Heaps and Priority Queue

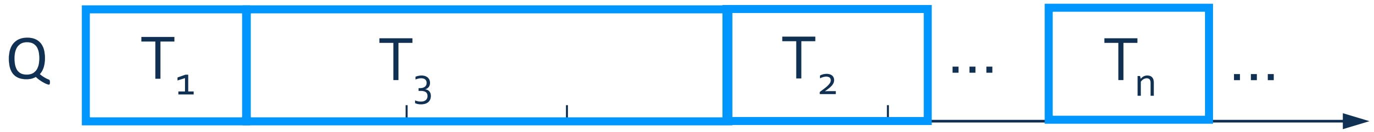
Reference: Chapter 2, Algorithms,4rd Edition, Robert Sedgewick, Kevin Wayne

Outline

- Priority Queue
- Binary Heaps
- Implementation and demo
- HeapSort

Example 1: Scheduling

- **EDF (Earliest Deadline First) Scheduling**
  - Tasks wait in the queue
  - A task with a shorter deadline has a higher priority
  - Executes a job with the earliest deadline



Example 1: Cont.

- Task T1 is dispatched and removed from the Task waiting queue.

- Before T1 is completed, Task Tn+1 arrives. It has the earliest deadline. Tn+1 will be dispatched next.

Priority Queue

- EDF scheduler processes Tasks in order. But not necessarily in full sorted order and not necessarily all at once.

- An appropriate data type for Task Waiting Queue supports two operations: ***remove*** *the maximum priority task* and ***insert*** *new tasks*. Such a data type is called a *priority queue*.

- Priority queues are characterized by the ***remove*** *the maximum* and ***insert*** operations.

Priority Queue Interface

```
public interface PriorityQueue <T extends Comparable<T> >
{
        void insert(T t);
        void remove() throws EmptyQueueException;
        T top() throws EmptyQueueException;
        boolean empty();
}
```

Example 2: Statistics

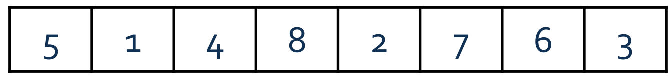
- Find the largest M items in a stream of N items (N huge, M large)
  - N is huge, cannot sort in memory
  - M is large, insert, remove must be fast.

Order of growth of finding the largest M in a stream of N items

Implementation	Time	Space
Sort	N log N	M
Array	N M	M

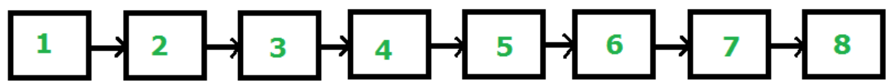
Elementary Implementations

- Unordered Array:



- Ordered Array:

- Linked List:



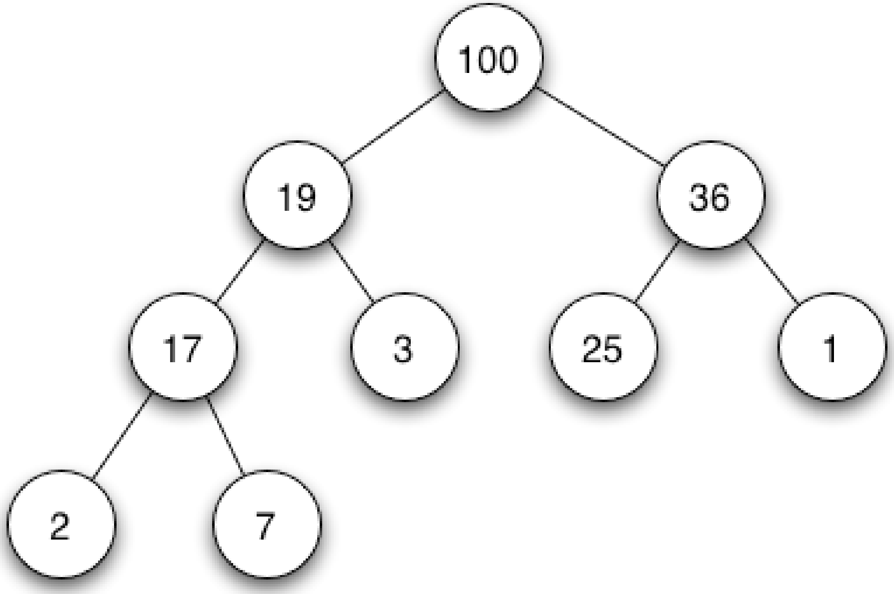
- Binary Tree

Order-of-growth of running time for priority queue with N items

Implementation	Insert	Remove Max	Max
Unordered Array	1	N	N
Ordered Array	N	1	1
Linked List (unsorted)	1	N	N
Goal	**Log N**	**Log N**	**1**

Binary Heap

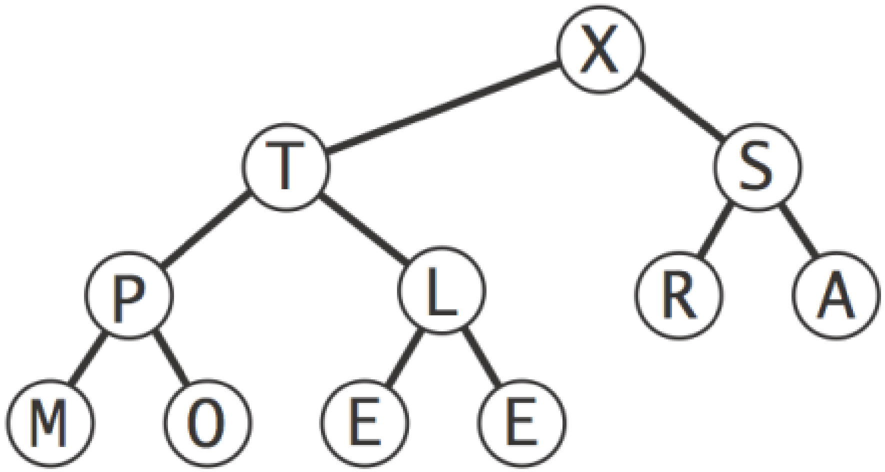
- Complete Binary Tree
- Each node is larger than (or equal to) its two children (if any).



Complete Binary Tree in Nature

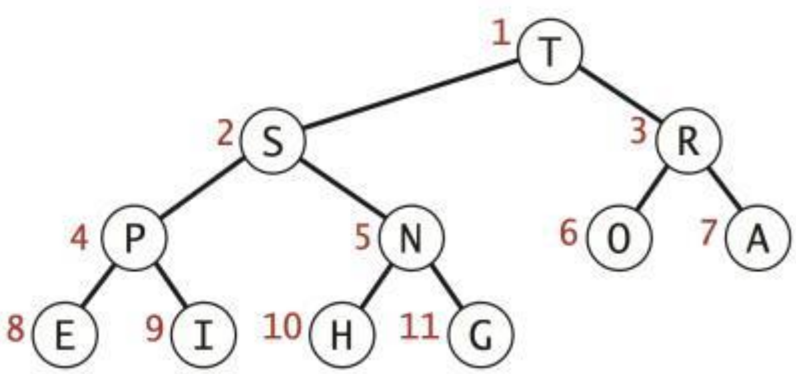
Binary Heap Properties

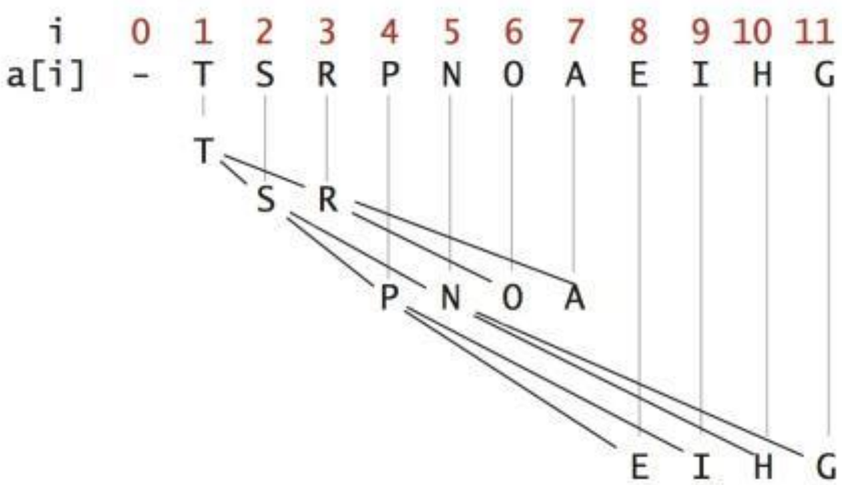
- The largest is found at the root.
- Height of complete tree with N nodes is ⌊lg N⌋
- Height only increases when N is a power of 2



Binary Heap Representations

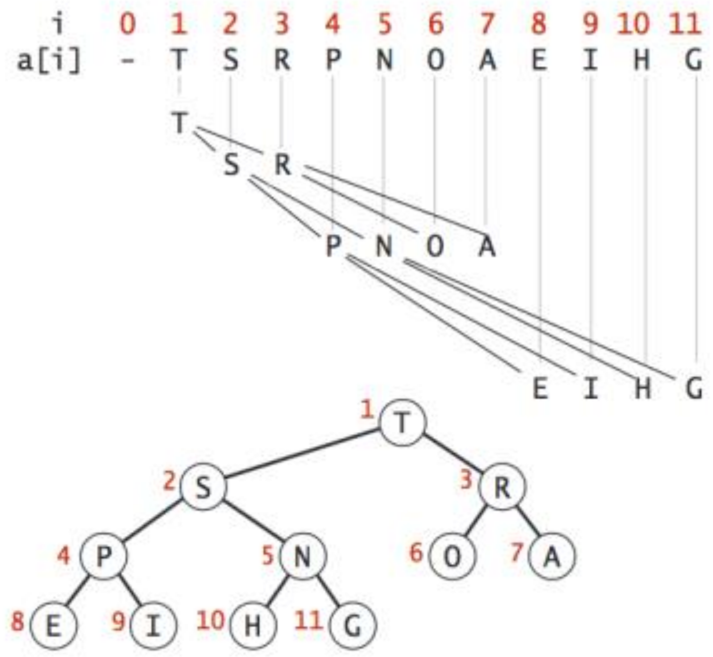
- Array representation of a complete binary tree
  - Take nodes in level order
  - No explicit links needed





Binary Heap Representations

- Largest key is **a[1]**, which is root of binary tree.
- Can use array indices to move through tree.
- Parent of node at **k** is at **k/2**.
- two children of the node at k are in positions 2k and 2k + 1.



