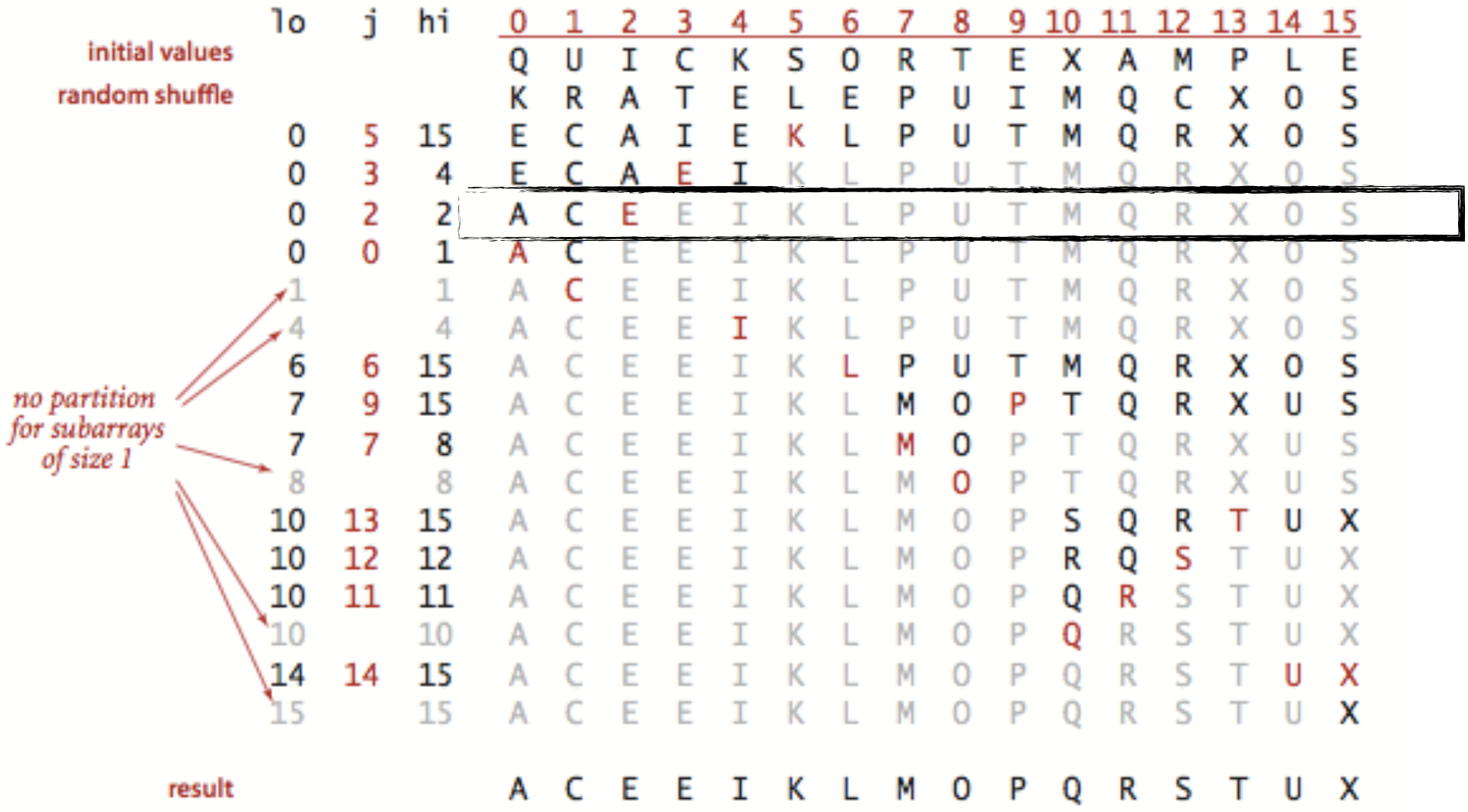
Partition for lo=0, hi=2 is complete. j=2

