Recurrences

David Kauchak
cs140
Fall 2022

Administrative

Mentor hours and office hours posted

Assignment 1

Assignment 2 out today (can start after this class)

Divide and Conquer

Divide: Break the problem into smaller sub-problems

Conquer: Solve the sub-problems. Generally, this involves waiting for the problem to be small enough that it is trivial to solve (i.e. 1 or 2 items)

Combine: Given the results of the solved sub-problems, combine them to generate a solution for the complete problem

Divide and Conquer: some thoughts

Often, the sub-problem is the same as the original problem

Dividing the problem in half frequently does the job

May have to get creative about how the data is split

Splitting tends to generate run times with log *n* in them

Divide and conquer

One approach:

- Pretend like you have a working version of your function, but it only works on smaller sub-problems
- If you split up the current problem in some way (e.g. in half) and solved those sub-problems, how could you then get the solution to the larger problem?

Divide and Conquer: Sorting

How should we split the data?

What are the sub-problems we need to solve?

How do we combine the results from these sub-problems?

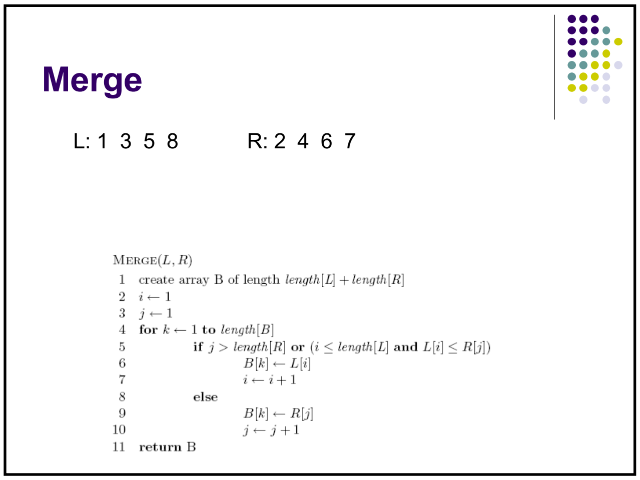
MergeSort

```
MERGE-SORT($A$)
if $length[A] == 1$
        return A
else
        $q \leftarrow \lfloor length[A]/2 \rfloor$
        create arrays $L[1..q]$ and $R[q+1..length[A]]$
        copy $A[1..q]$ to $L$
        copy $A[q+1..length[A]]$ to $R$
        $LS \leftarrow$ MERGE-SORT(L)
        $RS \leftarrow$ MERGE-SORT(R)
        return MERGE(LS, RS)
```

MergeSort: Merge

Assuming L and R are sorted already, merge the two to create a single sorted array

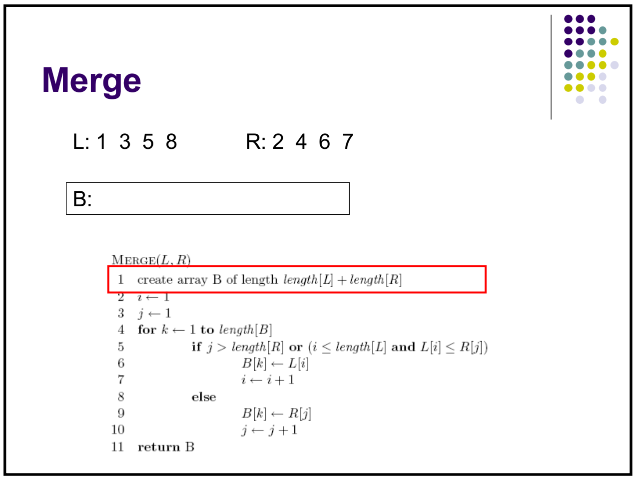
```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
        if $j > length[R]$ or $(i \leq length[L]$ and $L[i] \leq R[j])$
                $B[k] \leftarrow L[i]$
                $i \leftarrow i + 1$
        else
                $B[k] \leftarrow R[j]$
                $j \leftarrow j + 1$
return B
```



Merge

L: 1 3 5 8 R: 2 4 6 7

```
MERGE$(L, R)$
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

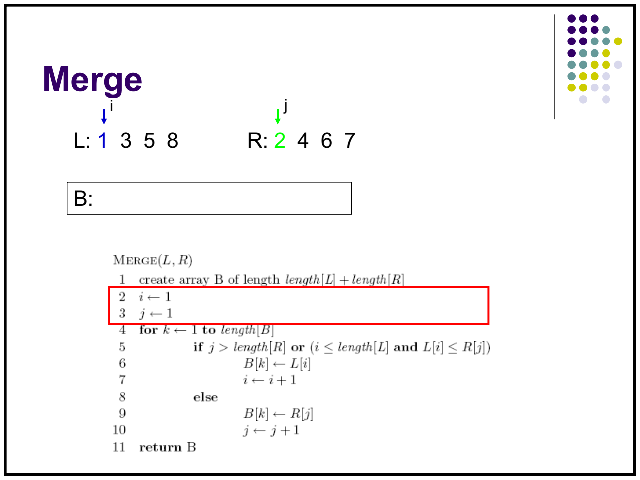


Merge

L: 1 3 5 8 R: 2 4 6 7

B:

