

كيفية المقارنة في الـ k value

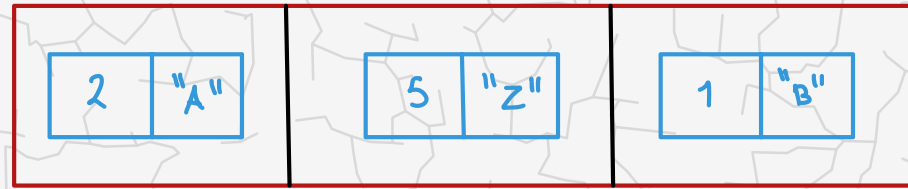
1	"B"
---	-----

2	"A"
---	-----

5	"Z"
---	-----

دقيقه به لازم  
اقله اذا اصغر  
اكثره اذا اكبر يكون

قله اذا اصغر  
اكثره اذا اكبر يكون  
نفسه بالاولويه  
نفسه بالارقام  
معلمنا من الارقام

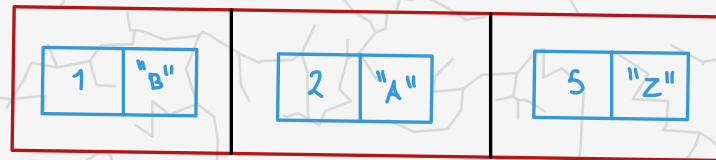


FIFO → first in first out  
منويجا حسب تاريخ

اولوية

## 2.4 PRIORITY QUEUES

- ▶ API and elementary implementations
- ▶ binary heaps
- ▶ heapsort
- ▶ event-driven simulation



<http://algs4.cs.princeton.edu>



<http://algs4.cs.princeton.edu>

## 2.4 PRIORITY QUEUES

---

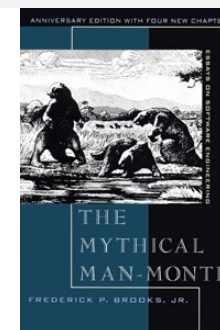
- *API and elementary implementations*
- *binary heaps*
- *heapsort*
- *event-driven simulation*

# Collections

A **collection** is a data types that store groups of items.

data type	key operations	data structure
<b>stack</b>	PUSH, POP	<i>linked list, resizing array</i>
<b>queue</b>	ENQUEUE, DEQUEUE	<i>linked list, resizing array</i>
<b>priority queue</b>	INSERT, DELETE-MAX <sup>أكبر</sup> <sub>اولوية</sub>	<i>binary heap</i>
<b>symbol table</b>	PUT, GET, DELETE	<i>BST, hash table</i>
<b>set</b>	ADD, CONTAINS, DELETE	<i>BST, hash table</i>

*“ Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won't usually need your code; it'll be obvious. ” — Fred Brooks*



# Priority queue

---

**Collections.** Insert and delete items. Which item to delete?

**Stack.** Remove the item most recently added.

**Queue.** Remove the item least recently added.

**Randomized queue.** Remove a random item.

**Priority queue.** Remove the largest (or smallest) item.


<i>operation</i>	<i>argument</i>	<i>return value</i>
<i>insert</i>	P	
<i>insert</i>	Q	
<i>insert</i>	E	
<i>remove max</i>		Q
<i>insert</i>	X	→ لأنه اكبر واحد
<i>insert</i>	A	
<i>insert</i>	M	
<i>remove max</i>		X
<i>insert</i>	P	
<i>insert</i>	L	
<i>insert</i>	E	
<i>remove max</i>		P

# Priority queue API

---

**Requirement.** Generic items are Comparable.

Key must be Comparable  
(bounded type parameter)



*Min*  
public class MaxPQ<Key extends Comparable<Key>>

MaxPQ()

*create an empty priority queue*

MaxPQ(Key[] a)

*create a priority queue with given keys*

void insert(Key v)

*insert a key into the priority queue*

Key delMax() → *deQueue*

*return and remove the largest key*

boolean isEmpty()

*is the priority queue empty?*

Key max()

*return the largest key*

int size()

*number of entries in the priority queue*

# Priority queue applications

---

- Event-driven simulation. [ customers in a line, colliding particles ]
- Numerical computation. [ reducing roundoff error ]
- Data compression. [ Huffman codes ]
- Graph searching. [ Dijkstra's algorithm, Prim's algorithm ]
- Number theory. [ sum of powers ]
- Artificial intelligence. [ A\* search ]
- Statistics. [ online median in data stream ]
- Operating systems. [ load balancing, interrupt handling ]
- Computer networks. [ web cache ]
- Discrete optimization. [ bin packing, scheduling ]
- Spam filtering. [ Bayesian spam filter ]