Heap representations

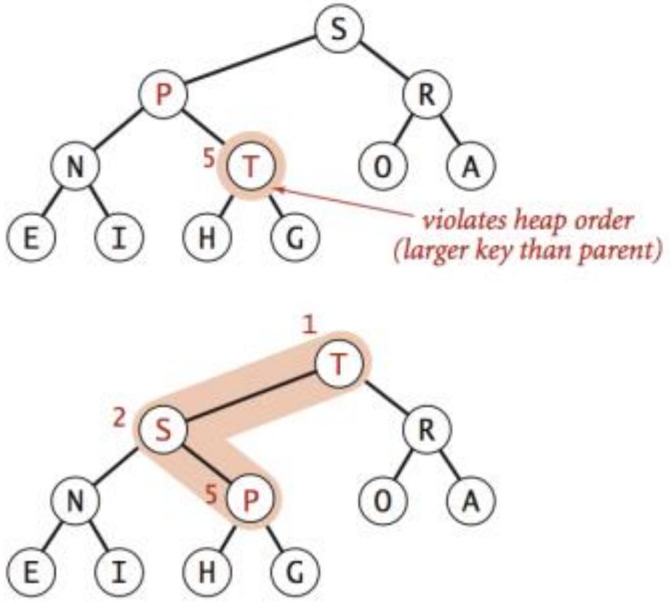
Algorithms on Heaps

Promotion: Child's key becomes larger key than its parent's key.

To eliminate the violation:

- Exchange key in child with key in parent.
- Repeat until heap order restored.

```
private void swim(int k) {
  while (k > 1 && less(k/2, k)){
      swap(k, k/2);
      k = k/2;
  }
}
```

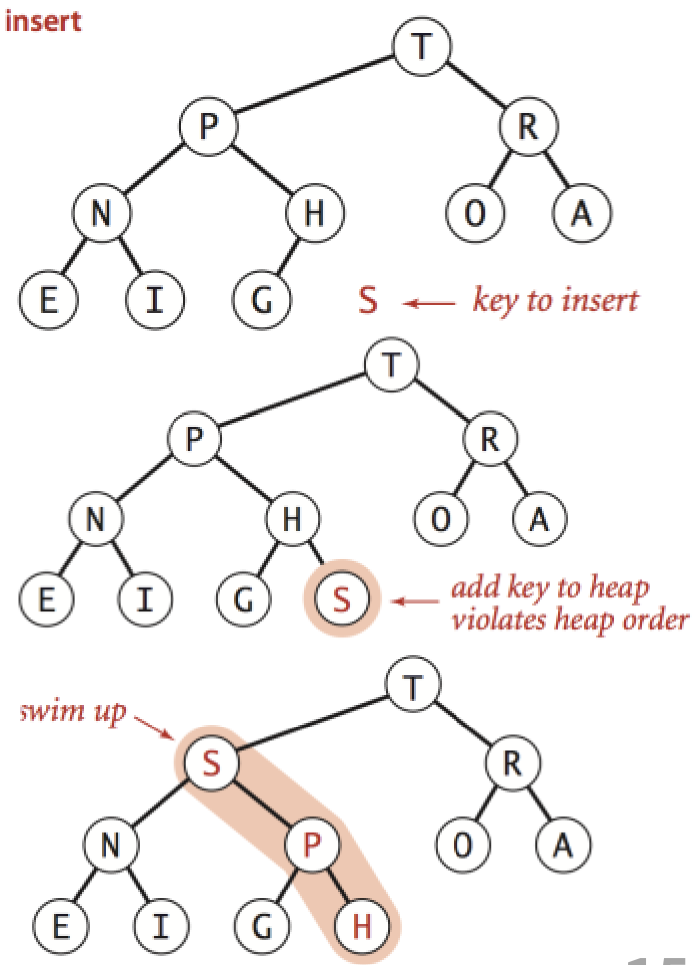


Algorithms on Heaps

Insertion in a heap:

- Insert. Add node at end, then swim it up.
- Cost. At most lg *N* compares.

```
public void insert(T t){
    pqArray.add(t);
    Size++;
    swim(Size);
}
```



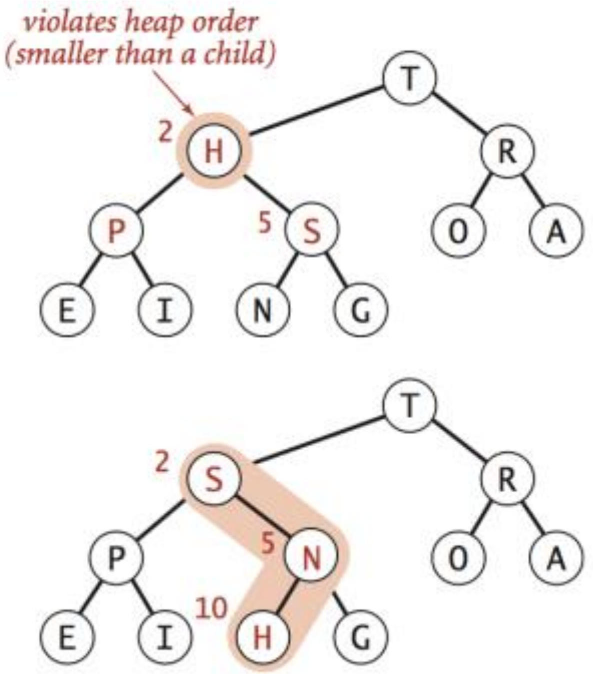
Algorithms on Heaps

Demotion: Parent's key becomes smaller than one (or both) of its children's keys.

To eliminate the violation:

- Exchange key in parent with key in larger child.
- Repeat until heap order restored.

```
private void sink(int k){
  while(2 * k <= Size){
    int j = 2*k;
    if(j< Size && less(j,j+1)) j++;
    if(!less(k,j)) break;
     swap(k,j);
     k = j;
  }
}
```



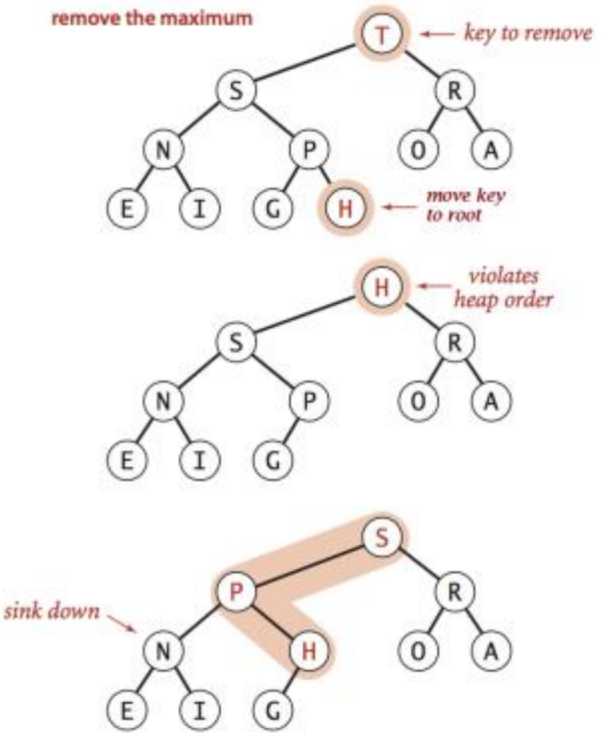
Top-down reheapify (sink)

Algorithms on Heaps

Remove the maximum in a heap:

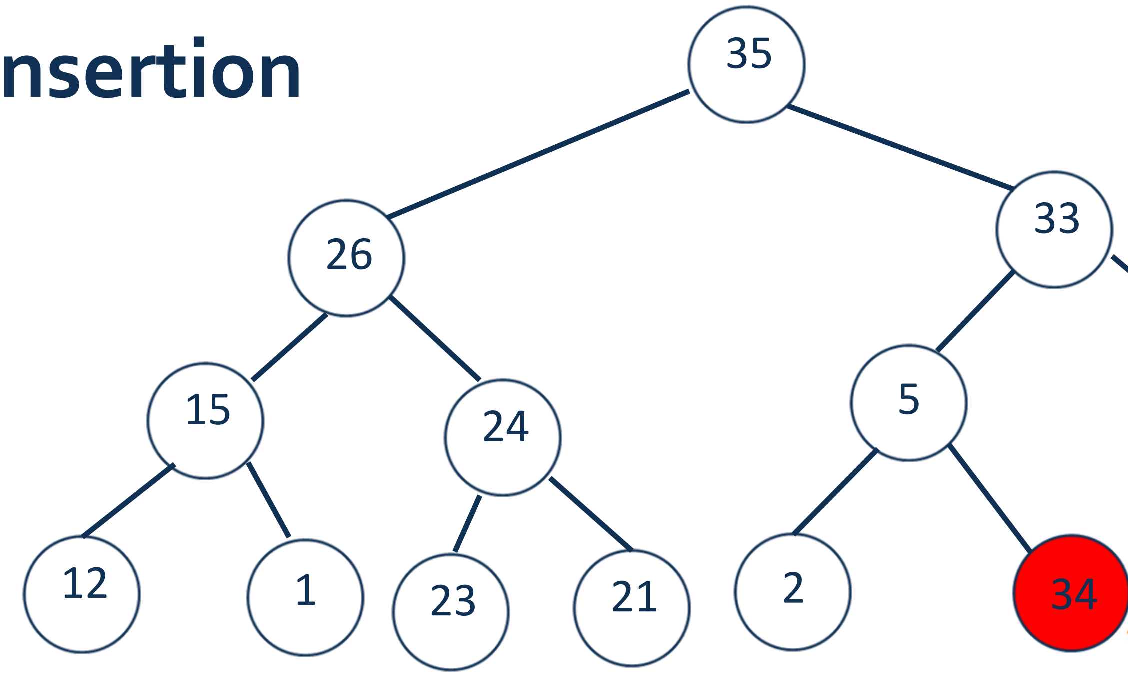
- Delete max: Replace root with node at end, then sink it down.
- Cost: At most 2 lg *N* compares.

```
public void remove(){
  if(Size == 0){
    throw new EmptyQueueException("Queue is empty.");
  }
  pqArray.set(1,pqArray.get(Size));
  pqArray.remove(Size);
  Size--;
  sink(1);
}
```