Call sort recursively on left subarray with lo=0, hi =1

```
private static void sort(Comparable[] a,
int lo, int hi) {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=0, hi=1

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo	hi															
i		j														

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=0, hi=1

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo	hi															
	i	j														

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=0, hi=1

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo	hi															
	i,j															

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=1

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo	hi															
j	i															

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=0, hi=1

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo	hi															
j	i															

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=0, hi=1

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
lo	hi															
j	i															

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=0, hi=1

A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo	hi														
j	i														

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition for lo=0, hi=1 is complete. j=0

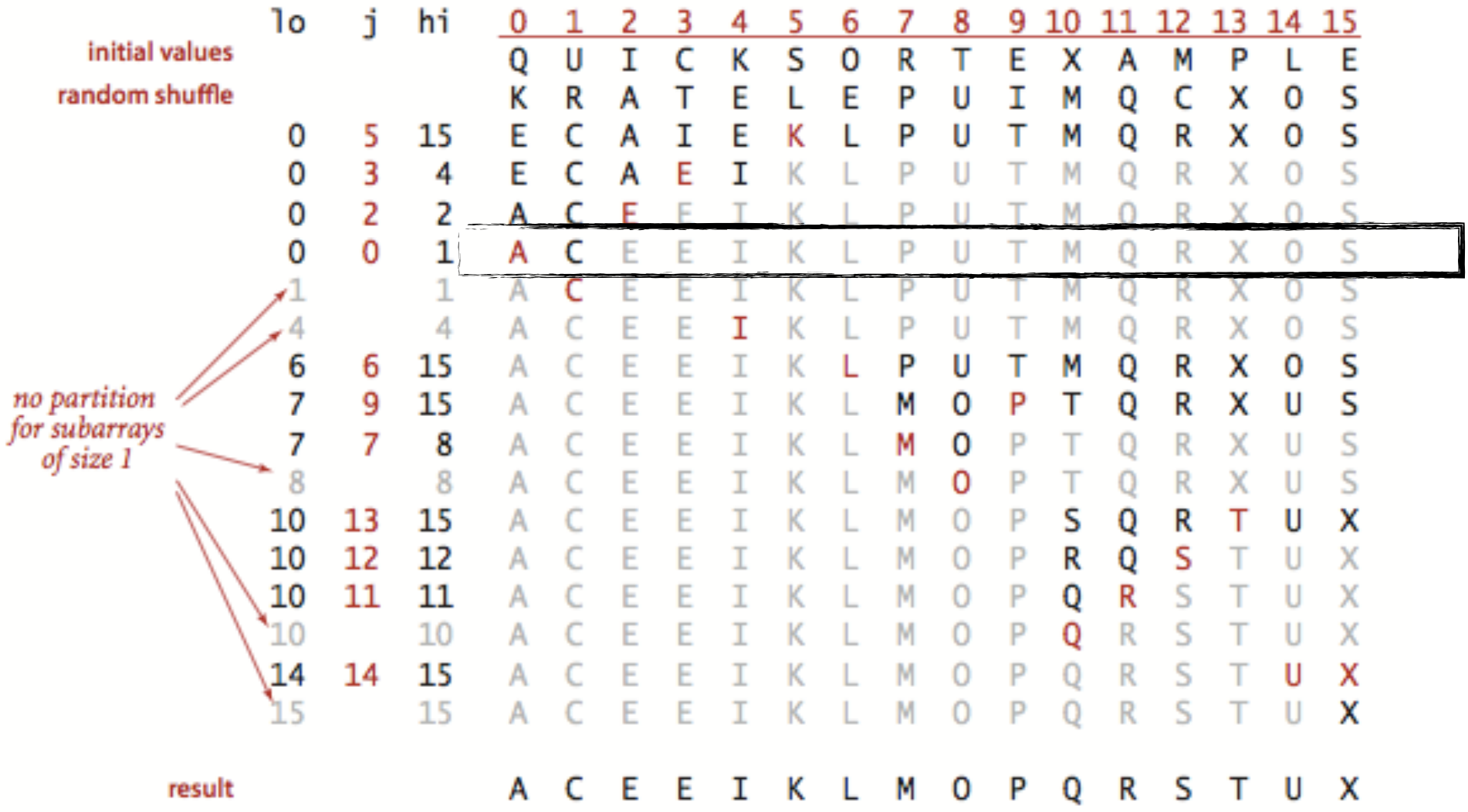
No partition for subarrays of size 1.

Call sort recursively on right subarray with lo=6, hi =15

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
						i										j

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i									j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i								j	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i							j		

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i						j			

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i					j				

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i				j					

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i			j						

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i		j							

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i	j								

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
							i,j									

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
						j	i									

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
						j	i									

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
						lo									hi	
						j	i									

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=6, hi=15

A	C	E	E	I	K	**L**	P	U	T	M	Q	R	X	O	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						lo									hi
						j	i								

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition for lo=6, hi=15 is complete. j=6

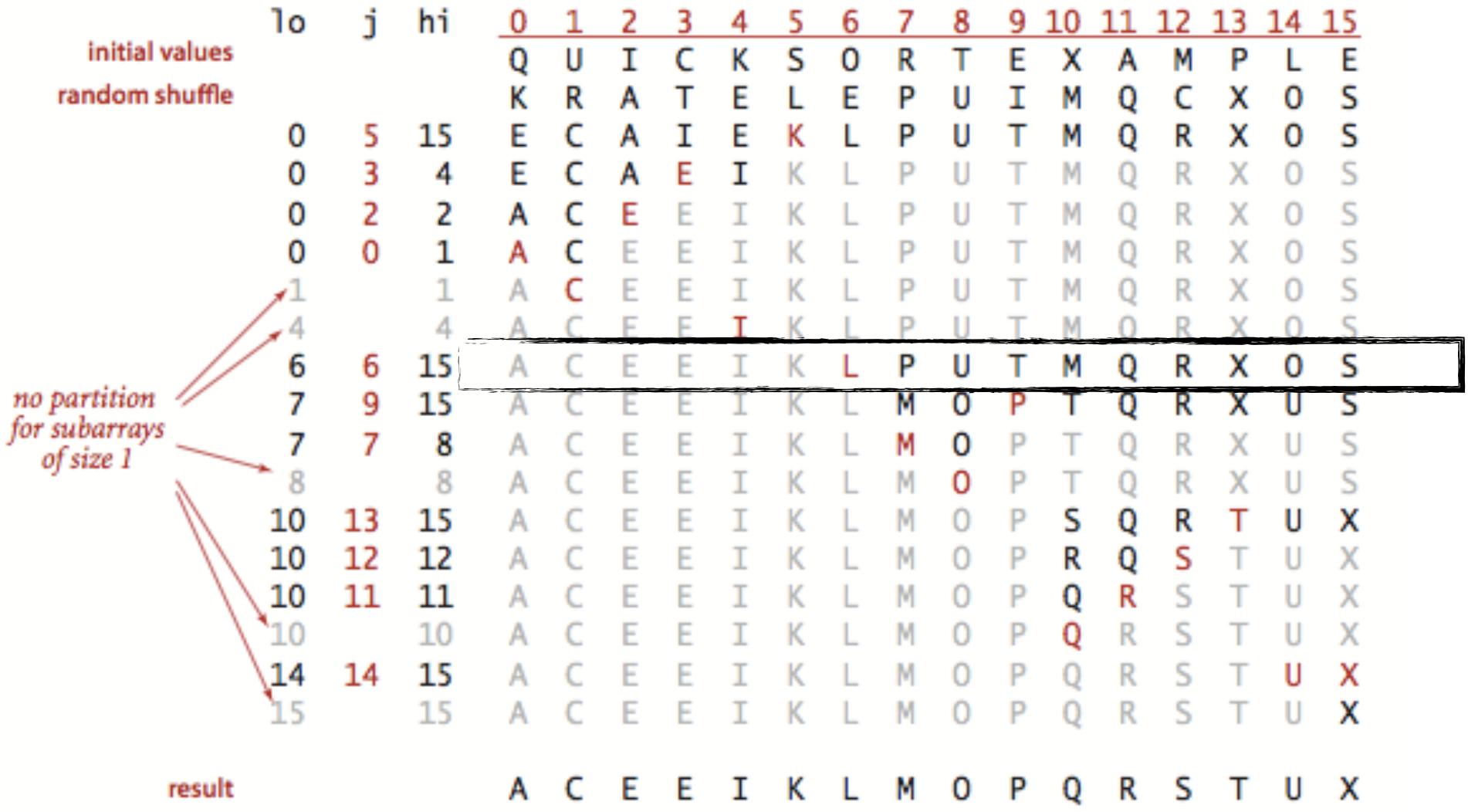
No partition for subarrays of size 1.

Call sort recursively on right subarray with lo=7, hi =15

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

Quicksort Trace

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```



Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
							i									j

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
								i								j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
								i							j	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
								i						j		

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	U	T	M	Q	R	X	O	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
								i						j		