```
MERGE$(L, R)$
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

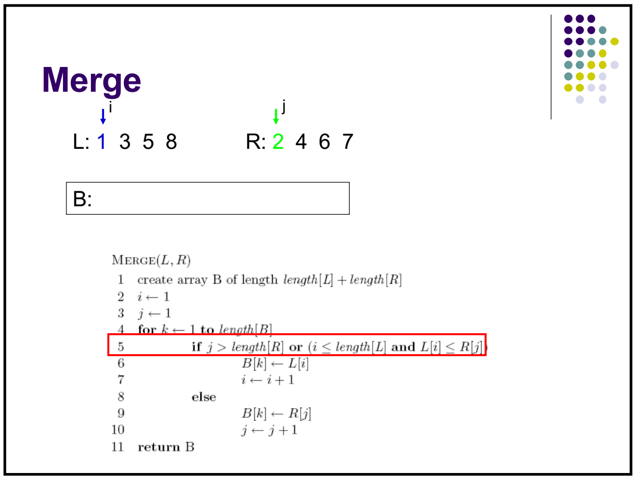


Merge

L: 1 3 5 8 R: 2 4 6 7

B:

```
MERGE$(L, R)$
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

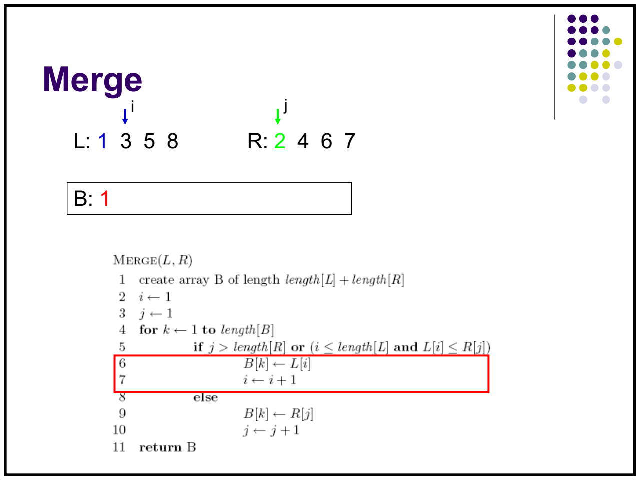


Merge

L: 1 3 5 8 R: 2 4 6 7

B:

```
MERGE$(L, R)$
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

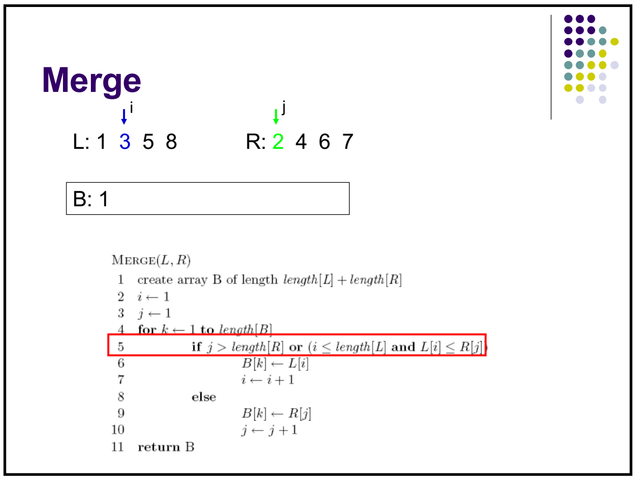


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

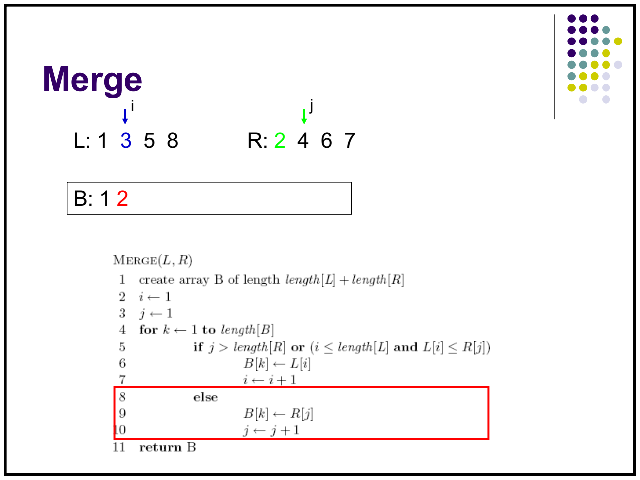


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

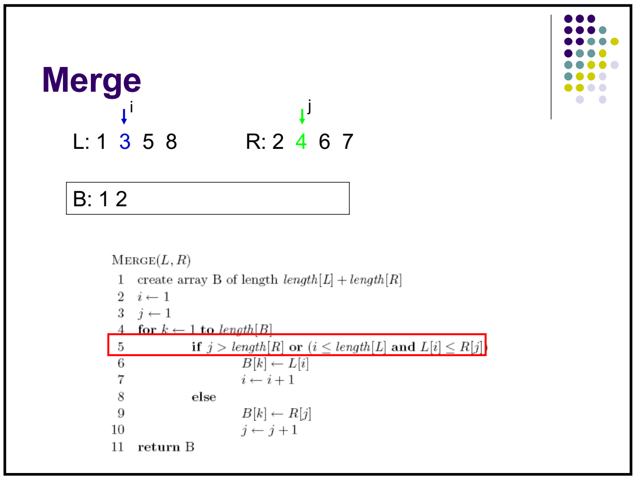


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

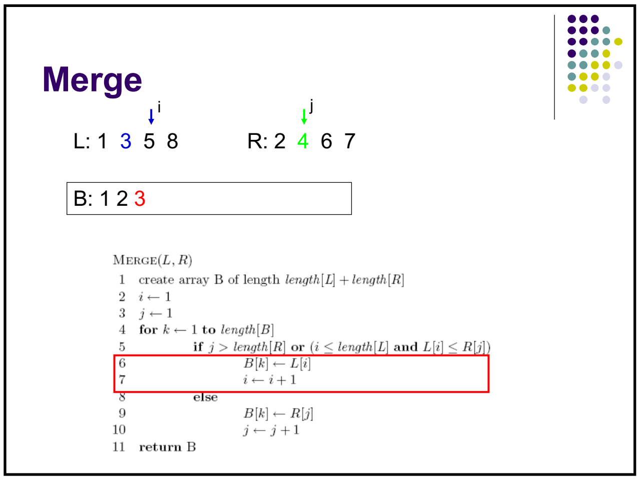


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```



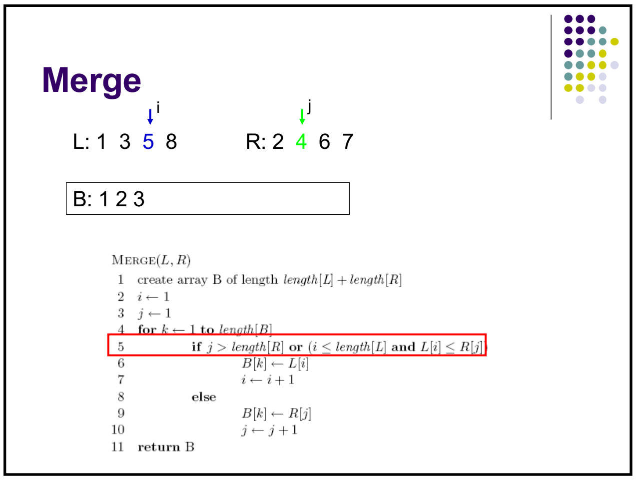
Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \le length[L]$ and $L[i] \le R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

17



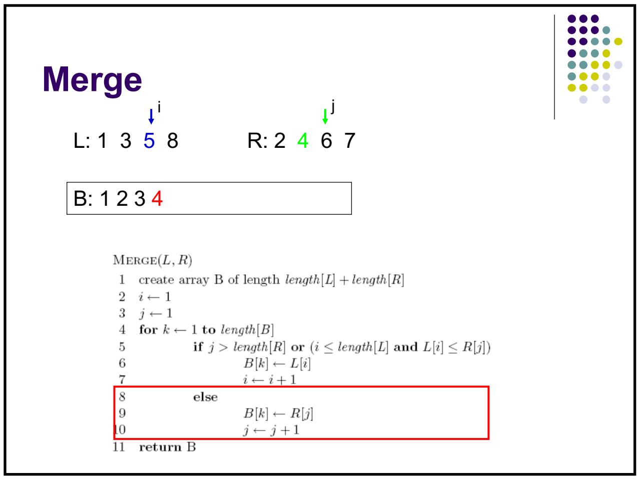
Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \le length[L]$ and $L[i] \le R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