Binary Heap Demo

Insertion



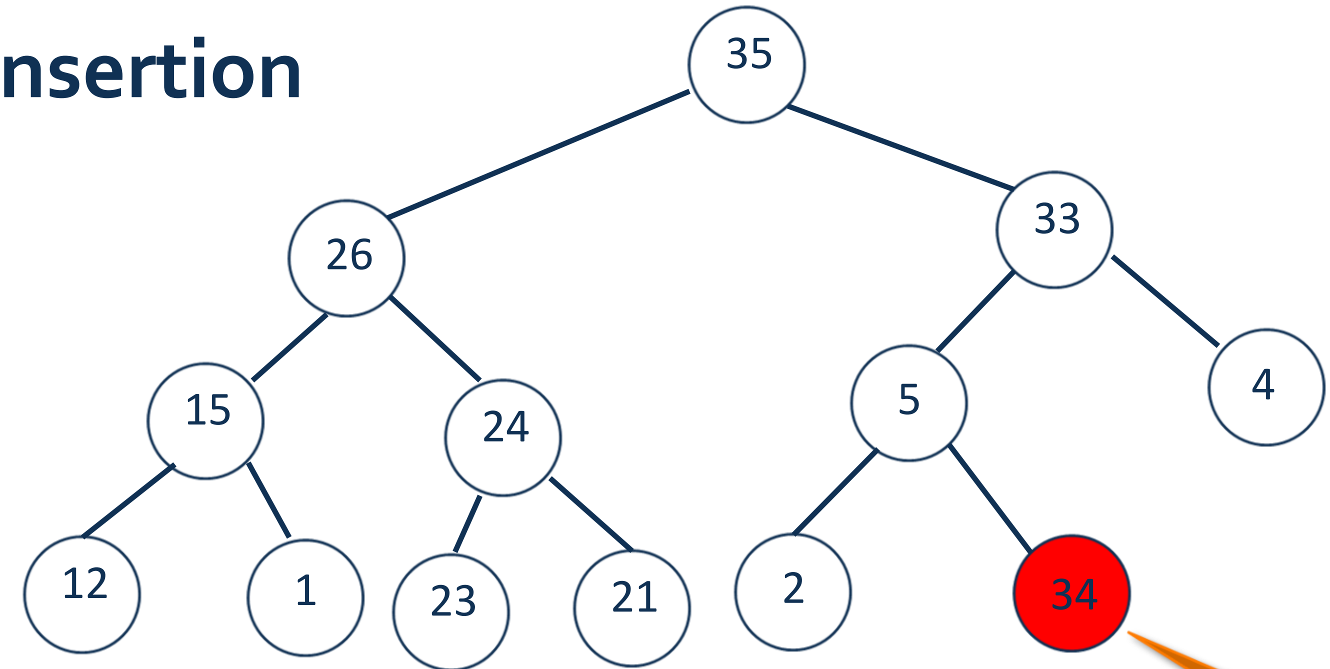
Insert 34

Violation. swim

--	35	26	33	15	24	5	4	12	1	23	21	2	34

Binary Heap Demo

Insertion

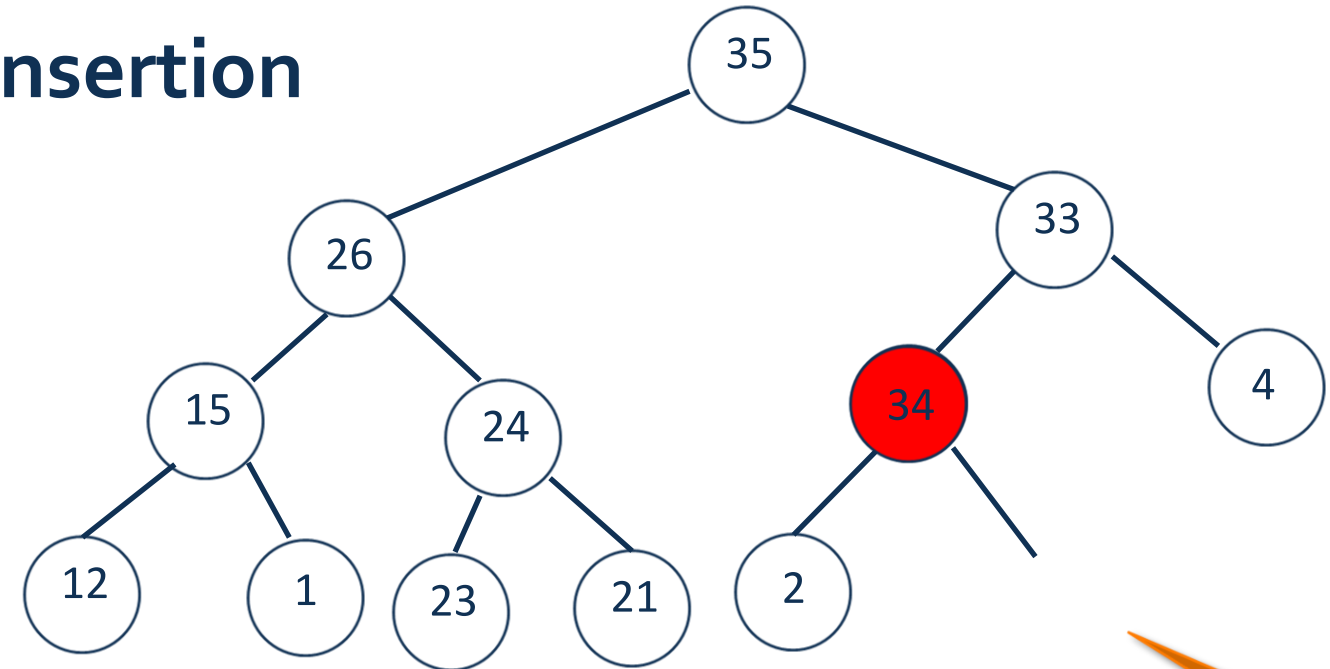


Violation.
swim

--	35	2 6	33	15	24	5	4	12	1	23	21	2	34

Binary Heap Demo

Insertion

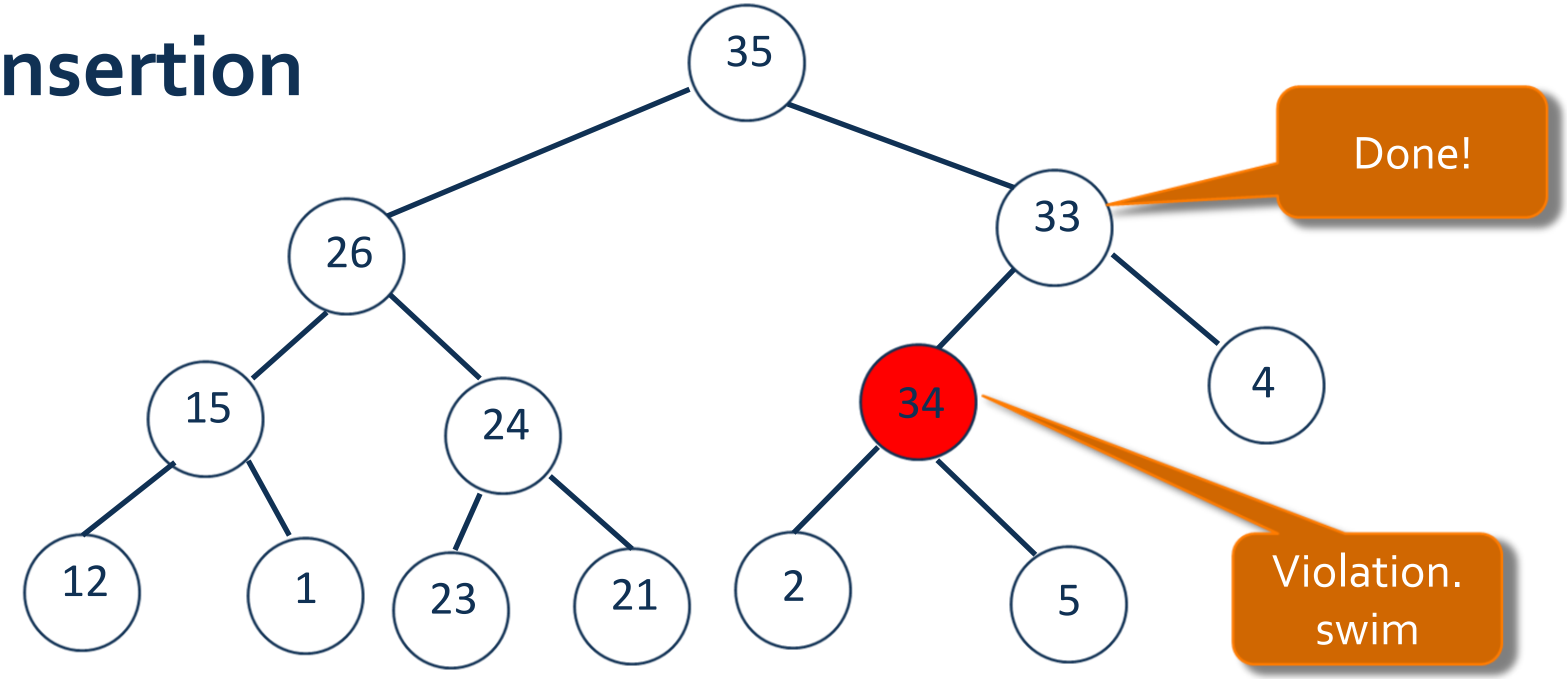


--	35	26	33	15	24	34	4	12	1	23	21	2

Violation.
swim

Binary Heap Demo

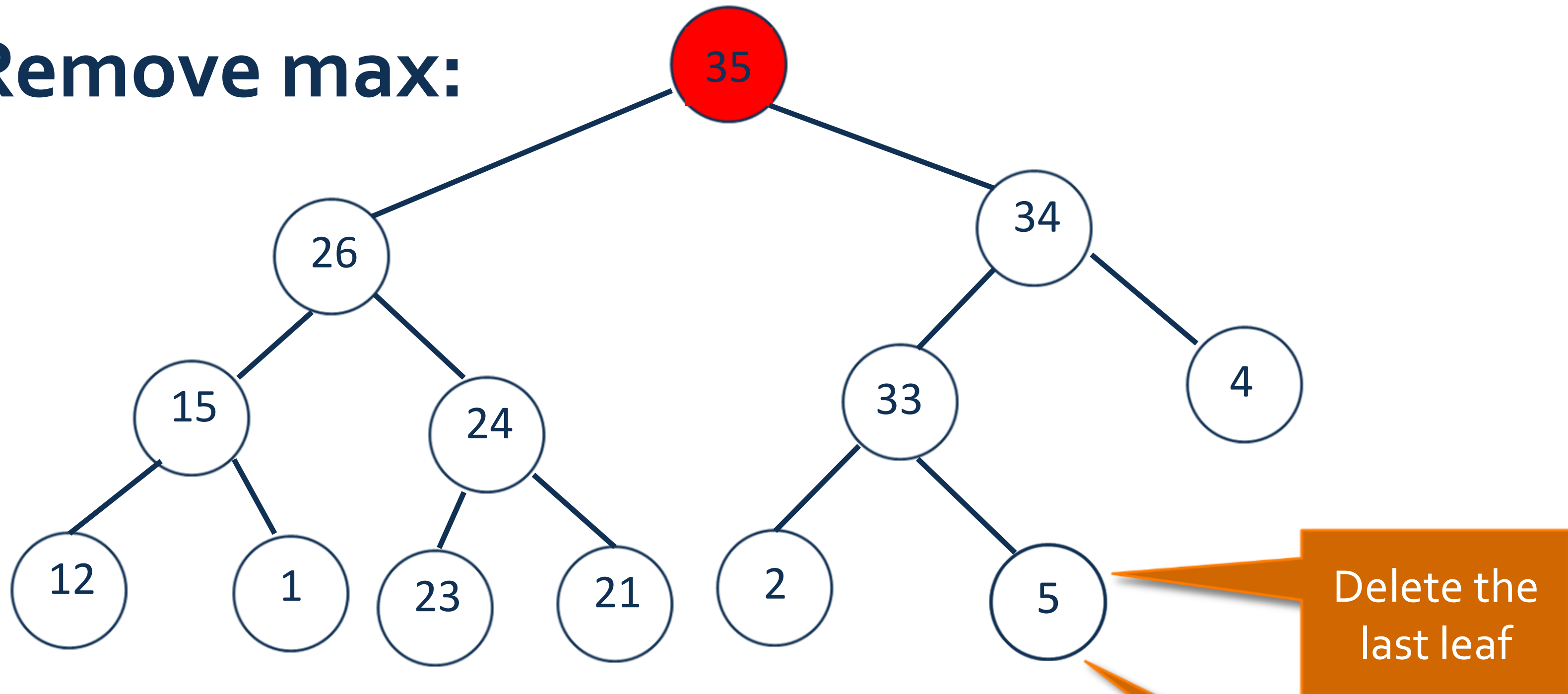
Insertion



--	35	2 6	33	15	24	34	4	12	1	23	21	2	5

Binary Heap Demo

Remove max:

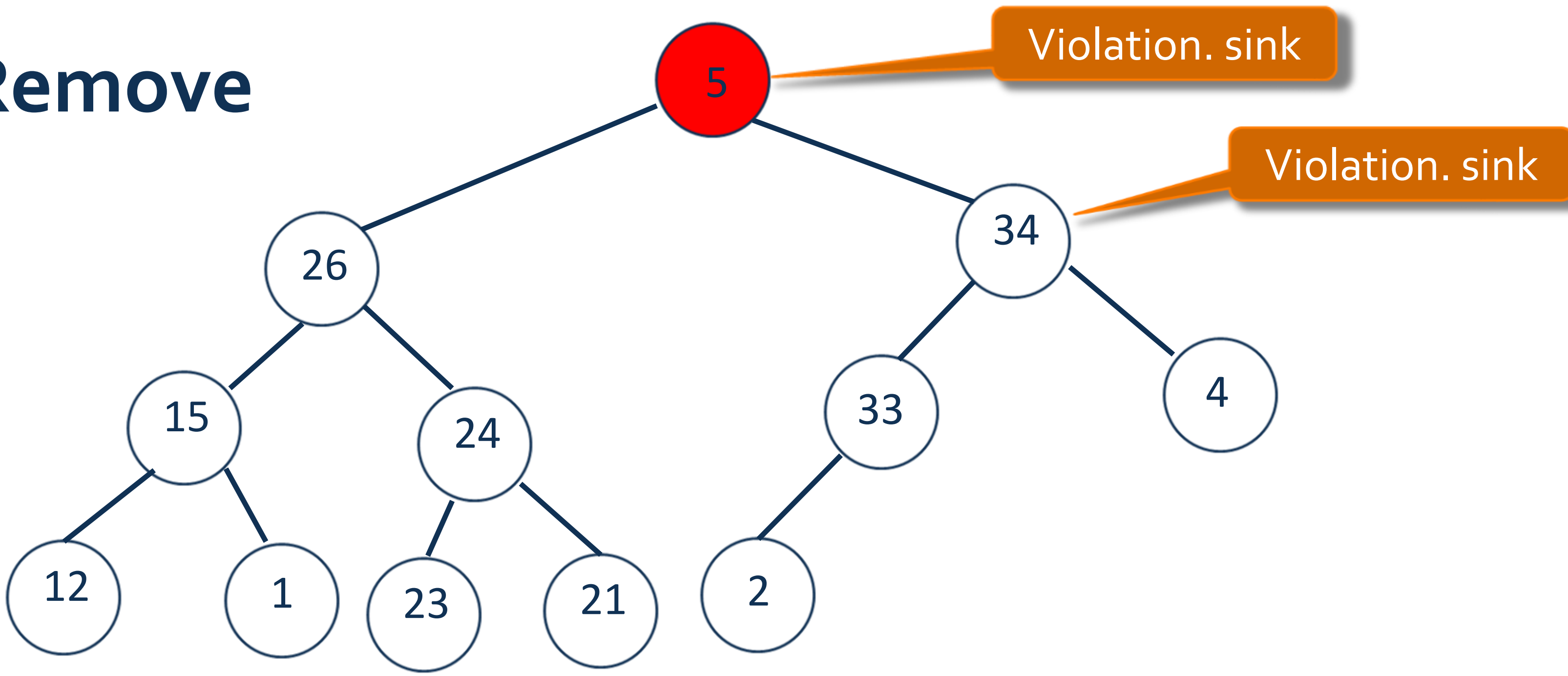


--	35	2 6	34	15	24	33	4	12	1	23	21	2	5

Move the last leaf to root

Binary Heap Demo

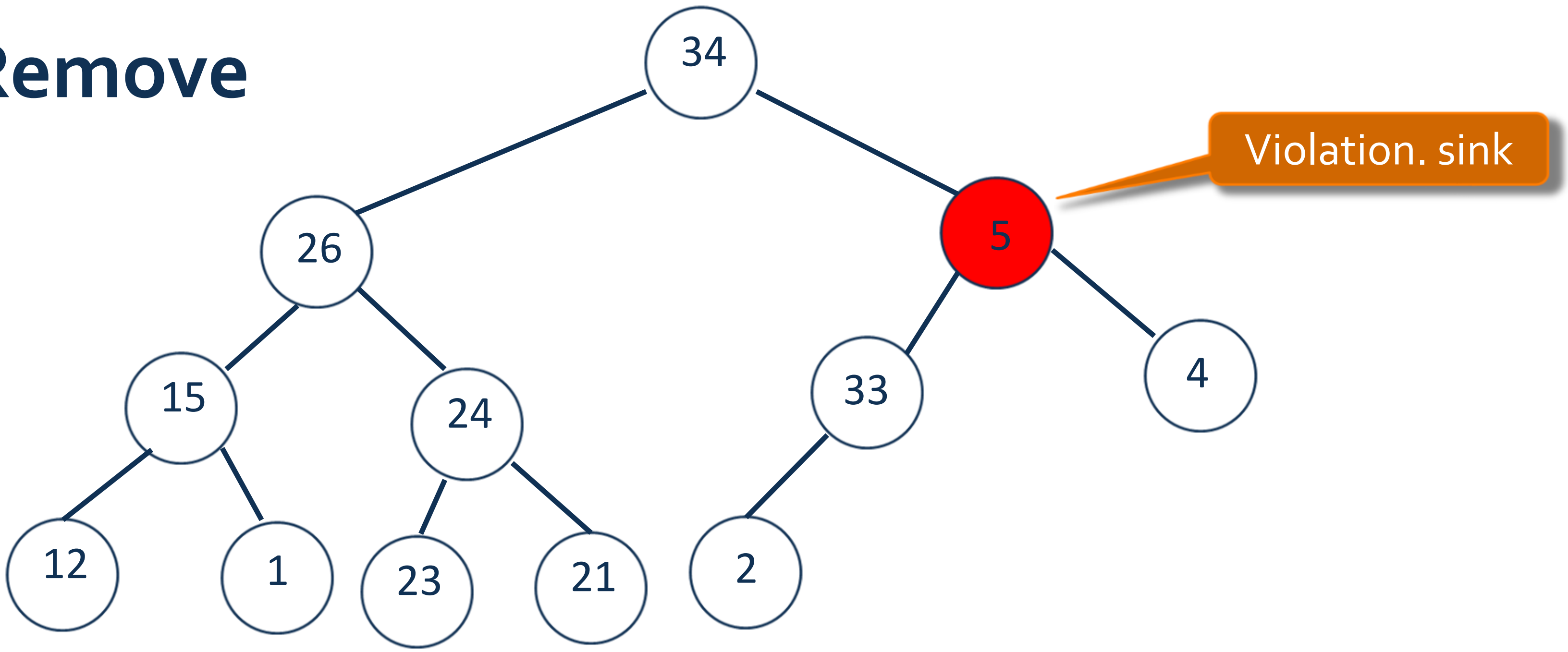
Remove



--	5	26	34	15	24	33	4	12	1	23	21	2

Binary Heap Demo

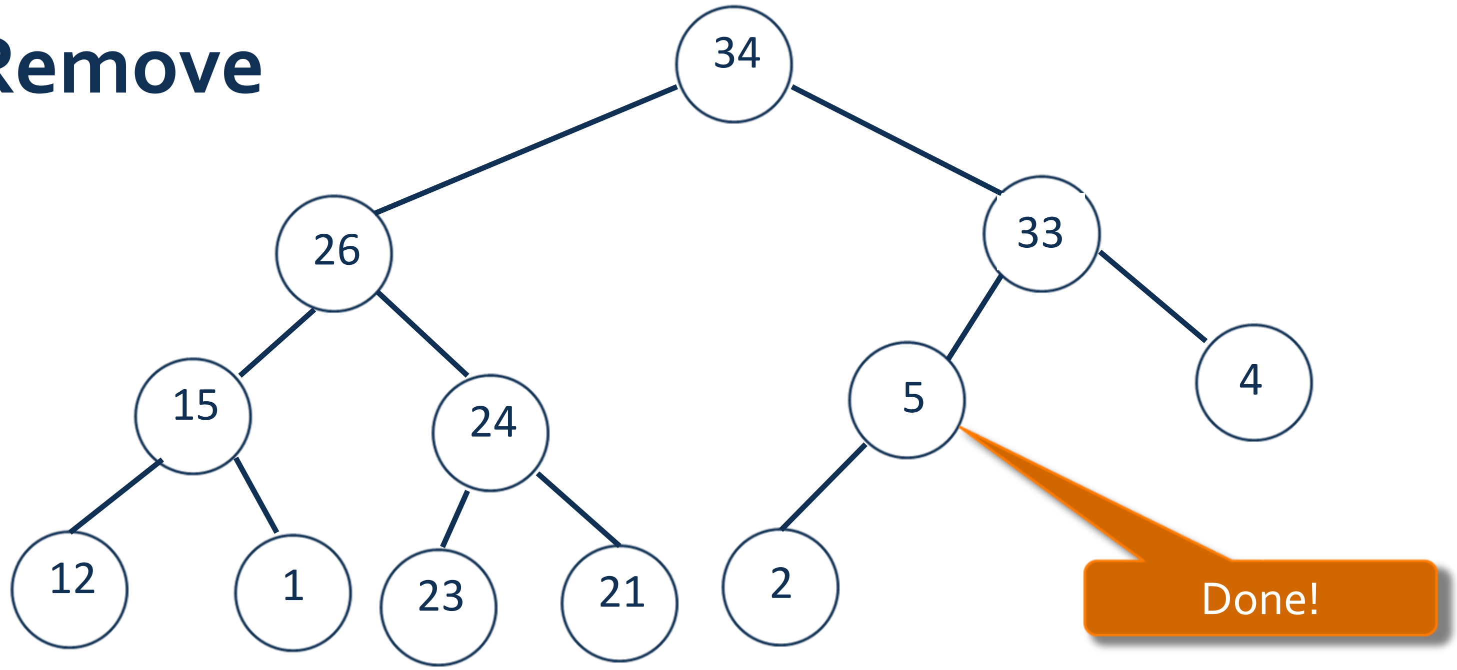
Remove



--	34	26	5	15	24	33	4	12	1	23	21	2

Binary Heap Demo

Remove



--	34	26	33	15	24	5	4	12	1	23	21	2

Binary Heap Java Code Demo

File name	Description
PriorityQueue.java	Interface
MaxPQ.java	PQ implementation
GraphVizWrite.java	Visualize the heap
EmptyQueueException.java	Exception
MaxPQTest.java	main method
InputHelper.java	input utility

Cost summary

Implementation	Insert	Remove Max	Max
Unordered Array	1	N	N
Ordered Array	N	1	1
Linked List (unsorted)	1	N	N
Binary Heap	**Log N**	**Log N**	**1**

Immutability of keys

- Assumption: client does not change keys while they're on the PQ.
- Best practice: use immutable keys.

Immutability: implementing in Java

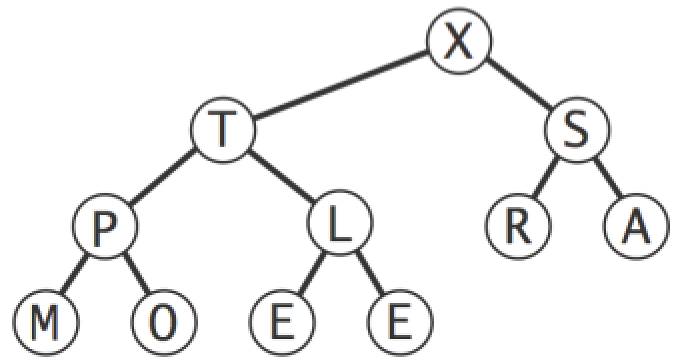
- Immutable data type. Can't change the data type value once created.
- Immutable. **String**, **Integer**, **Double**, **Color**, **Vector**, **Transaction**, **Point2D**.
- Mutable. **StringBuilder**, **Stack**, **Counter**, Java array.