```
exch(a, i, j);
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	T	M	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
								i						j		

```
exch(a, i, j);
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	T	M	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									i					j		

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	T	M	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									i				j			

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	T	M	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									i			j				

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	T	M	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									i		j					

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	T	M	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									i	j						

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	T	M	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									i	j						

```
exch(a, i, j);
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	M	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									i	j						

```
exch(a, i, j);
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	M	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
										i,j						

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	M	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									j	i						

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	M	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									j	i						

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	**P**	O	M	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									j	i						

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=7, hi=15

A	C	E	E	I	K	L	M	O	**P**	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo								hi	
									j	i						

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

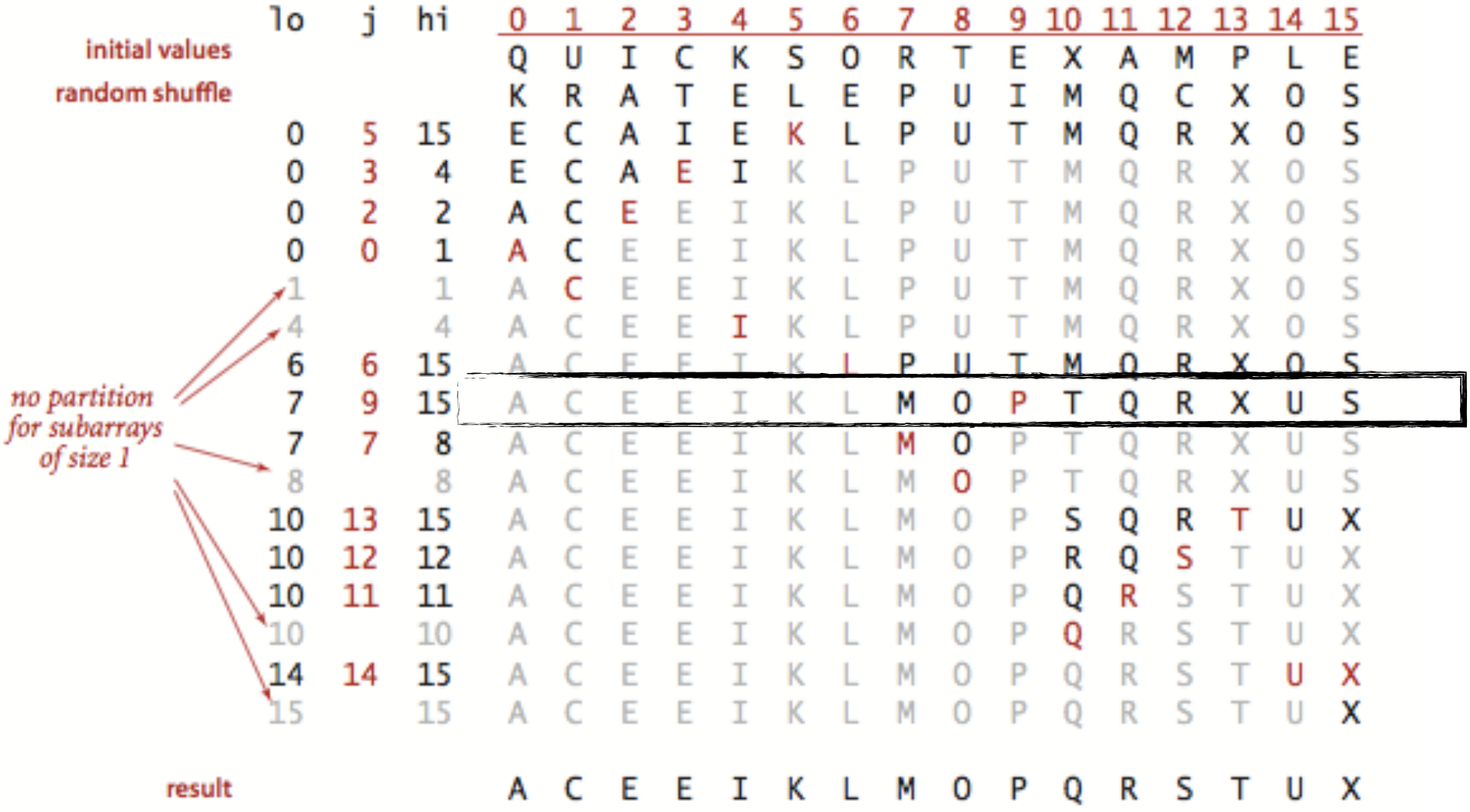
Partition for lo=6, hi=15 is complete. j=9

Call sort recursively on left subarray with lo=7, hi = 8