18



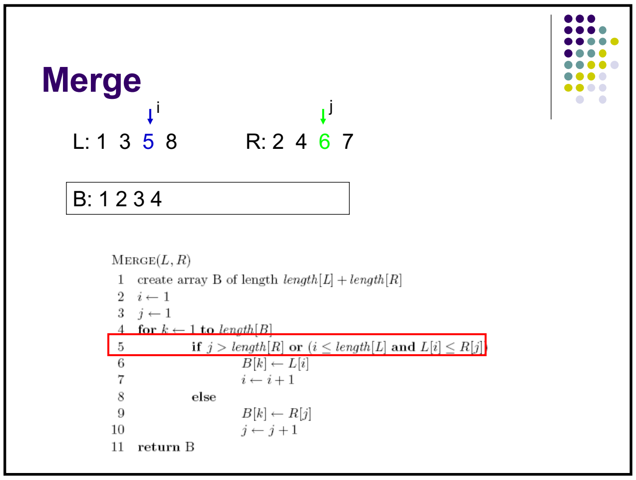
Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3 4

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \le length[L]$ and $L[i] \le R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

19



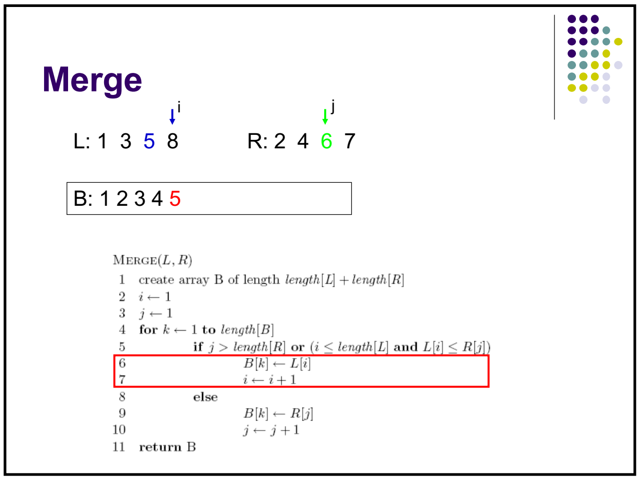
Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3 4

```
MERGE($L, R$)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \le length[L]$ and $L[i] \le R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

20

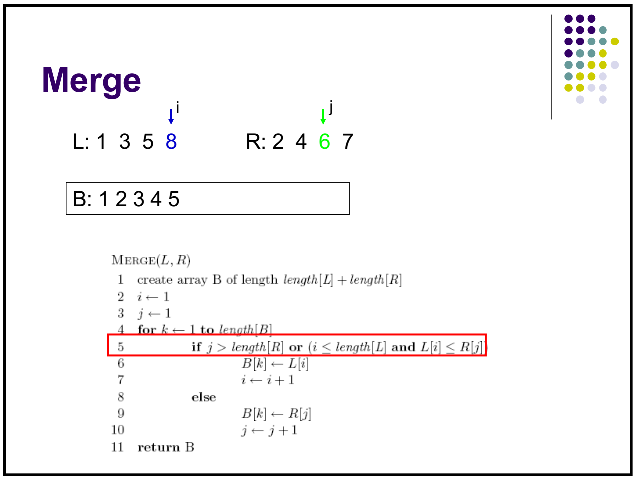


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3 4 5

```
MERGE(L, R)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

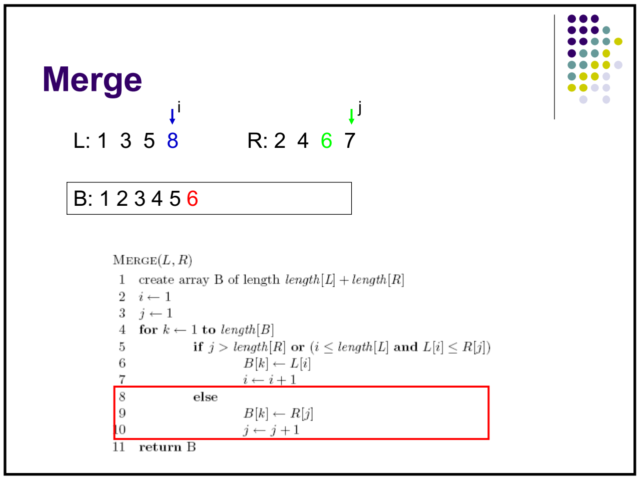


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3 4 5

```
MERGE(L, R)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

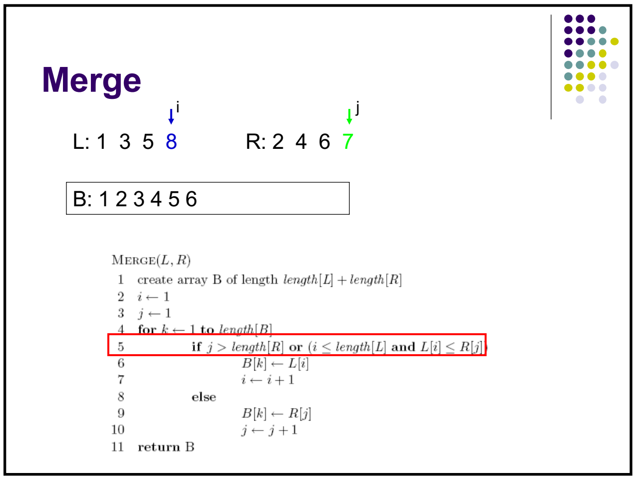


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3 4 5 6