**Generalizes:** stack, queue, randomized queue.

# Priority queue elementary implementations

Challenge. Implement **all** operations efficiently.

time complex

implementation	① insert	② del max	③ max
<b>unordered array</b>	1	$N$	$N$
<b>ordered array</b>	$N$	1	1
<b>goal</b>	$\log N$	$\log N$	$\log N$

do not search

لان بمر عدد كل ال node.

order of growth of running time for priority queue with  $N$  items

$O(N)$   
 insert ←  
 constant  
 ordered data



<http://algs4.cs.princeton.edu>

## 2.4 PRIORITY QUEUES

---

- *API and elementary implementations*
- *binary heaps* ↳ complete binary tree
- *heapsort*
- *event-driven simulation*

# Complete binary tree

Binary tree. Empty or node with links to left and right binary trees. <sup>2 child</sup>

Complete tree. Perfectly balanced, except for bottom level.

heaps: Max heap: Node must be greater of the Children.

ماتسوي مقارنه بين اليمين واليسار

ماتسوي مقارنه ال node مع ال parent

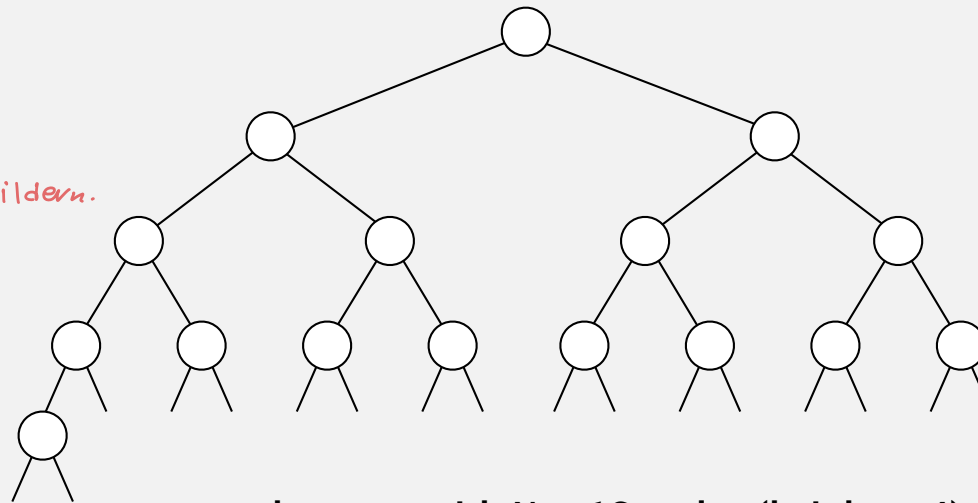
All node shift to right

Min: heap: node must be less of the Children.

always remove root.

$(\log_2 n)$

input: max, delet, insert, min  
space: O(N)



complete tree with  $N = 16$  nodes (height = 4)

Property. Height of complete tree with  $N$  nodes is  $\lceil \lg N \rceil$ .

Pf. Height increases only when  $N$  is a power of 2.

سويٹا ما Queue  
عاديہ

Q5. What does each removeMin call return within the following sequence of priority queue (min heap) ADT operations. Each node in the binary tree stores the priority key (integer) and the value (character).

insert(5, A),  
insert(4, B),  
insert(7, F),  
insert(1, D),  
removeMin(),  
insert(3, J),  
insert(6, L),  
removeMin(),  
removeMin(),  
insert(8, G),  
removeMin(),  
insert(2, H),  
removeMin(),  
removeMin()

(1, D) | (4, B) | (5, A) | (7, F)  
ليق ان لها اولوية

← يخدم على الKey

return: (1, D), (3, J), (4, B), (5, A), (2, H), (6, L)

سويٹا كذا علشان ما نتلخبط

(3, J) | (4, B) | (5, A) | (6, L) | (7, F) | (8, G) | (2, H)  
المفروضهم كدهم

return: (3, J), (4, B), (5, A)

(2, H) | (6, L) | (7, F) | (8, G)

return: (2, H), (6, L)

# A complete binary tree in nature

---



Hyphaene Compressa - Doum Palm

© Shlomit Pinter

# Binary heap representations

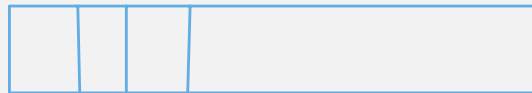
**Binary heap.** Array representation of a **heap-ordered** complete binary tree.