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=7, hi=8

A	C	E	E	I	K	L	**M**	O	P	T	Q	R	X	U	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							lo	hi							
							i		j						

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=7, hi=8

A	C	E	E	I	K	L	**M**	O	P	T	Q	R	X	U	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							lo	hi							
								i	j						

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=7, hi=8

A	C	E	E	I	K	L	**M**	O	P	T	Q	R	X	U	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							lo	hi							
								i,j							

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=8

A	C	E	E	I	K	L	**M**	O	P	T	Q	R	X	U	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							lo	hi							
							j	i							

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=7, hi=8

A	C	E	E	I	K	L	**M**	O	P	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
							lo	hi								
							j	i								

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=7, hi=8

A	C	E	E	I	K	L	**M**	O	P	T	Q	R	X	U	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							lo	hi							
							j	i							

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=7, hi=8

A	C	E	E	I	K	L	**M**	O	P	T	Q	R	X	U	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							lo	hi							
							j	i							

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition for lo=7, hi=8 is complete. j=7

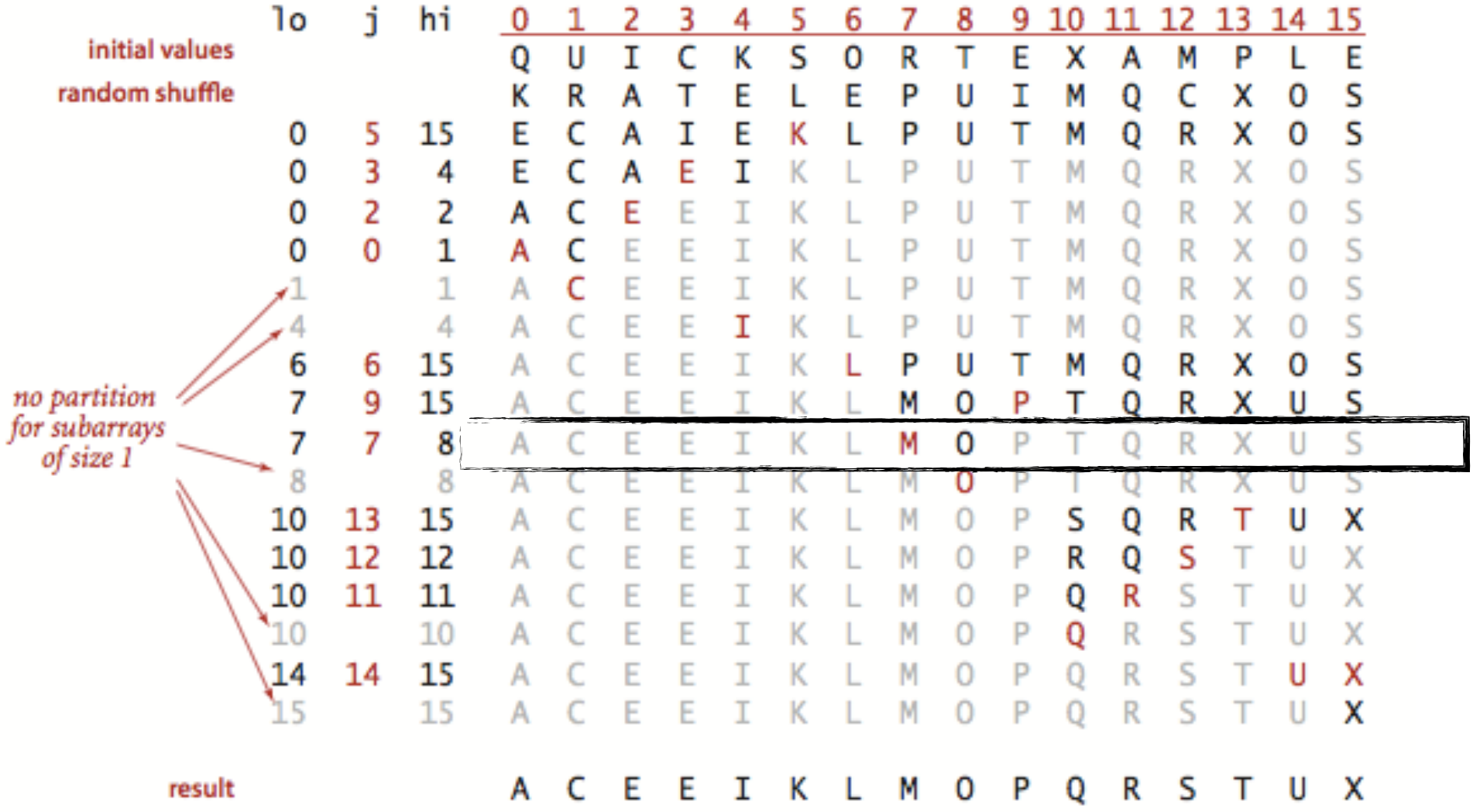
No partition for subarrays of size 1.

Call sort recursively on right subarray with lo=10, hi = 15

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
										i						j

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
											i					j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
												i				j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
													i			j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
													i		j	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	**T**	Q	R	X	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
													i		j	