```
MERGE(L, R)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```

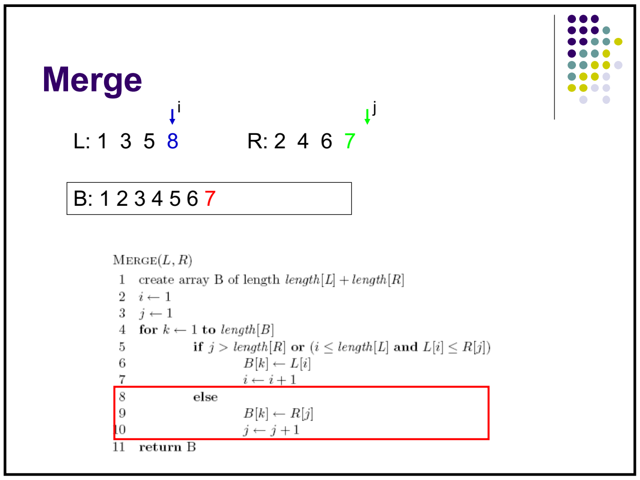


Merge

L: 1 3 5 8   R: 2 4 6 7

B: 1 2 3 4 5 6

```
MERGE(L, R)
create array B of length $length[L] + length[R]$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow 1$ to $length[B]$
    if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
        $B[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $B[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
return B
```



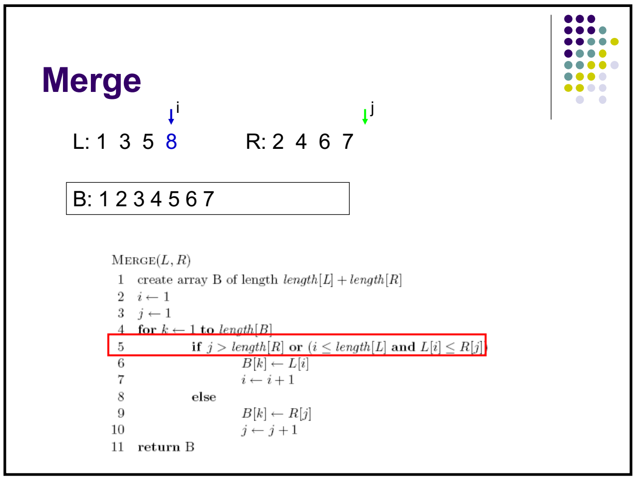
Merge

↓i ↓j

L: 1 3 5 8 R: 2 4 6 7

B: 1 2 3 4 5 6 7

```
MERGE$(L, R)$
  create array B of length $length[L] + length[R]$
  $i \leftarrow 1$
  $j \leftarrow 1$
  for $k \leftarrow 1$ to $length[B]$
        if $j > length[R]$ or $(i \le length[L]$ and $L[i] \le R[j])$
              $B[k] \leftarrow L[i]$
              $i \leftarrow i + 1$
        else
              $B[k] \leftarrow R[j]$
              $j \leftarrow j + 1$
  return B
```



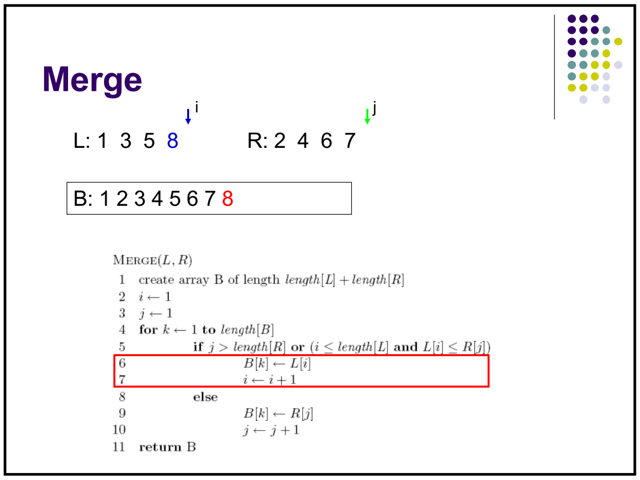
Merge

↓i ↓j

L: 1 3 5 8 R: 2 4 6 7

B: 1 2 3 4 5 6 7

```
MERGE$(L, R)$
  create array B of length $length[L] + length[R]$
  $i \leftarrow 1$
  $j \leftarrow 1$
  for $k \leftarrow 1$ to $length[B]$
        if $j > length[R]$ or $(i \le length[L]$ and $L[i] \le R[j])$
              $B[k] \leftarrow L[i]$
              $i \leftarrow i + 1$
        else
              $B[k] \leftarrow R[j]$
              $j \leftarrow j + 1$
  return B
```