**Heap-ordered binary tree.**

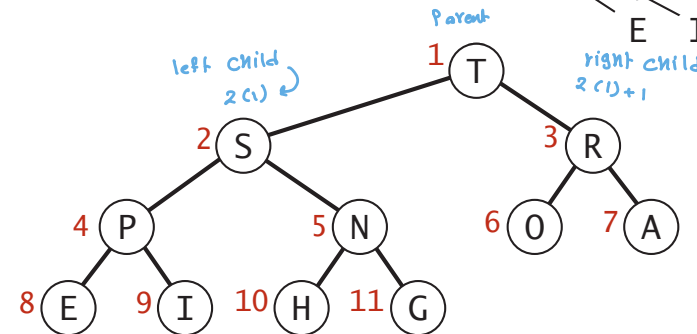
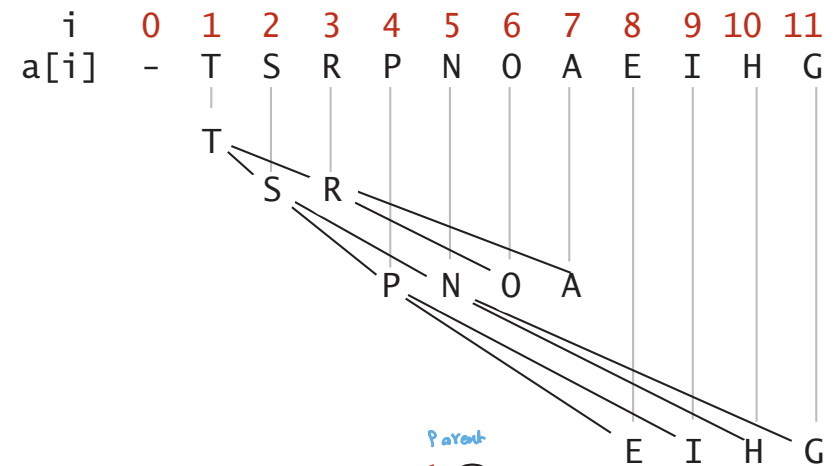
- Keys in nodes.
- Parent's key no smaller than children's keys.

*use*  
**Array representation.** *b/c it is complete*

- Indices start at 1.
- Take nodes in **level** order.
- No explicit links needed!



*always skip: zero index*



Heap representations

# Binary heap properties

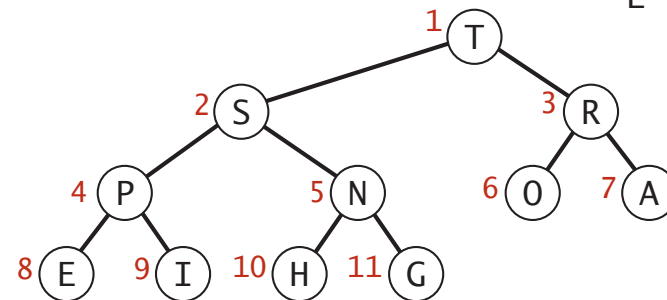
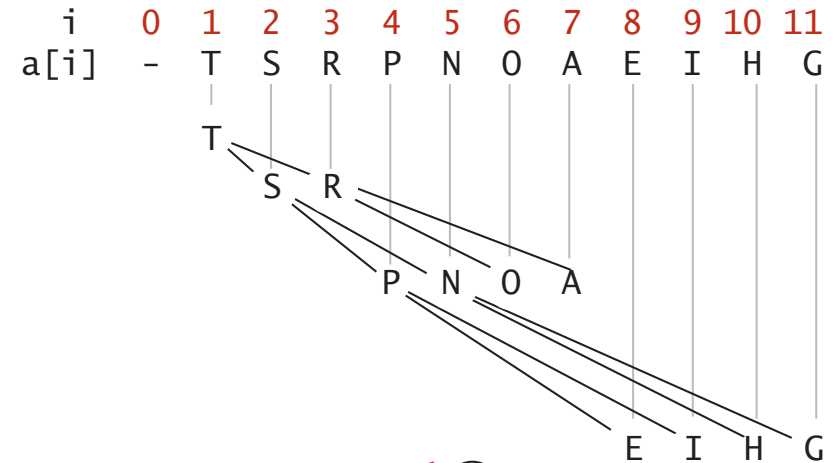
الاب اكبر من اولاده

**Proposition.** Largest key is  $a[1]$ , which is root of binary tree.

**Proposition.** Can use array indices to move through tree.

- Parent of node at  $k$  is at  $k/2$ .  $11/2 = 5 \rightarrow$  الاب
- Children of node at  $k$  are at  $2k$  and  $2k+1$ .