```
exch(a, i, j);
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	**T**	Q	R	S	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
													i		j	

```
exch(a, i, j);
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	S	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo					hi
														i	j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	S	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo					hi
														i,j	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	S	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
													j	i		

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	**T**	Q	R	S	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
													j	i		

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	T	Q	R	S	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo					hi
													j	i	

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=10, hi=15

A	C	E	E	I	K	L	M	O	P	S	Q	R	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo					hi	
													j	i		

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

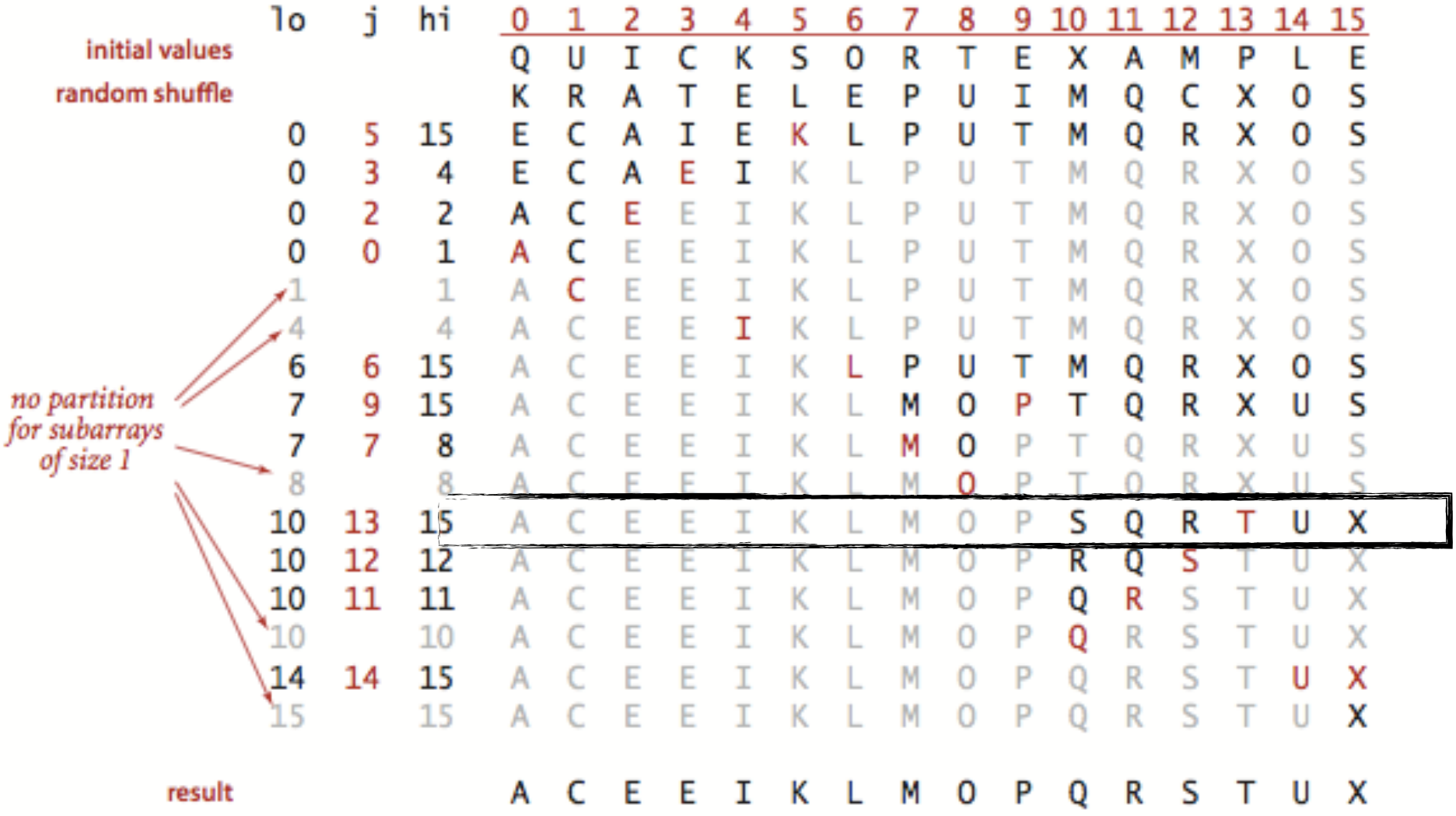
Partition for lo=6, hi=15 is complete. j=13

Call sort recursively on left subarray with lo=10, hi = 12

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

Quicksort Trace

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```



Partition Example, lo=10, hi=12

A	C	E	E	I	K	L	M	O	P	S	Q	R	T	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo		hi			
										i			j		

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=10, hi=12

A	C	E	E	I	K	L	M	O	P	**S**	Q	R	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo		hi				
											i		j			

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=10, hi=12

