CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

22: Priority Queues and Heapsort

David Kauchak

Alexandra Papoutsaki

Lecture 22: Priority Queues and Heapsort

- Priority Queue
- Heapsort

Some slides adopted from Algorithms 4th Edition or COS226

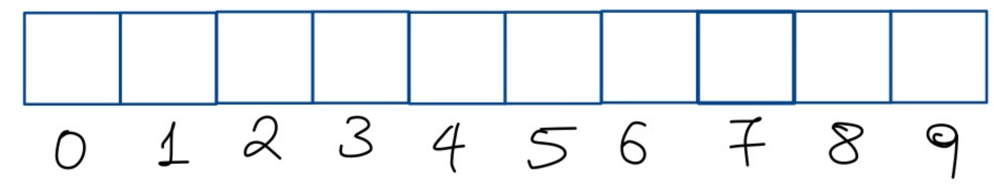
Priority Queue ADT

- Two operations:
  - Delete the maximum
  - Insert
- Applications: load balancing and interruption handling in OS, Huffman codes for compression, A* search for AI, Dijkstra’s and Prim's algorithm for graph search, etc.
- How can we implement a priority queue efficiently?

Option 1: Unordered array

- The *lazy* approach where we defer doing work (deleting the maximum) until necessary.
- Insert is $O(1)$ (will be implemented as push in stacks).
- Delete maximum is $O(n)$ (have to traverse the entire array to find the maximum element).

```java
public class UnorderedArrayMaxPQ<Key extends Comparable<Key>> {
    private Key[] pq;       // elements
    private int n;          // number of elements

    // set inititial size of heap to hold size elements
    public UnorderedArrayMaxPQ(int capacity) {
        pq = (Key[]) new Comparable[capacity];
        n = 0;
    }

    public boolean isEmpty()   { return n == 0; }
    public int size()          { return n;      }
    public void insert(Key x)  { pq[n++] = x;   }

    public Key delMax() {
        int max = 0;
        for (int i = 1; i < n; i++)
            if (less(max, i)) max = i;
        exch(max, n-1);

        return pq[--n];
    }
    private boolean less(int i, int j) {
        return pq[i].compareTo(pq[j]) < 0;
    }

    private void exch(int i, int j) {
        Key swap = pq[i];
        pq[i] = pq[j];
        pq[j] = swap;
    }
}
```