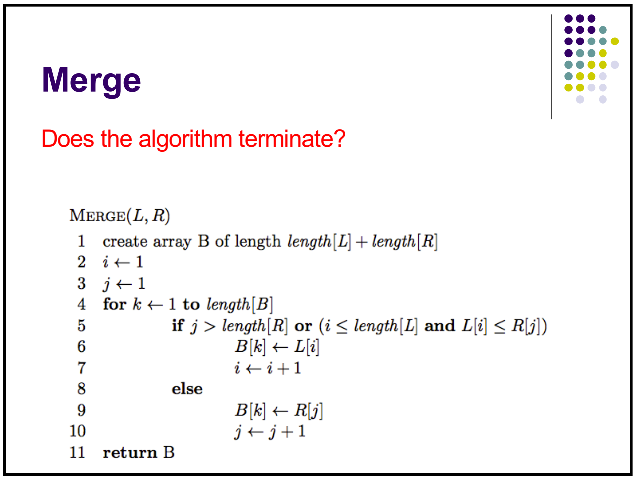
Merge

↓i ↓j

L: 1 3 5 8 R: 2 4 6 7

B: 1 2 3 4 5 6 7 8

```
MERGE$(L, R)$
  create array B of length $length[L] + length[R]$
  $i \leftarrow 1$
  $j \leftarrow 1$
  for $k \leftarrow 1$ to $length[B]$
        if $j > length[R]$ or $(i \le length[L]$ and $L[i] \le R[j])$
              $B[k] \leftarrow L[i]$
              $i \leftarrow i + 1$
        else
              $B[k] \leftarrow R[j]$
              $j \leftarrow j + 1$
  return B
```



Merge

Does the algorithm terminate?

```
MERGE$(L, R)$
  create array B of length $length[L] + length[R]$
  $i \leftarrow 1$
  $j \leftarrow 1$
  for $k \leftarrow 1$ to $length[B]$
        if $j > length[R]$ or $(i \le length[L]$ and $L[i] \le R[j])$
              $B[k] \leftarrow L[i]$
              $i \leftarrow i + 1$
        else
              $B[k] \leftarrow R[j]$
              $j \leftarrow j + 1$
  return B
```

Merge

Is it correct?

Loop invariant:

```
MERGE$(L, R)$
1   create array B of length $length[L] + length[R]$
2   $i \leftarrow 1$
3   $j \leftarrow 1$
4   for $k \leftarrow 1$ to $length[B]$
5           if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
6                   $B[k] \leftarrow L[i]$
7                   $i \leftarrow i + 1$
8           else
9                   $B[k] \leftarrow R[j]$
10                  $j \leftarrow j + 1$
11  return B
```

Merge

Is it correct?

Loop invariant: At the beginning of the **for** loop of lines 4-10 the first k-1 elements of B are the smallest *k-1* elements from L and R in sorted order.

```
MERGE$(L, R)$
1   create array B of length $length[L] + length[R]$
2   $i \leftarrow 1$
3   $j \leftarrow 1$
4   for $k \leftarrow 1$ to $length[B]$
5           if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
6                   $B[k] \leftarrow L[i]$
7                   $i \leftarrow i + 1$
8           else
9                   $B[k] \leftarrow R[j]$
10                  $j \leftarrow j + 1$
11  return B
```

Merge

Running time?

```
MERGE$(L, R)$
1   create array B of length $length[L] + length[R]$
2   $i \leftarrow 1$
3   $j \leftarrow 1$
4   for $k \leftarrow 1$ to $length[B]$
5           if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
6                   $B[k] \leftarrow L[i]$
7                   $i \leftarrow i + 1$
8           else
9                   $B[k] \leftarrow R[j]$
10                  $j \leftarrow j + 1$
11  return B
```

Merge

Running time? $\Theta(n)$ - linear

```
MERGE$(L, R)$
1   create array B of length $length[L] + length[R]$
2   $i \leftarrow 1$
3   $j \leftarrow 1$
4   for $k \leftarrow 1$ to $length[B]$
5           if $j > length[R]$ or ($i \leq length[L]$ and $L[i] \leq R[j]$)
6                   $B[k] \leftarrow L[i]$
7                   $i \leftarrow i + 1$
8           else
9                   $B[k] \leftarrow R[j]$
10                  $j \leftarrow j + 1$
11  return B
```

MergeSort

Running time?

```
MERGE-SORT($A$)
if $length[A] == 1$
    return A
else
    $q \leftarrow \lfloor length[A]/2 \rfloor$
    create arrays $L[1..q]$ and $R[q+1..length[A]]$
    copy $A[1..q]$ to $L$
    copy $A[q+1..length[A]]$ to $R$
    $LS \leftarrow$ MERGE-SORT(L)
    $RS \leftarrow$ MERGE-SORT(R)
    return MERGE(LS, RS)
```

Merge-Sort

Running time?