Heap by deflt  $\rightarrow$  Max heap ( $N \geq L, R$ )  
 if Minssion Min heap ( $N \leq L, R$ )  
 $R \leftrightarrow L$  No realtion don't care



Heap representations

# Binary heap demo

---

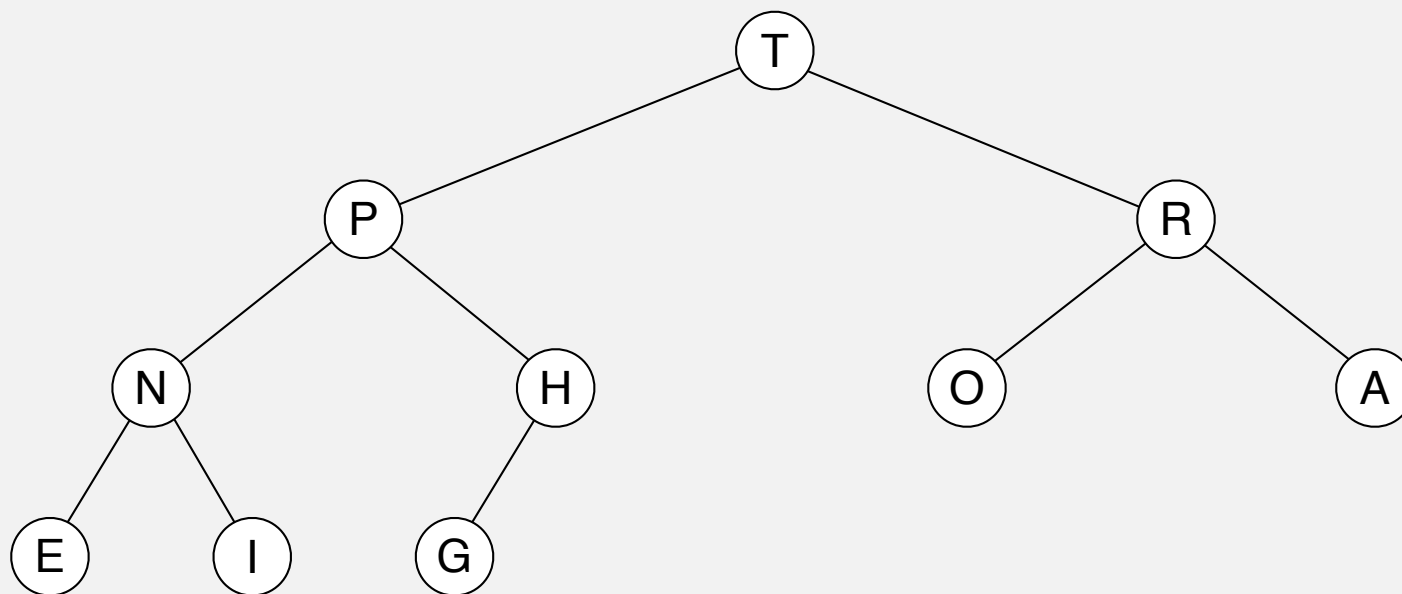
**Insert.** Add node at end, then **swim it up.**