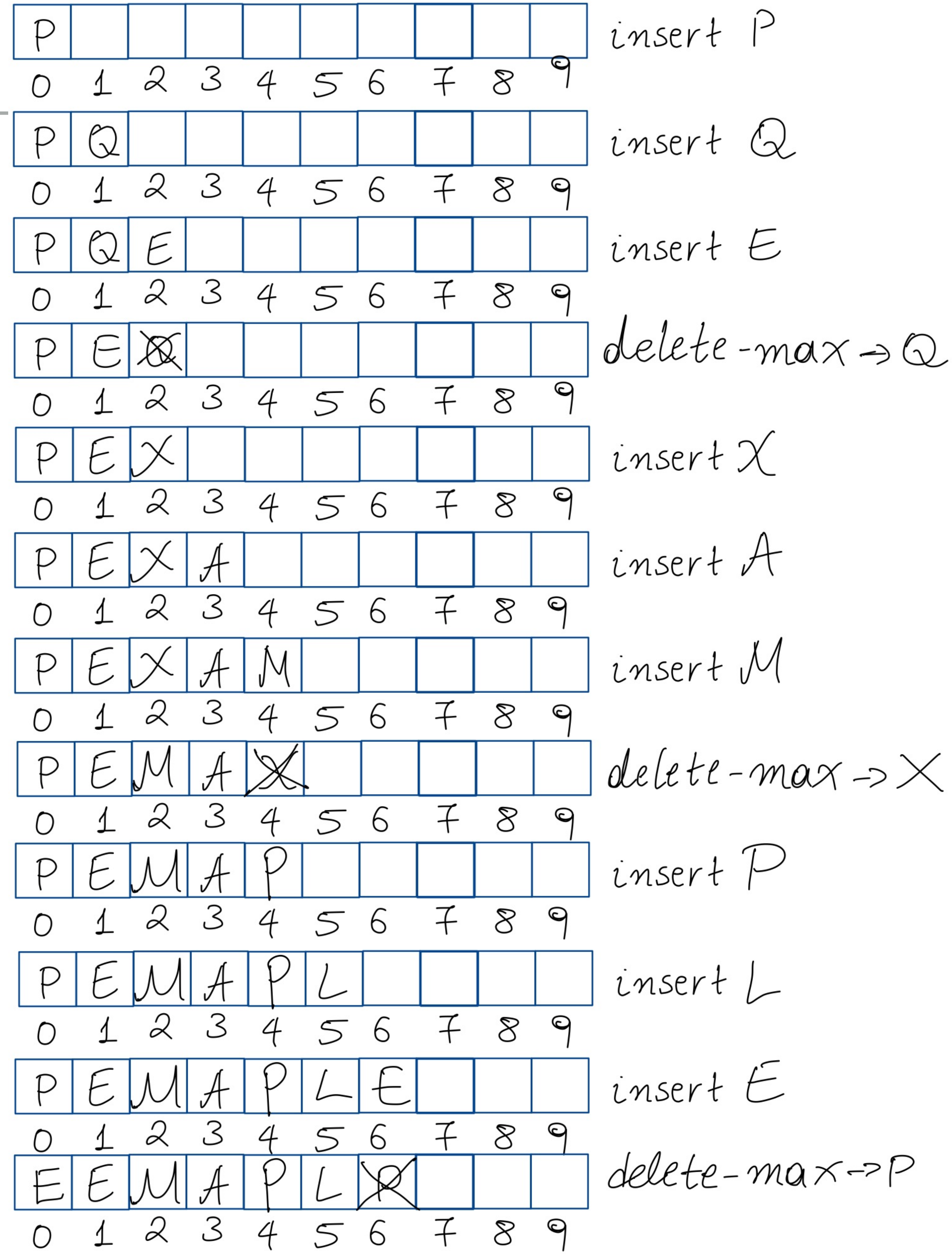


Practice Time

- Given an empty array of capacity 10, perform the following operations in a priority queue based on an unordered array (lazy approach):

1. Insert P
2. Insert Q
3. Insert E
4. Delete max
5. Insert X
6. Insert A
7. Insert M
8. Delete max
9. Insert P
10. Insert L
11. Insert E
12. Delete max

Answer


P
0 1 2 3 4 5 6 7 8 9
insert P
P Q
insert Q
P Q E
insert E
P E Q
delete-max → Q
P E X
insert X
P E X A
insert A
P E X A M
insert M
P E M A X
delete-max → X
P E M A P
insert P
P E M A P L
insert L
P E M A P L E
insert E
E E M A P L P
delete-max → P

Option 2: Ordered array

- The *eager* approach where we do the work (keeping the list sorted) up front to make later operations efficient.
- Insert is $O(n)$ (we have to find the index to insert and shift elements to perform insertion).
- Delete maximum is $O(1)$ (just take the last element which will the maximum).

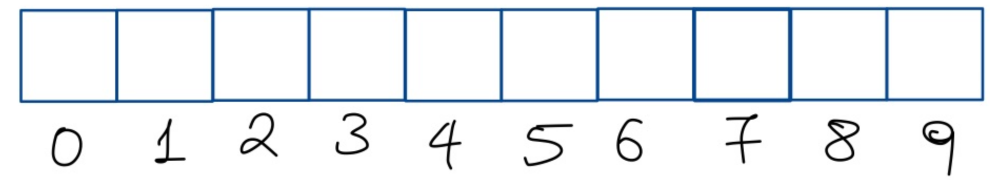
```java
public class OrderedArrayMaxPQ<Key extends Comparable<Key>> {
    private Key[] pq;          // elements
    private int n;             // number of elements

    // set inititial size of heap to hold size elements
    public OrderedArrayMaxPQ(int capacity) {
        pq = (Key[]) (new Comparable[capacity]);
        n = 0;
    }


    public boolean isEmpty() { return n == 0;  }
    public int size()        { return n;       }
    public Key delMax()      { return pq[--n]; }

    public void insert(Key key) {
        int i = n-1;
        while (i >= 0 && less(key, pq[i])) {
            pq[i+1] = pq[i];
            i--;
        }
        pq[i+1] = key;
        n++;
    }

   private boolean less(Key v, Key w) {
        return v.compareTo(w) < 0;
    }
```