Heap Sort

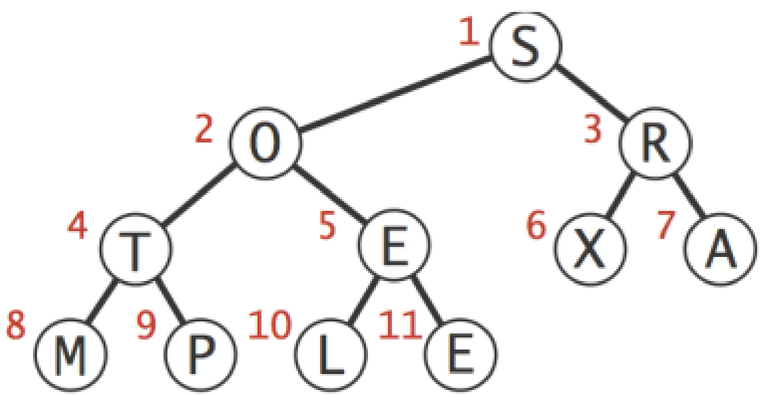
- Sort an array using heap representations
- worst case running time $O(nlgn)$
- an **in-place** sorting algorithm: only a constant number of array elements are stored outside the input array at any time. thus, require at most $O(1)$ additional memory

Heap Sort

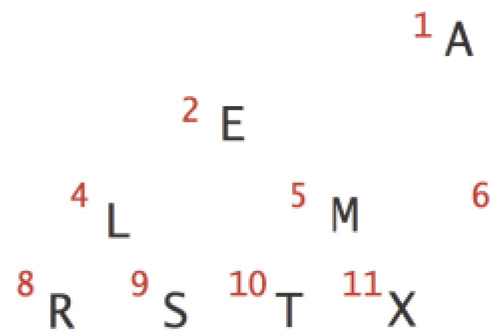
- **Idea:**
  1. Create max-heap with all N keys.
  2. Repeatedly remove the maximum key.



Build a Max Heap



Original Array

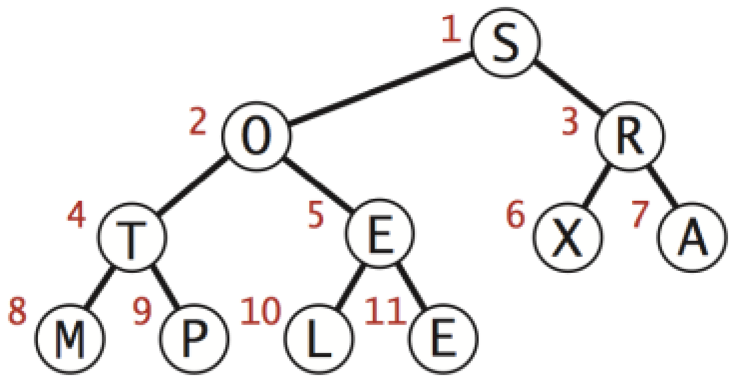


Sorted Array

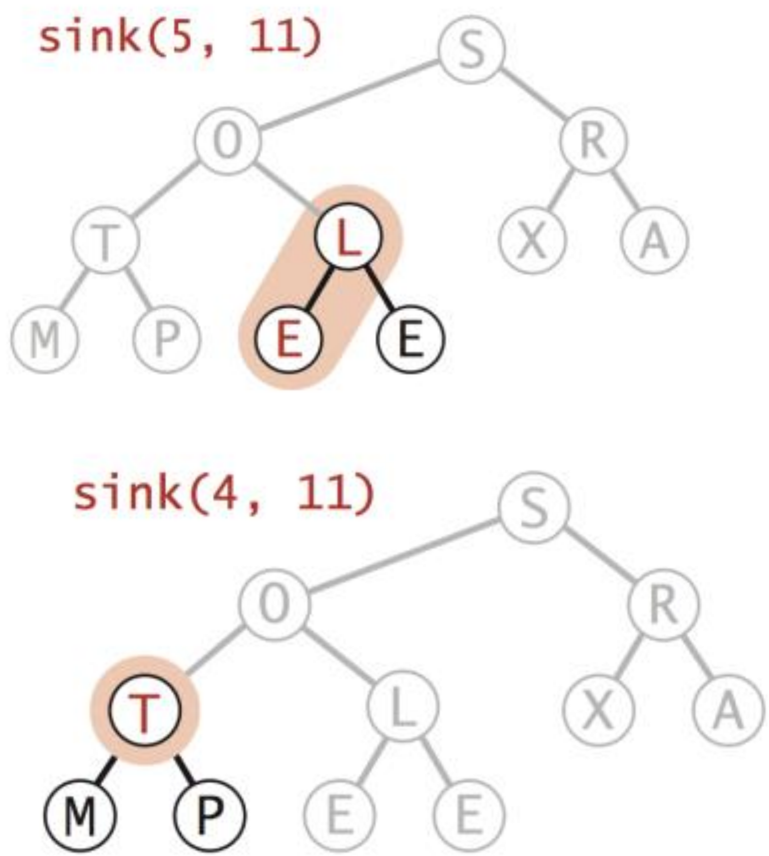
Step 1: Build max-heap

Build heap using bottom-up method

```
for (int k = N/2; k >= 1; k--)
  sink(k, N);
```



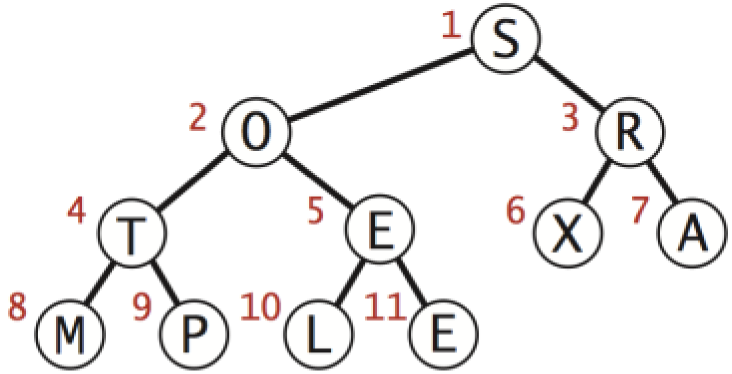
Arbitrary Array



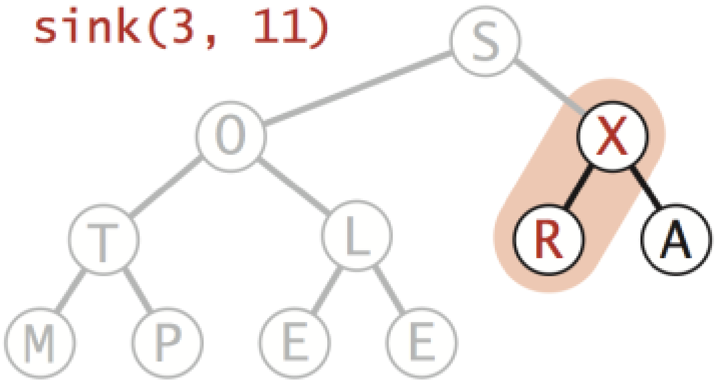
Step 1: Build max-heap

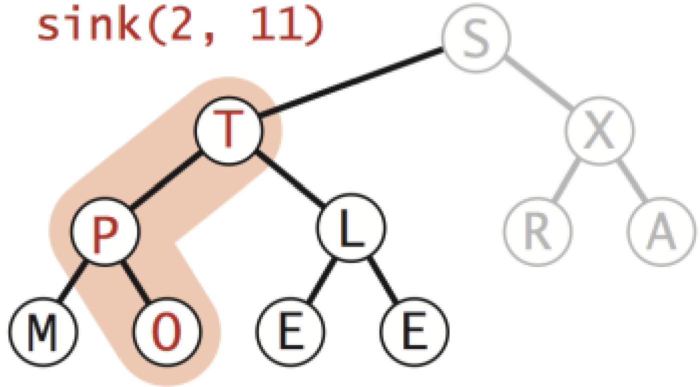
Build heap using bottom-up method

```
for (int k = N/2; k >= 1; k--)
  sink(k, N);
```



Arbitrary Array

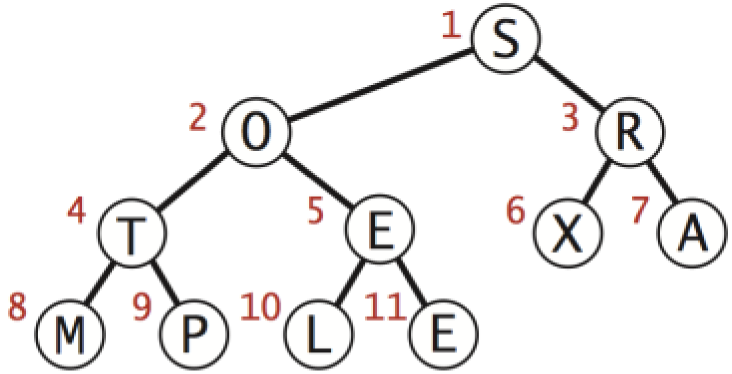




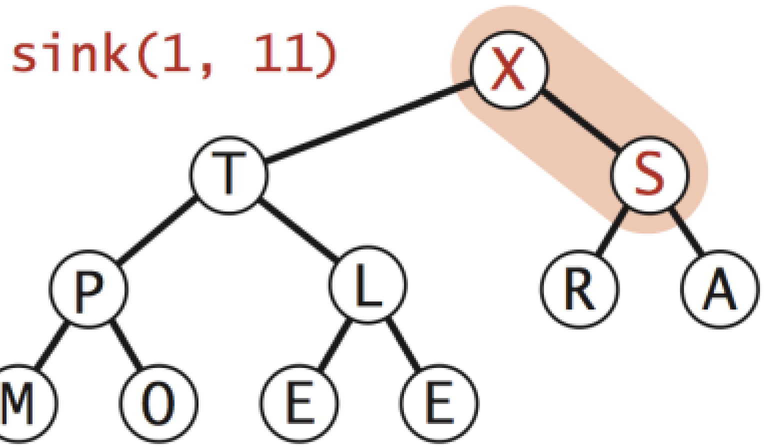
Step 1: Build max-heap

Build heap using bottom-up method

```
for (int k = N/2; k >= 1; k--)
  sink(k, N);
```