A	C	E	E	I	K	L	M	O	P	**S**	Q	R	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo		hi				
												i	j			

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=10, hi=12

A	C	E	E	I	K	L	M	O	P	**S**	Q	R	T	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo		hi			
												i,j			

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=10, hi=12

A	C	E	E	I	K	L	M	O	P	**S**	Q	R	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo		hi				
												i,j				

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=10, hi=12

A	C	E	E	I	K	L	M	O	P	**S**	Q	R	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo		hi				
												i,j				

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=10, hi=12

A	C	E	E	I	K	L	M	O	P	R	Q	**S**	T	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo		hi			
												i,j			

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

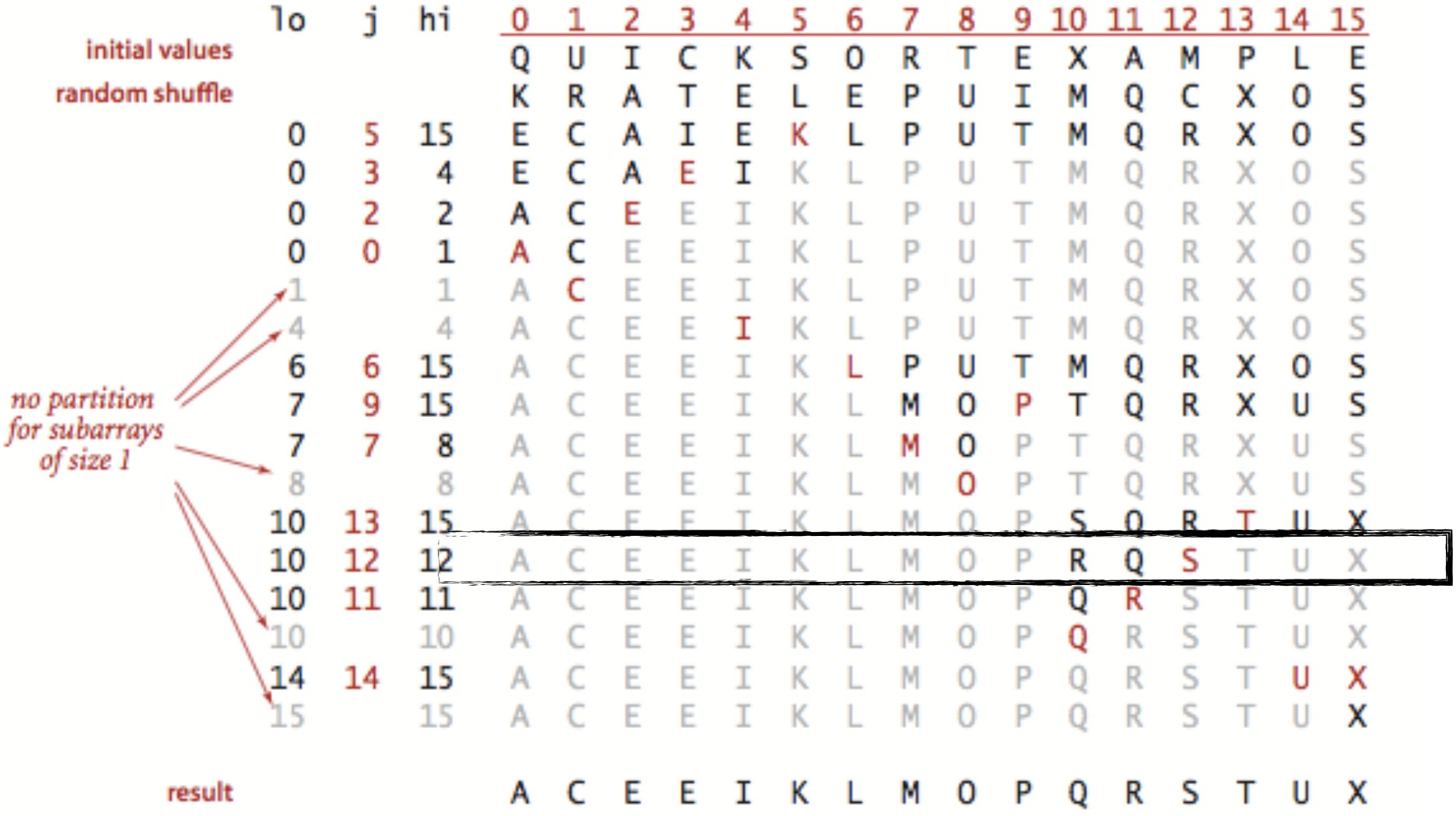
Partition for lo=10, hi=12 is complete. j=12

Call sort recursively on left subarray with lo=10, hi = 11

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=10, hi=11

A	C	E	E	I	K	L	M	O	P	R	Q	S	T	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo	hi				
										i		j			

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=10, hi=11

A	C	E	E	I	K	L	M	O	P	**R**	Q	S	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo	hi					
											i	j				

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=10, hi=11

A	C	E	E	I	K	L	M	O	P	**R**	Q	S	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo	hi					
											i,j					

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=10, hi=11

A	C	E	E	I	K	L	M	O	P	**R**	Q	S	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo	hi					
											i,j					