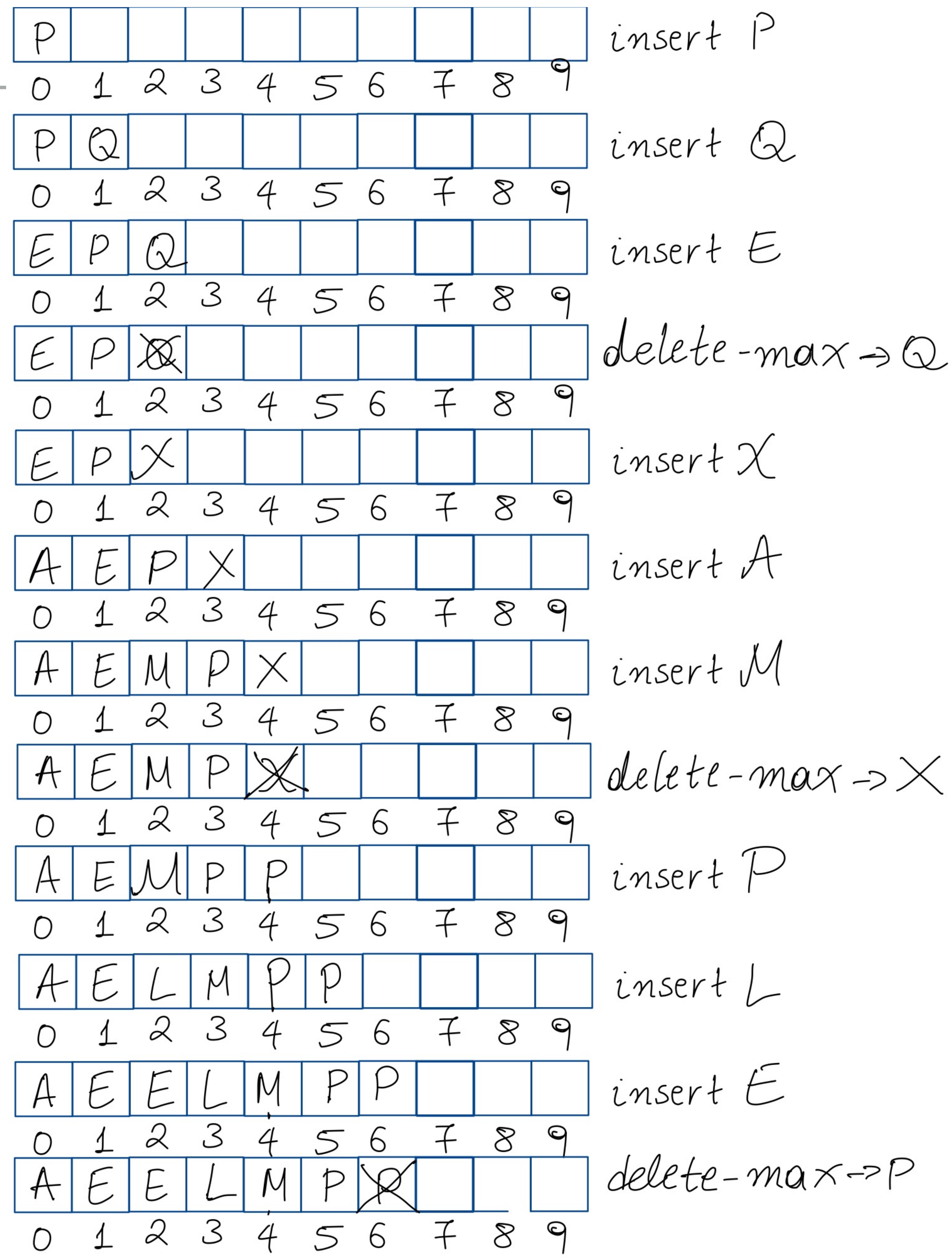


Practice Time

- Given an empty array of capacity 10, perform the following operations in a priority queue based on an ordered array (eager approach):

1. Insert P
2. Insert Q
3. Insert E
4. Delete max
5. Insert X
6. Insert A
7. Insert M
8. Delete max
9. Insert P
10. Insert L
11. Insert E
12. Delete max

Answer


P | insert P
P Q | insert Q
E P Q | insert E
E P Q | delete-max -> Q
E P X | insert X
A E P X | insert A
A E M P X | insert M
A E M P X | delete-max -> X
A E M P P | insert P
A E L M P P | insert L
A E E L M P P | insert E
A E E L M P P | delete-max -> P
0 1 2 3 4 5 6 7 8 9