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + D(n) + C(n) & \text{otherwise} \end{cases}$$

D(n): cost of splitting (dividing) the data

C(n): cost of merging/combining the data

Merge-Sort

Running time?

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + D(n) + C(n) & \text{otherwise} \end{cases}$$

D(n): cost of splitting (dividing) the data - linear Θ(n)

C(n): cost of merging/combining the data – linear Θ(n)

Merge-Sort

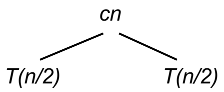
Running time?

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$

Which is?

Merge-Sort

$$T(n)=\begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2)+cn & \text{otherwise}\end{cases}$$

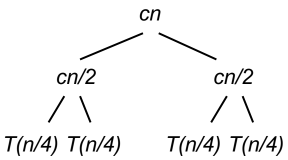


cn

T(n/2) T(n/2)

Merge-Sort

$$T(n)=\begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2)+cn & \text{otherwise}\end{cases}$$



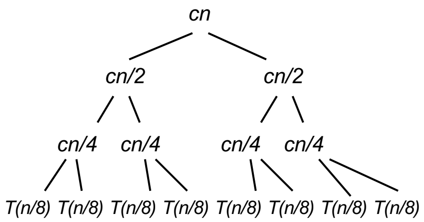
cn

cn/2 cn/2

T(n/4) T(n/4) T(n/4) T(n/4)

Merge-Sort

$$T(n)=\begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2)+cn & \text{otherwise}\end{cases}$$



cn

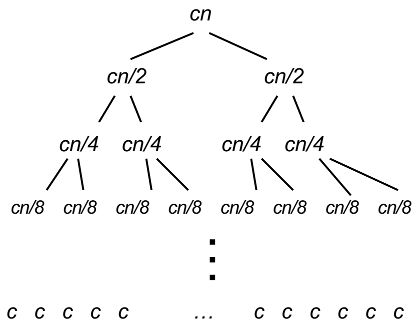
cn/2 cn/2

cn/4 cn/4 cn/4 cn/4

T(n/8) T(n/8) T(n/8) T(n/8) T(n/8) T(n/8) T(n/8) T(n/8)

Merge-Sort

$$T(n)=\begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2)+cn & \text{otherwise}\end{cases}$$



cn

cn/2 cn/2

cn/4 cn/4 cn/4 cn/4

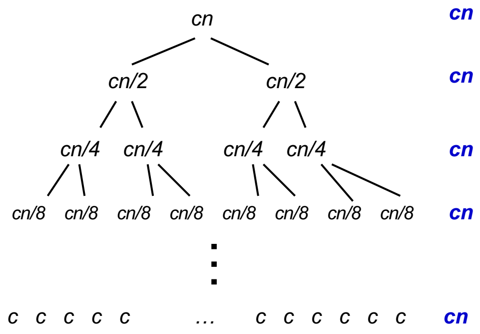
cn/8 cn/8 cn/8 cn/8 cn/8 cn/8 cn/8 cn/8

⋮

c c c c c … c c c c c c

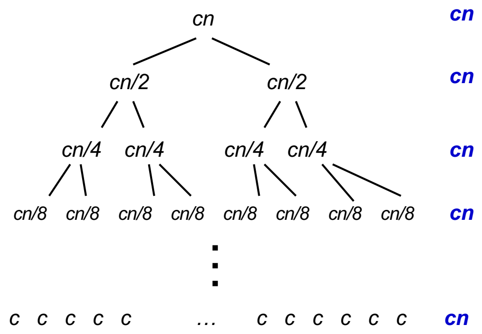
Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2)+cn & \text{otherwise} \end{cases}$$



Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2)+cn & \text{otherwise} \end{cases}$$



Depth?

Merge-Sort

We can calculate the depth, by determining when the recursion gets to down to a small problem size, e.g. 1

At each level, we divide by 2

$$\frac{n}{2^d} = 1$$

$$2^d = n$$

$$\log 2^d = \log n$$

$$d \log 2 = \log n$$

$$d = \log_2 n$$

Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2)+cn & \text{otherwise} \end{cases}$$

Running time?