Arbitrary Array



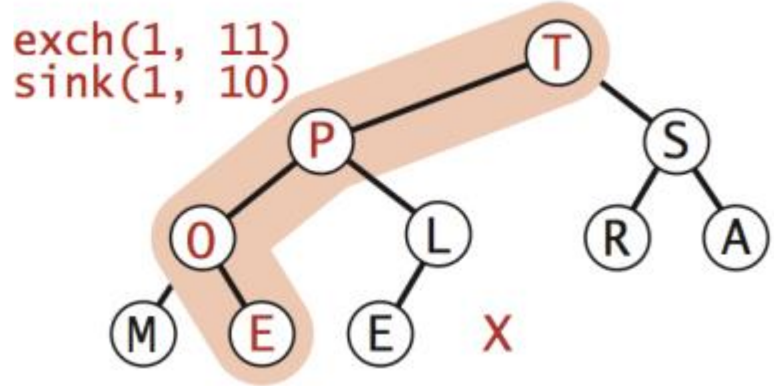
Max-heap

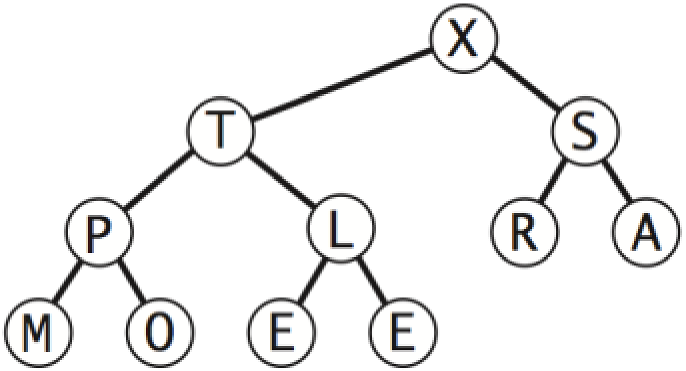
Step 2: Sortdown

Remove the maximum, one at a time

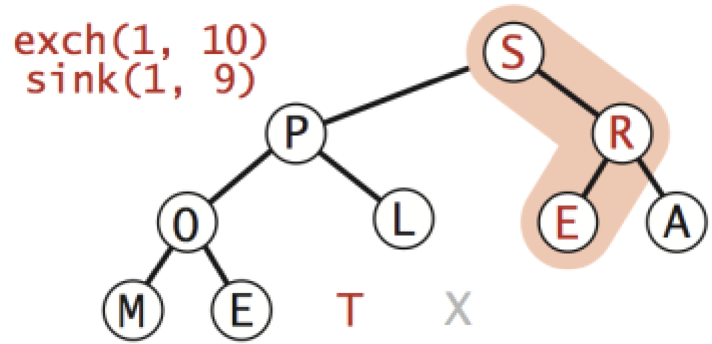
Leave in array, instead of nulling out.

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





Heap ordered array

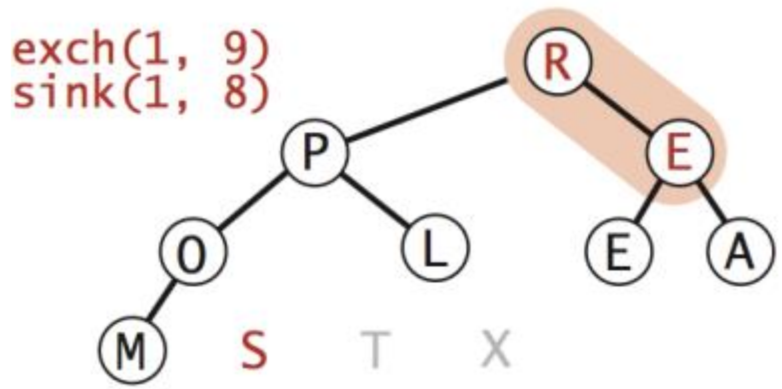


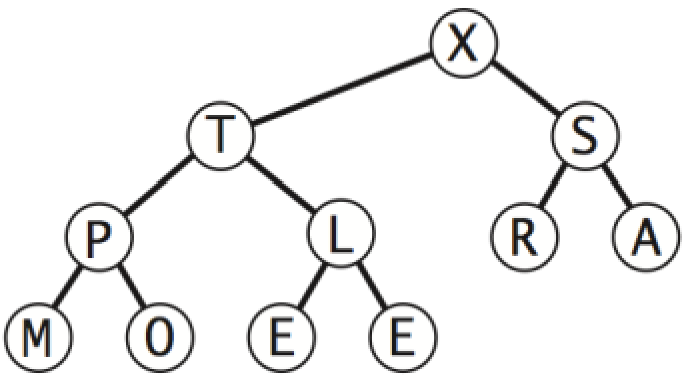
Step 2: Sortdown

Remove the maximum, one at a time

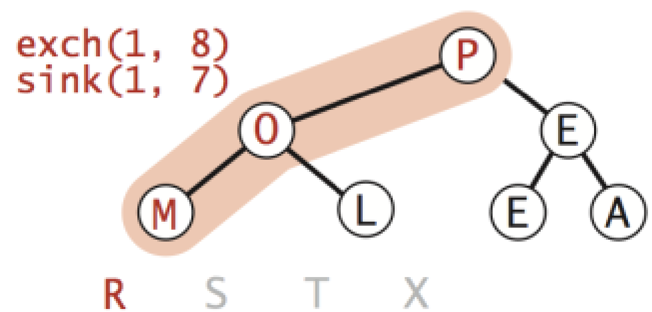
Leave in array, instead of nulling out.

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





Heap ordered array

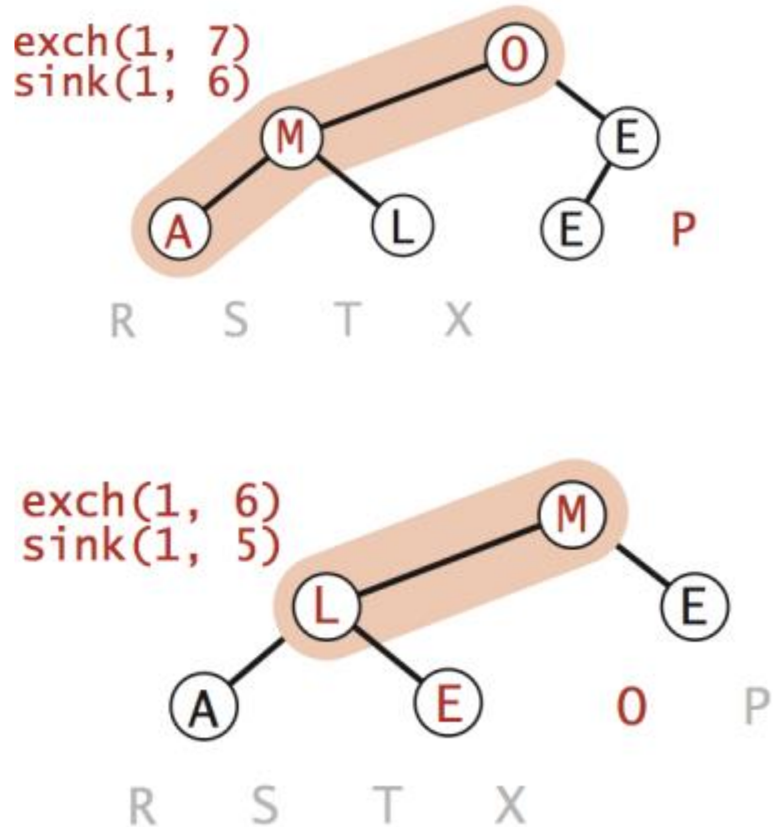


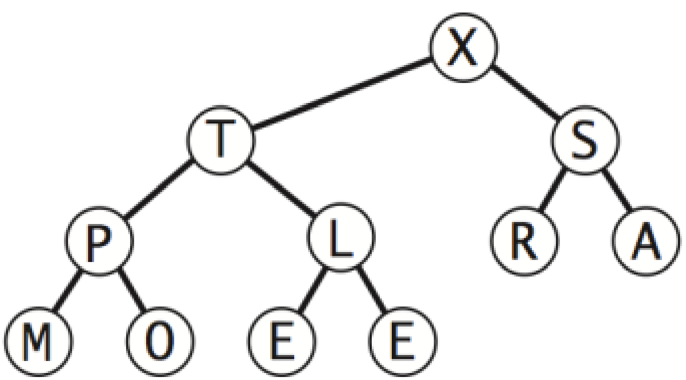
Step 2: Sortdown

Remove the maximum, one at a time

Leave in array, instead of nulling out.