Option 3: Binary heap

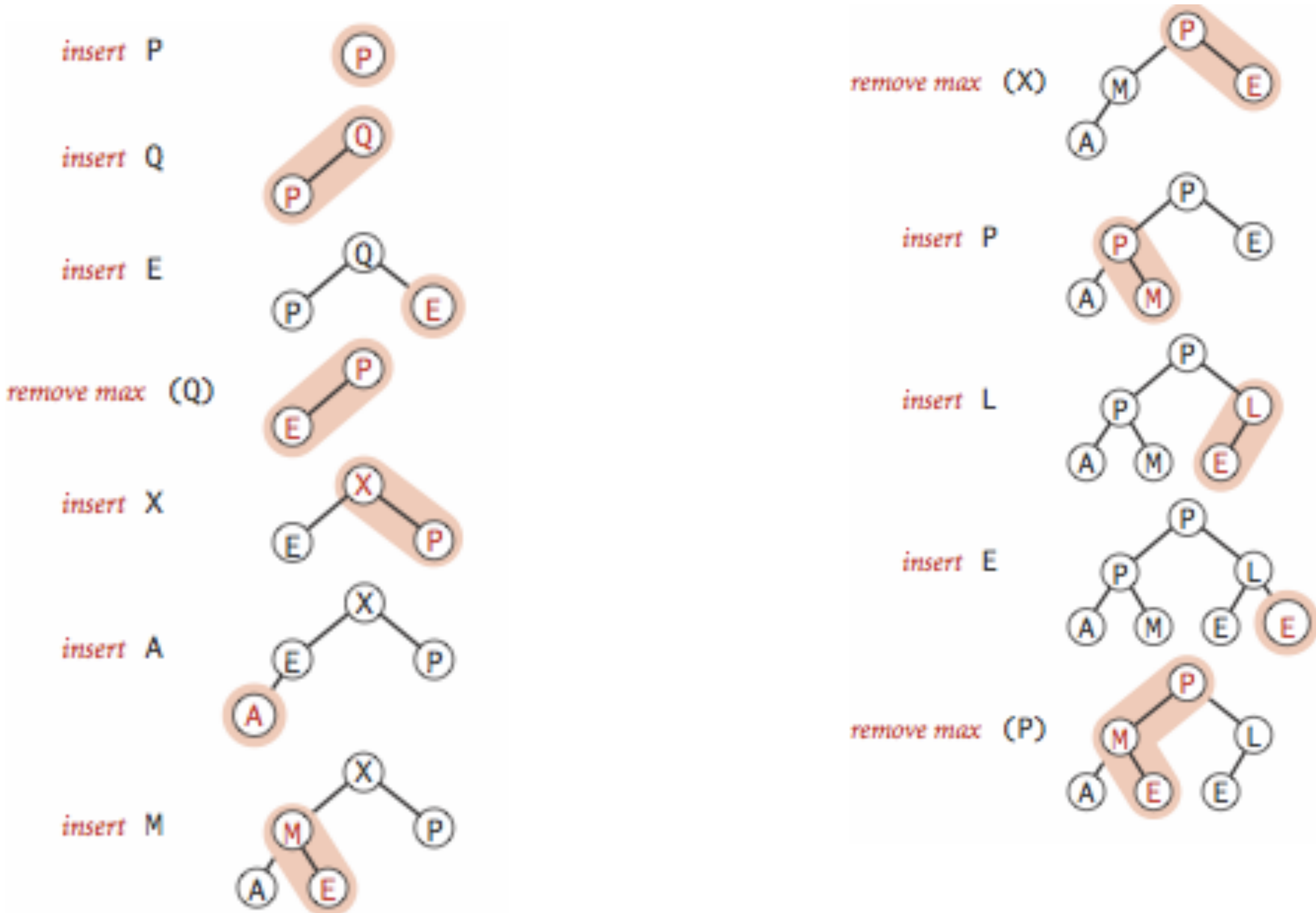
- Will allow us to both insert and delete max in $O(\log n)$ running time.
- There is no way to implement a priority queue in such a way that insert and delete max can be achieved in $O(1)$ running time.
- Priority queues are synonyms to binary heaps.

Practice Time

- Given an empty binary heap that represents a priority queue, perform the following operations:

1. Insert P
2. Insert Q
3. Insert E
4. Delete max
5. Insert X
6. Insert A
7. Insert M
8. Delete max
9. Insert P
10. Insert L
11. Insert E
12. Delete max

Answer


insert P
insert Q
insert E
remove max (Q)
insert X
insert A
insert M
remove max (X)
insert P
insert L
insert E
remove max (P)