- Each level costs *cn*
- log *n* levels

cn log *n* = Θ(*n* log *n*)

Recurrence

A function that is defined with respect to itself on smaller inputs

$$T(n) = 2T(n/2) + n$$

$$T(n) = 16T(n/4) + n$$

$$T(n) = 2T(n-1) + n^2$$

Why are we interested in recurrences?

Computational cost of divide and conquer algorithms

$$T(n) = aT(n/b) + D(n) + C(n)$$

- *a* subproblems of size *n/b*
- *D(n)* the cost of dividing the data
- *C(n)* the cost of recombining the subproblem solutions

In general, the runtimes of most recursive algorithms can be expressed as recurrences

The challenge

Recurrences are often easy to define because they mimic the structure of the program

But… they do not directly express the computational cost, i.e. n, n^2, …

We want to remove self-recurrence and find a more understandable form for the function

Three approaches

Substitution method: when you have a good guess of the solution, prove that it's correct

Recursion-tree method: If you don't have a good guess, the recursion tree can help. Then solve with substitution method.

Master method: Provides solutions for recurrences of the form:

$$T(n) = aT(n/b) + f(n)$$

Substitution method

Guess the form of the solution

Then prove it's correct by induction

$$T(n) = T(n/2) + d$$

Halves the input then constant amount of work

Guesses?

Substitution method

Guess the form of the solution

Then prove it's correct by induction

$$T(n) = T(n/2) + d$$

Halves the input then constant amount of work

Similar to binary search:

Guess: $O(\log_2 n)$

Proof?

$$T(n) = T(n/2) + d = O(\log_2 n)?$$

Ideas?

Proof?

$$T(n) = T(n/2) + d = O(\log_2 n)?$$

Proof by induction!
-Assume it's true for smaller T(k), i.e. k < n
-prove that it's then true for current T(n)

$$T(n) = T(n/2) + d$$

Assume $T(k) = O(\log_2 k)$ for all $k < n$
Show that $T(n) = O(\log_2 n)$

From our assumption, $T(n/2) = O(\log_2 n)$:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

From the definition of big-*O*: $T(n/2) \le c \log_2(n/2)$

How do we now prove T(n) = O(log n)?

$$T(n) = T(n/2) + d$$

To prove that $T(n) = O(\log_2 n)$ identify the appropriate constants:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

i.e. some constant c' such that $T(n) \le c' \log_2 n$

$$\begin{aligned} T(n) &= T(n/2) + d \\ &\le c\log_2(n/2) + d && \text{from our inductive hypothesis} \\ &\le c\log_2 n - c\log_2 2 + d \\ &\le c\log_2 n \boxed{-c + d} && \text{residual} \end{aligned}$$

Key question: does a constant exist such that:
$T(n) \le c' log_2 n$

$$T(n) = T(n/2) + d$$

To prove that $T(n) = O(\log_2 n)$ identify the appropriate constants:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

i.e. some constant c_2 such that $T(n) \le c_2 \log_2 n$

Key question: does a constant exist such that:
$T(n) \le c' log_2 n$

$$\mathrm{T(n)} \le c log_2 n \underbrace{- c + d}$$

if $c \ge d$, then, yes!
(if not, just let c' = d)

Base case?

For an inductive proof we need to show two things:

- Assuming it's true for $k < n$ show it's true for n
- *Show that it holds for some base case*

What is the base case in our situation?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \text{ is small} \\ T(n/2) + d & \text{otherwise} \end{cases}$$

$$T(n) = T(n-1) + n$$

Guess the solution?

At each iteration, does a linear amount of work (i.e. iterate over the data) and reduces the size by one at each step

$O(n^2)$

Assume $T(k) = O(k^2)$ for all $k < n$

- again, this implies that $T(n-1) \le c(n-1)^2$

Show that $T(n) = O(n^2)$, i.e. $T(n) \le c'n^2$

$$\begin{aligned} T(n) &= T(n-1) + n \\ &\le c(n-1)^2 + n && \text{from our inductive hypothesis} \\ &= c(n^2 - 2n + 1) + n \\ &= cn^2 \boxed{-2cn + c + n} && \text{residual} \end{aligned}$$

if $-2cn + c + n \le 0$

then let c' = c and there exists a constant such that $T(n) \le c'n^2$

$$\begin{aligned} T(n) &= T(n-1) + n \\ &\le c(n-1)^2 + n && \text{from our inductive hypothesis} \\ &= c(n^2 - 2n + 1) + n \\ &= cn^2 \boxed{-2cn + c + n} && \text{residual} \end{aligned}$$

$$-2cn + c + n \le 0$$

$$-2cn + c \le -n$$

$$c(-2n+1) \le -n$$

$$c \ge \frac{n}{2n-1}$$

which holds for any c ≥1 for n ≥1

$$c \ge \frac{1}{2 - 1/n}$$