```
// check if pointers cross
if (i >= j) break;
```

Partition Example, lo=10, hi=11

A	C	E	E	I	K	L	M	O	P	**R**	Q	S	T	U	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
										lo	hi					
											i,j					

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=10, hi=11

A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										lo	hi				
										i,j					

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition for lo=10, hi=11 is complete. j=11

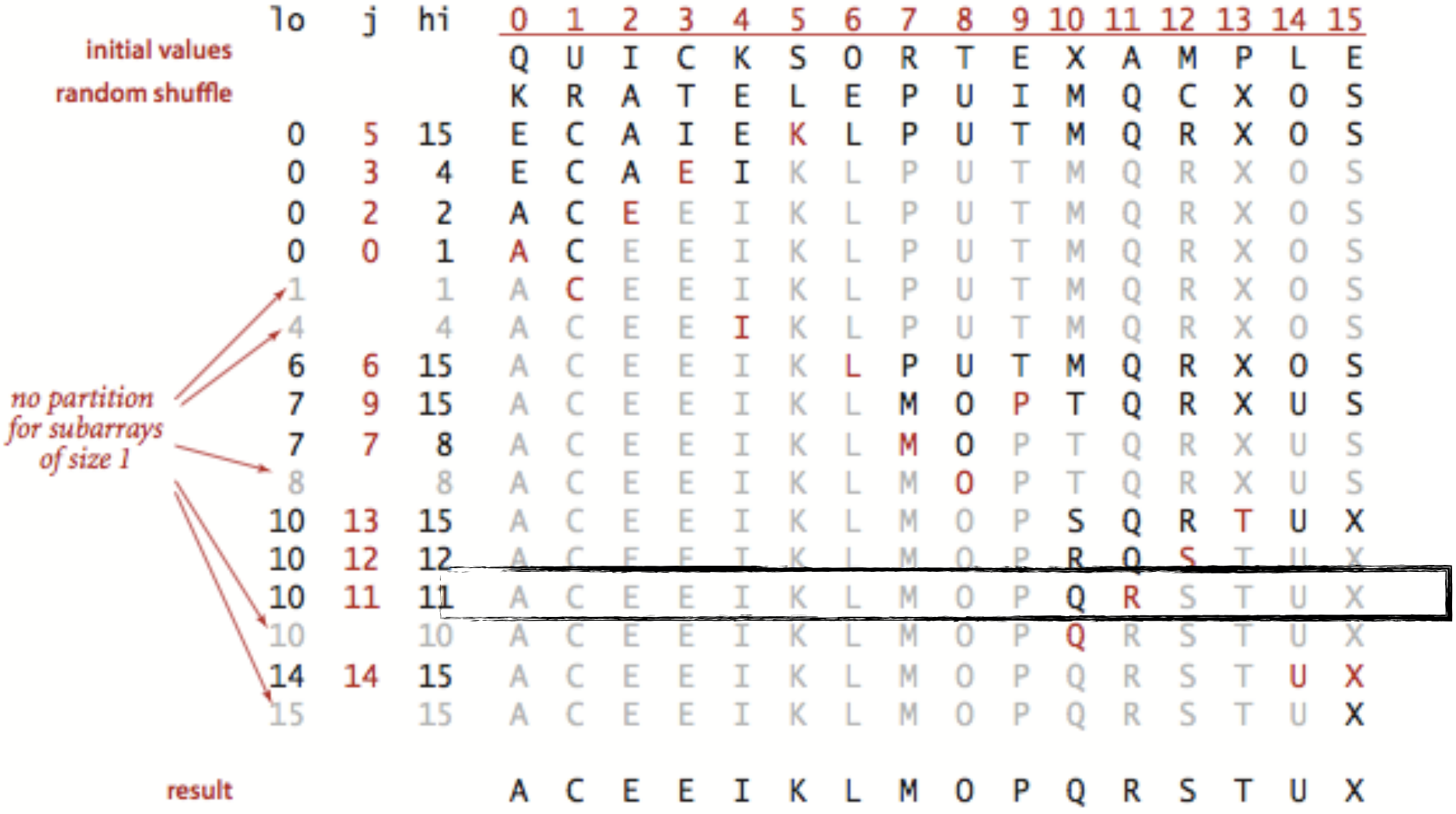
No partition for subarrays of size 1.

Call sort recursively on right subarray with lo=14, hi = 15

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Partition Example, lo=14, hi=15

A	C	E	E	I	K	L	M	O	P	Q	R	S	T	**U**	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
														lo	hi	
														i		j

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi] and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
```