Lecture 22: Priority Queues and Heapsort

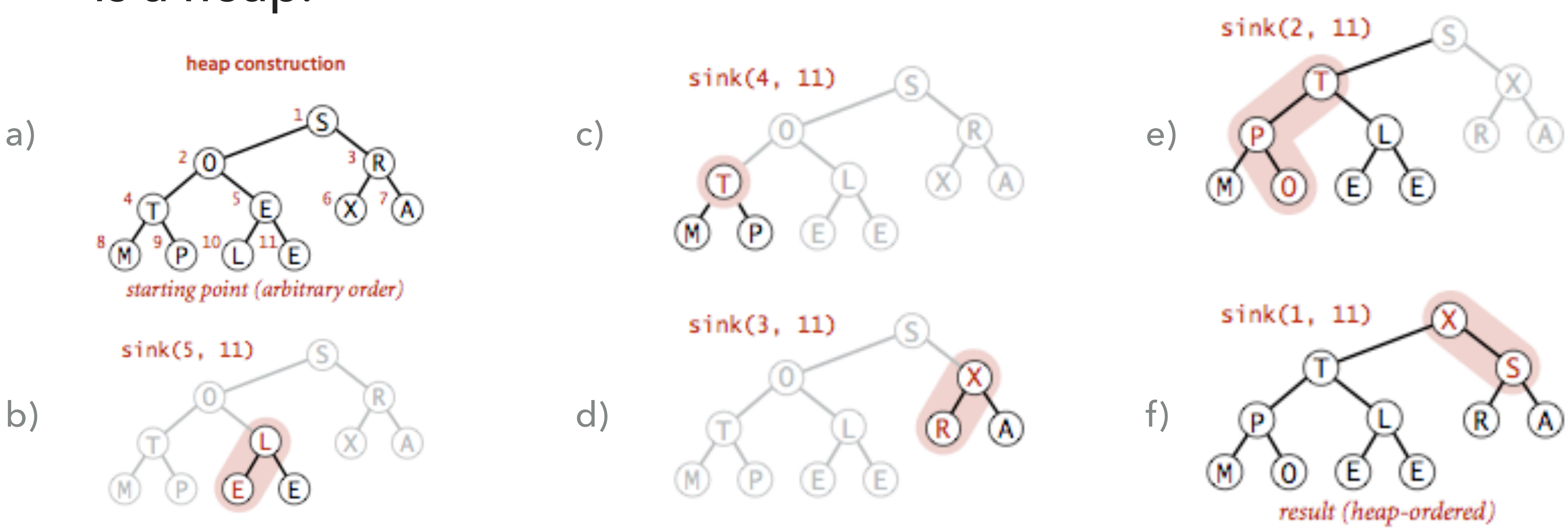
- Priority Queue
- Heapsort

Basic plan for heap sort

- Use a priority queue to develop a sorting method that works in two steps:
- 1) Heap construction: build a binary heap with all n keys that need to be sorted.
- 2) Sortdown: repeatedly remove and return the maximum key.