**Remove the maximum.** Exchange root with node at end, then **sink it down.**

حذف

heap ordered

لازم مرتبه



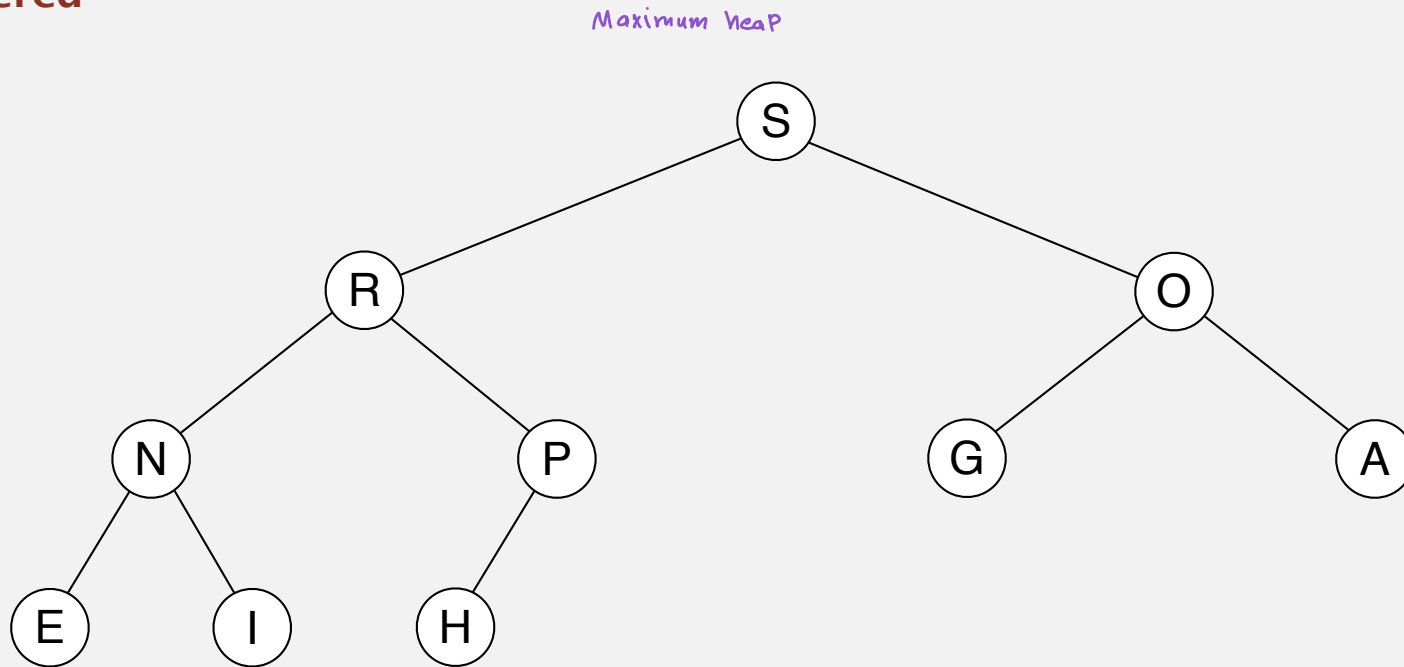
# Binary heap demo

---

**Insert.** Add node at end, then swim it up.

**Remove the maximum.** Exchange root with node at end, then sink it down.

**heap ordered**



# Promotion in a heap

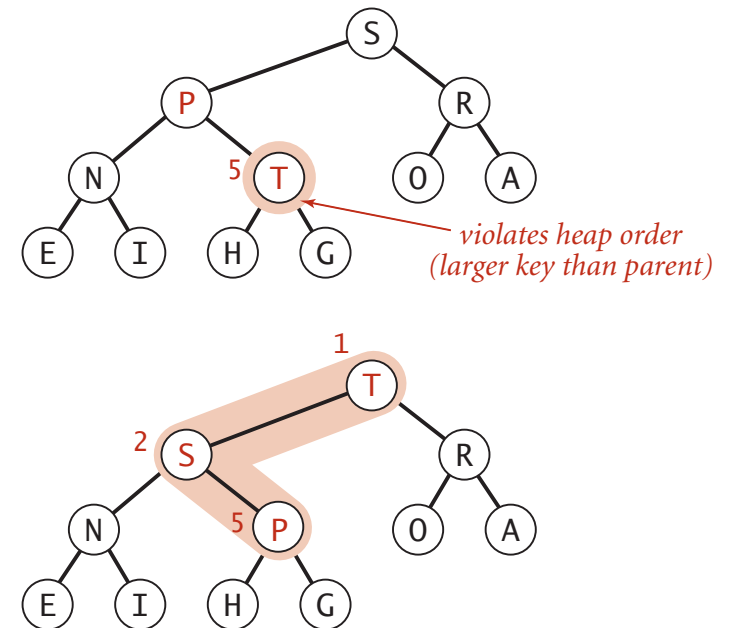
**Scenario.** Child's key becomes **larger** key than its parent's key.