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





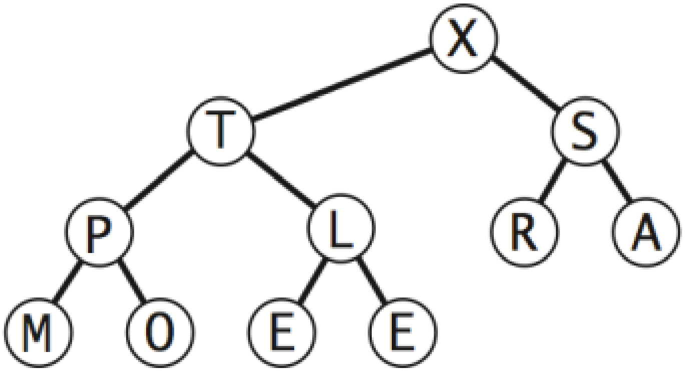
Heap ordered array

Step 2: Sortdown

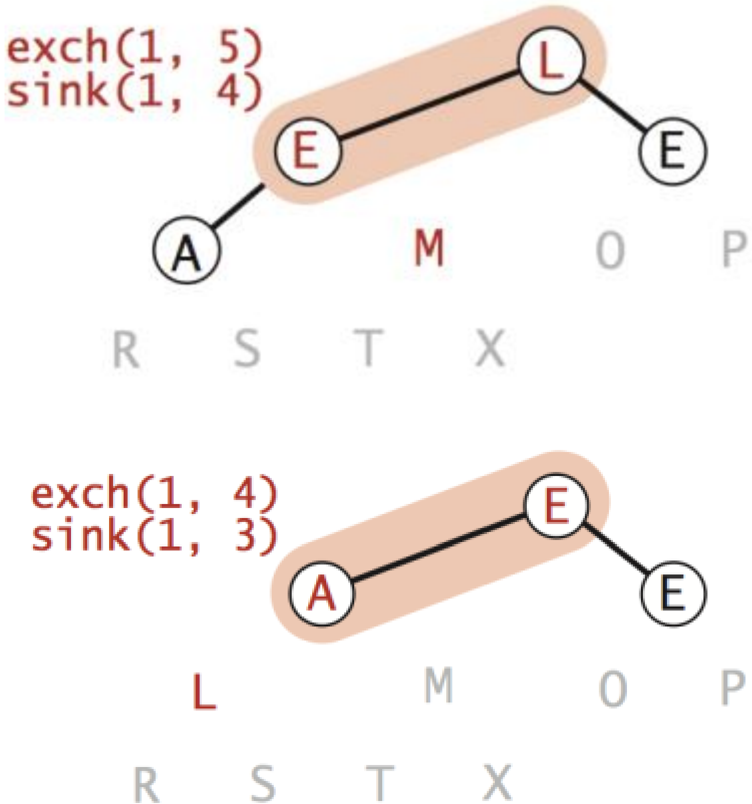
Remove the maximum, one at a time

Leave in array, instead of nulling out.

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



Heap ordered array

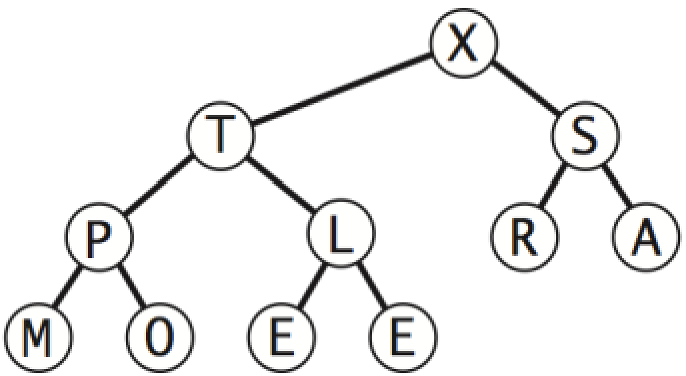


Step 2: Sortdown

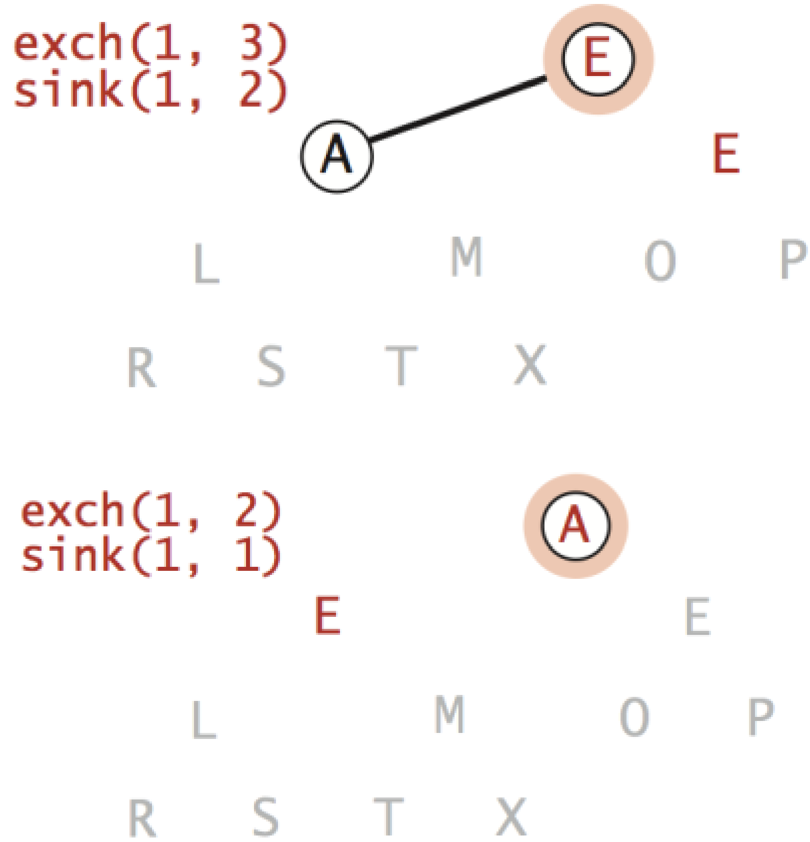
Remove the maximum, one at a time

Leave in array, instead of nulling out.

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



Heap ordered array

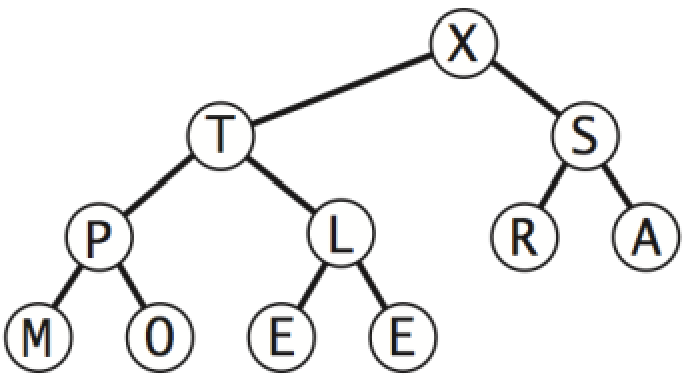


Step 2: Sortdown

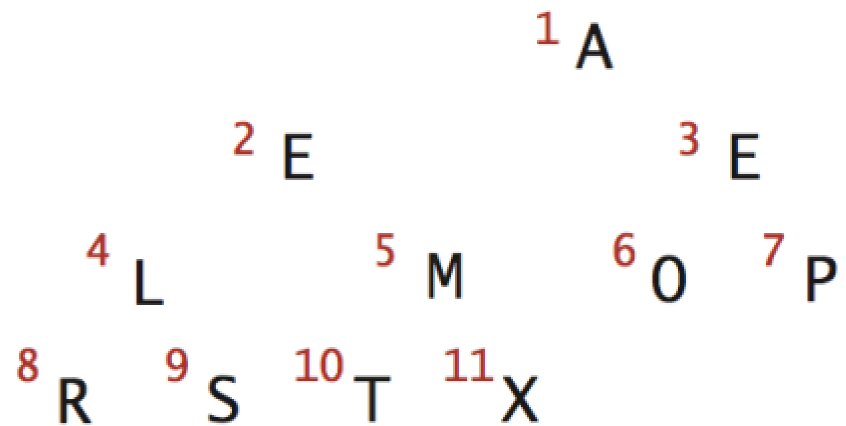
Remove the maximum, one at a time

Leave in array, instead of nulling out.

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



Heap ordered array



Sorted Result

--	A	E	E	L	M	O	P	R	S	T	X
0	1	2	3	4	5	6	7	8	9	10	11

Implementation and Demo

Mathematical Analysis

- Heap construction uses fewer than 2 N compares and exchanges.
- Heapsort uses at most 2 N lg N compares and exchanges.

Significance:

- In-place sorting algorithm with N log N worst-case.
- Mergesort: no, linear extra space.
- Quicksort: no, quadratic time in worst case.
- Heapsort: yes
- Heapsort is optimal for both time and space,

Disadvantages:

- Makes poor use of cache memory.
- Not stable.

Sorting Algorithms Comparison

	In palce	Stable	Worst	Average	Best
Quick Sort	✓		n^2	2NlogN	NlogN
Merge Sort		✓	NlogN	NlogN	NlogN
Heap Sort	✓		2NlogN	2NlogN	NlogN
?	✓	✓	NlogN	NlogN	NlogN