$O(n)$ Heap construction

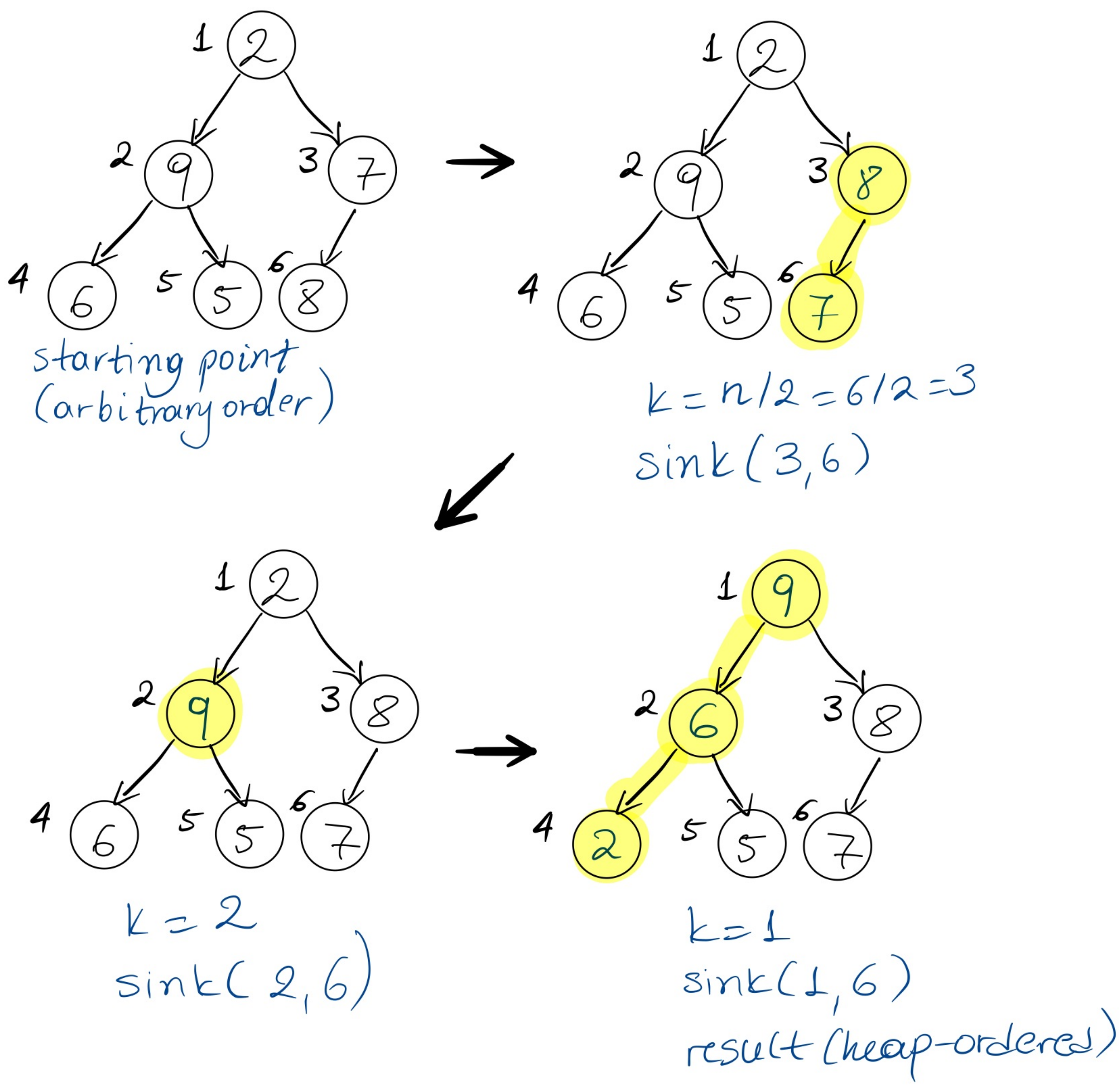
- Ignore all leaves (indices n/2+1,…,n).
- ```
  for(int k = n/2; k >= 1; k--)
          sink(a, k, n);
  ```
- Key insight: After `sink(a,k,n)` completes, the subtree rooted at k is a heap.


  ```

Practice Time

- Run the first step of heapsort, heap construction, on the array [2,9,7,6,5,8].

Answer: Heap construction

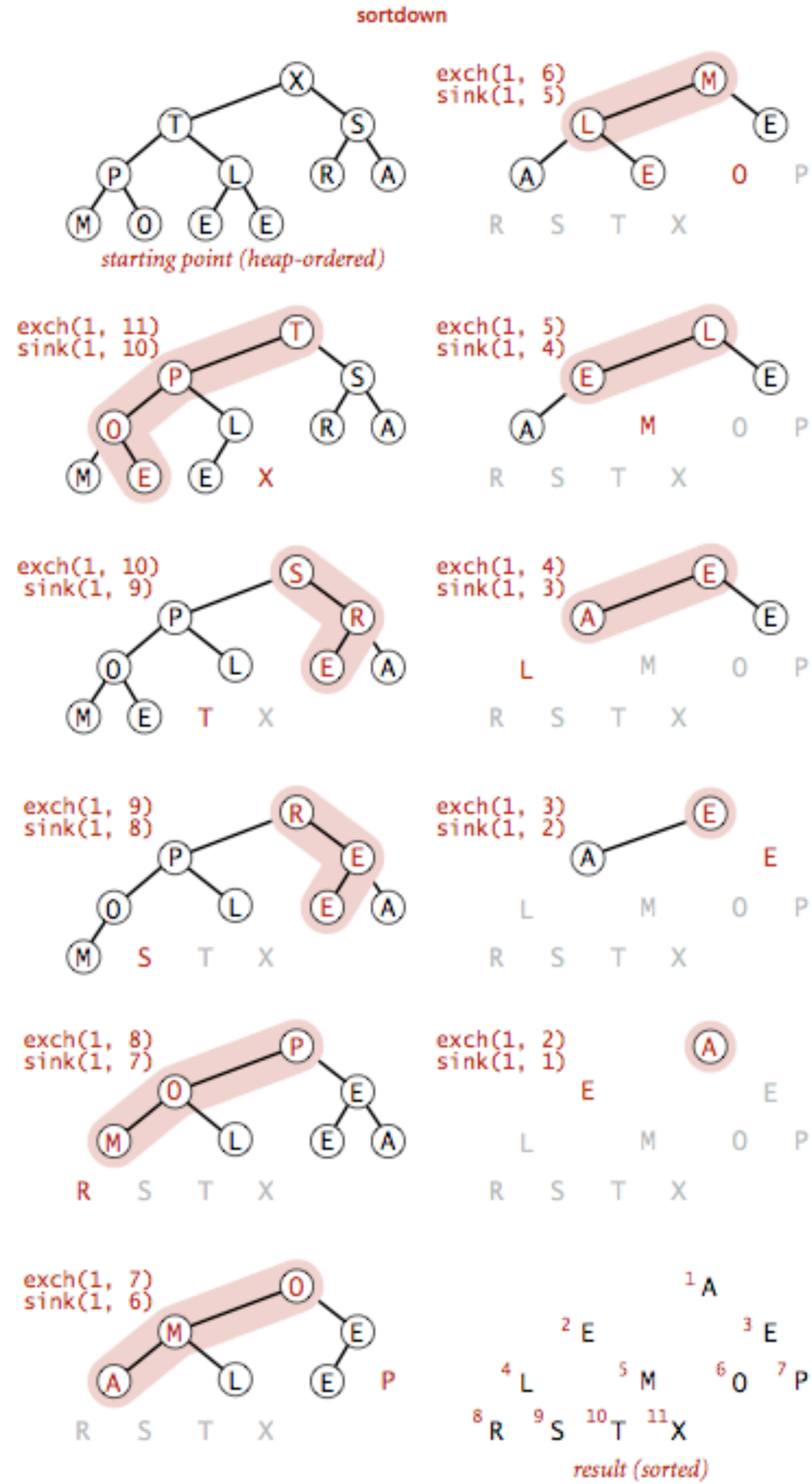


Sortdown

- Remove the maximum, one at a time, but leave in array instead of nulling out.
- ```
  while(n>1){
     exch(a, 1, n--);
     sink(a, 1, n);
  }
  ```
- Key insight: After each iteration the array consists of a heap-ordered subarray followed by a sub-array in final order.
  ```