Partition Example, lo=14, hi=15

A	C	E	E	I	K	L	M	O	P	Q	R	S	T	**U**	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
														lo	hi	
															i	j

```
// find greater than (or equal to) v to swap
while (less(a[++i], v)) {
    if (i == hi) break;
}
```

Partition Example, lo=14, hi=15

A	C	E	E	I	K	L	M	O	P	Q	R	S	T	**U**	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
														lo	hi	
															i,j	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=14, hi=15

A	C	E	E	I	K	L	M	O	P	Q	R	S	T	**U**	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
														lo	hi	
														j	i	

```
// find item smaller than (or equal to) v to swap
while (less(v, a[--j])) {
    if (j == lo) break; // redundant since a[lo] acts as sentinel
}
```

Partition Example, lo=14, hi=15

A	C	E	E	I	K	L	M	O	P	Q	R	S	T	**U**	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
														lo	hi	
														j	i	

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition Example, lo=14, hi=15

A	C	E	E	I	K	L	M	O	P	Q	R	S	T	**U**	X	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
														lo	hi	
														j	i	

```
// put partitioning item v at a[j]
exch(a, lo, j);

// now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
return j;
```

Partition for lo=14, hi=15 is complete. j=14

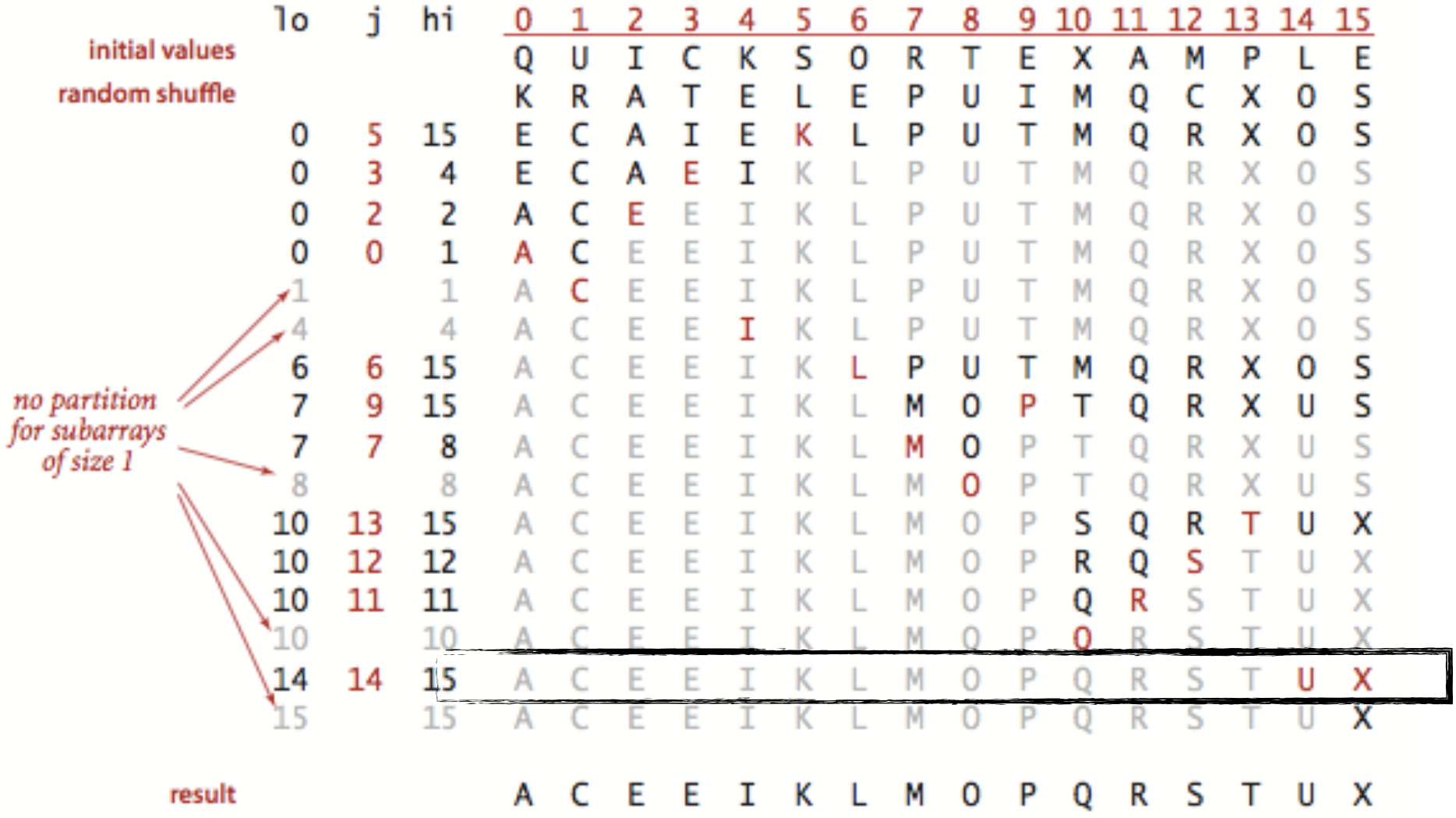
No partition for subarrays of size 1.

Sorting complete

```
private static void sort(Comparable[] a, int lo, int hi) {

        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
```

```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Trace



Great algorithms are better than good ones

- Your laptop executes 10^8 comparisons per second
- A supercomputer executes 10^{12} comparisons per second

	Insertion sort			Mergesort			Quicksort		
Computer	**Thousand inputs**	**Million inputs**	**Billion inputs**	**Thousand inputs**	**Million inputs**	**Billion inputs**	**Thousand inputs**	**Million inputs**	**Billion inputs**
Home	Instant	2 hours	300 years	instant	1 sec	15 min	Instant	0.5 sec	10 min
Supercomputer	Instant	1 second	1 week	instant	instant	instant	instant	instant	Instant

Quicksort analysis: best case

- Quicksort divides everything exactly in half.
- Similar to merge sort.
- Number of compares is $\sim n \log n$.

Quicksort analysis: worst case

- Data are already sorted or we pick the smallest or largest key as pivot.
- Number of compares is $\sim n^2$ - quadratic!
- Extremely unlikely (less likely than the probably that your computer is struck by lightning) if we shuffle and our shuffling is not broken.

Things to remember about quick sort

- 39% more compares than merge sort but in practice it is faster because it does not move data much.
  - If good implementation, even in sorted arrays it can be linearithmic. If not, we end up with quadratic.
- $O(n \log n)$ average, $O(n^2)$ worst, in practice faster than mergesort.
- In-place sorting.
- **Not** stable.

Quicksort practical improvements

- Use insertion sort for small subarrays.
- Best choice of pivot is the median of a small sample.

- For years, Java used quicksort for collections of primitives and mergesort for collections of objects due to stability.
  - Has moved to dual-pivot quick sort (Yaroslavskiy, Bentley, and Bloch, 2009) and timsort (Peters, 1993), respectively.

Sorting: the story so far

Which Sort	In place	Stable	Best	Average	Worst	Remarks
Selection	X		$O(n^2)$	$O(n^2)$	$O(n^2)$	n exchanges
Insertion	X	X	$O(n)$	$O(n^2)$	$O(n^2)$	Use for small arrays or partially ordered
Merge		X	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Guaranteed performance; stable
Quick	X		$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$n \log n$ probabilistic guarantee; fastest in practice

Sorting based on comparisons

- All sorting algorithms we have seen so far and we will see in this class are compare-based.
- No compare-based sorting algorithm can sort n elements in less than $O(n \log n)$ time in the worst case.
  - Proof and proper notation in CS140.

Lecture 14: Quicksort

- This week's assignment
- Quicksort

Readings:

- Textbook:
  - Chapter 2.3 (Pages 288-296)
- Website:
  - Quicksort: https://algs4.cs.princeton.edu/23quicksort/
  - Code: https://algs4.cs.princeton.edu/23quicksort/Quick.java.html

Practice Problems:

- 2.3.1-2.3.4