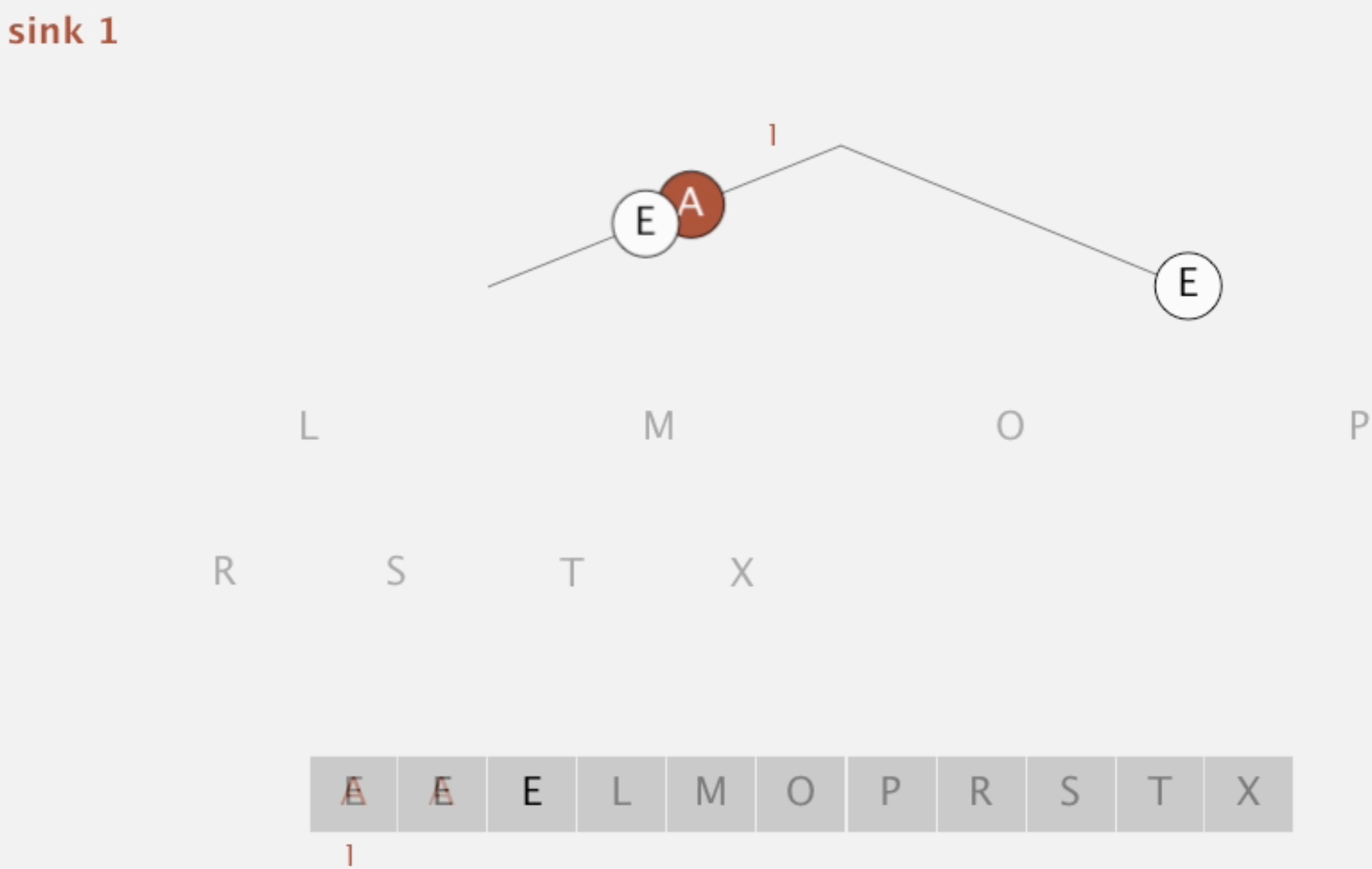
Sortdown

- ```
  while(n>1){
     exch(a, 1, n--);
     sink(a, 1, n);
  }
  ```


  ```

Heapsort demo

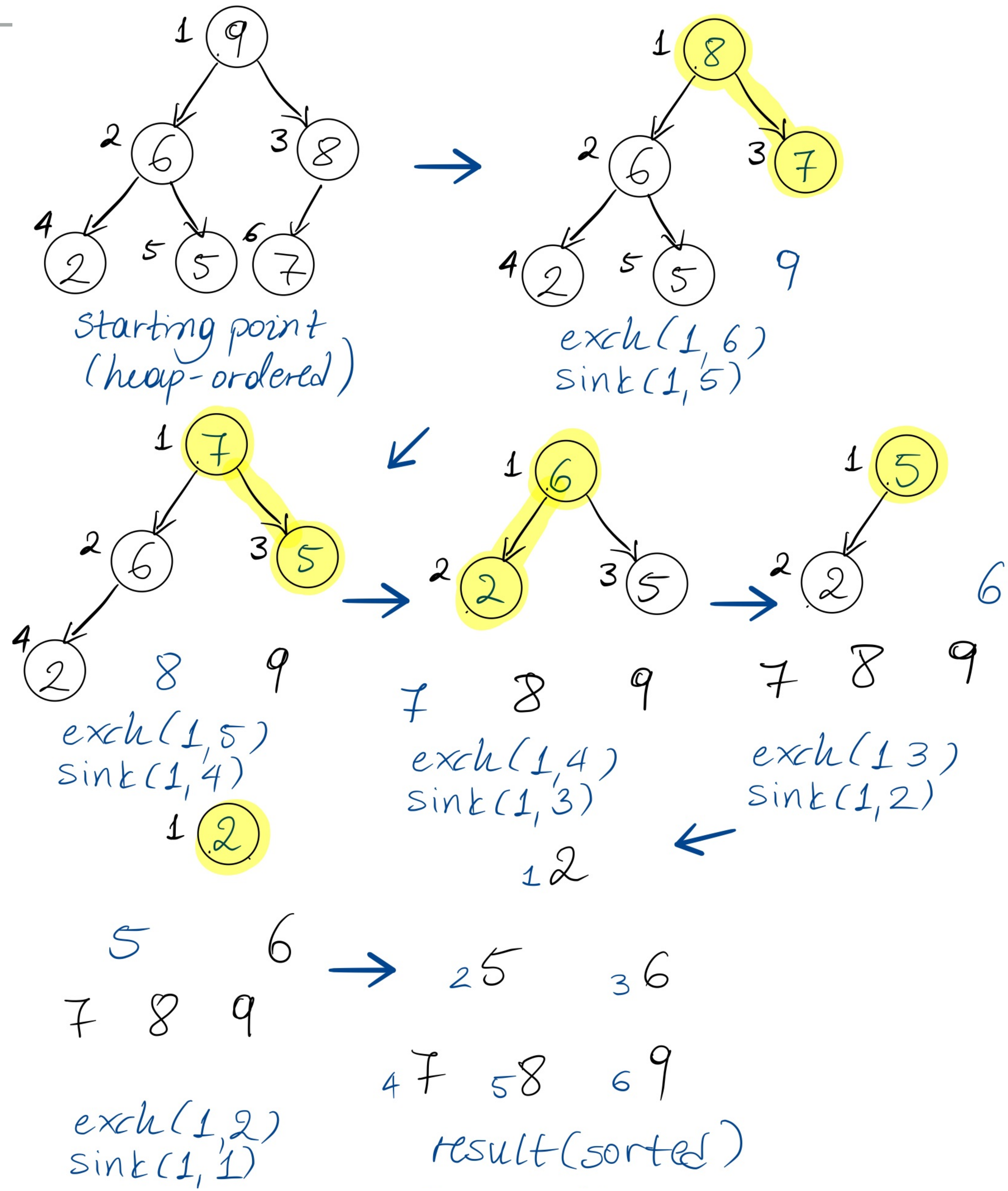
Sortdown. Repeatedly delete the largest remaining item.



Practice Time

- Given the heap you constructed before, run the second step of heapsort, sortdown, to sort the array [2,9,7,6,5,8].

Answer: Sortdown



Heapsort analysis

- Heap construction makes $O(n)$ exchanges and $O(n)$ compares.
- Sortdown and therefore the entire heap sort $O(n \log n)$ exchanges and compares.
- In-place sorting algorithm with $O(n \log n)$ worst-case!
- Remember:
  - mergesort: not in place, requires linear extra space.
  - quicksort: quadratic time in worst case.
- Heapsort is optimal both for time and space in terms of Big-O, but:
  - Inner loop longer than quick sort.
  - Poor use of cache.
  - Not stable.

Sorting: Everything you need to remember about it!

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!
Heap	X		$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Guaranteed performance; in place

Lecture 22: Priority Queues and Heapsort

- Priority Queue
- Heapsort

Readings:

- Textbook:
  - Chapter 2.4 (Pages 308-327), 2.5 (336-344)
- Website:
  - Priority Queues: https://algs4.cs.princeton.edu/24pq/
- Visualization:
  - Create (nlogn) and heapsort: https://visualgo.net/en/heap

Practice Problems:

- 2.4.1-2.4.11. Also try some creative problems.