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))
    {
        exch(k, k/2);
        k = k/2;
    }
}
```

parent of node at k is at k/2



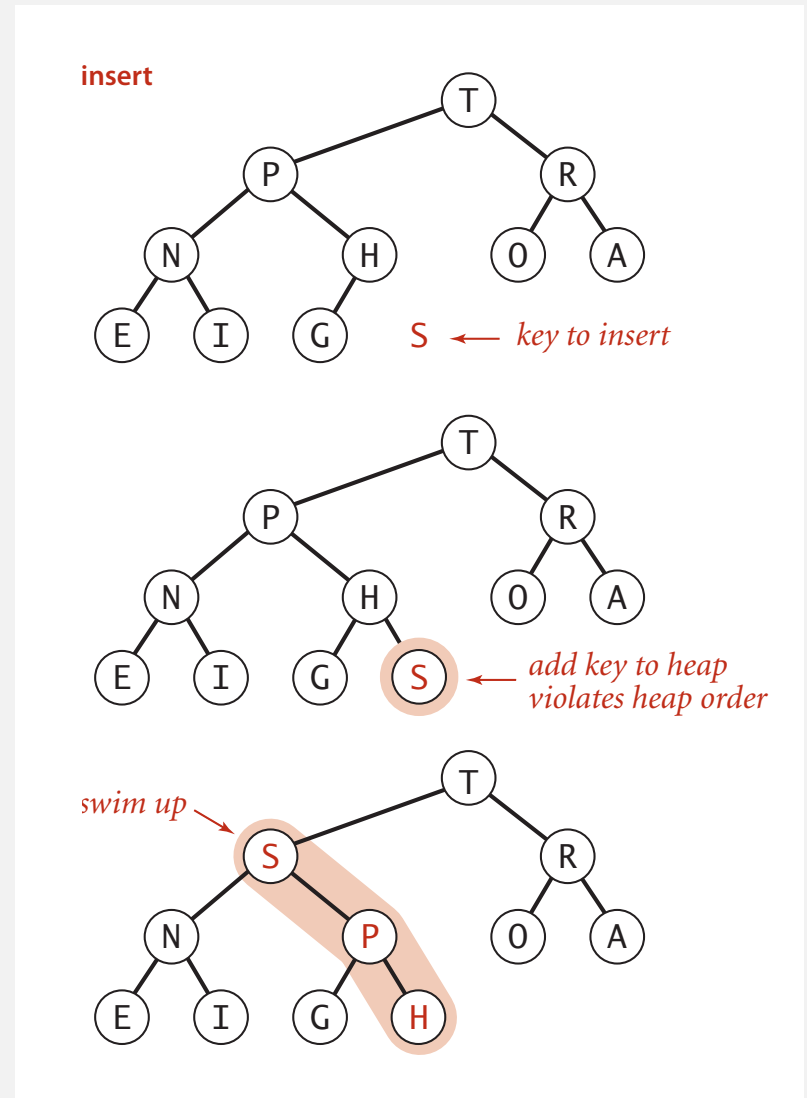
**Peter principle.** Node promoted to level of incompetence.

# Insertion in a heap

**Insert.** Add node at end, then swim it up.

**Cost.** At most  $1 + \lg N$  compares.

```
public void insert(Key x)
{
    pq[++N] = x;
    swim(N);
}
```



# Demotion in a heap

Scenario. Parent's key becomes **smaller** than one (or both) of its children's.

To eliminate the violation:

why not smaller child?

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

```
private void sink(int k)
```

```
{
```

```
  while (2*k <= N)
```

```
  {
```

```
    int j = 2*k;
```

```
    if (j < N && less(j, j+1)) j++;
```

```
    if (!less(k, j)) break;
```

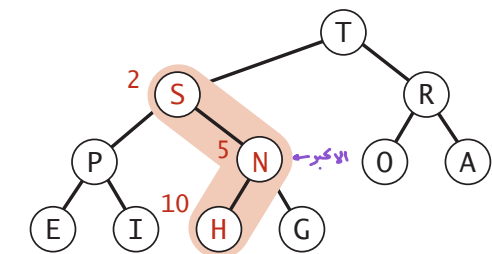
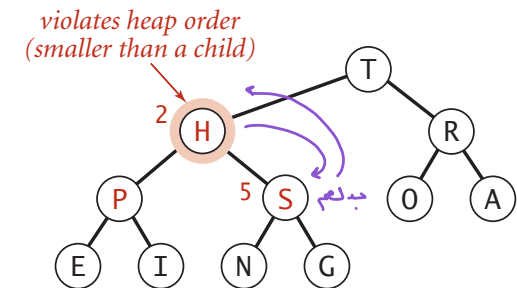
```
    exch(k, j);
```

```
    k = j;
```

```
  }
```

```
}
```

children of node at k are  
2k and 2k+1



Top-down reheapify (sink)

بفريقنا من تحت واي Swime UP اجعله فوق  
ولكن احذف - احذف الـ node وايضا مكان root وديتجا.

Power struggle. Better subordinate promoted.

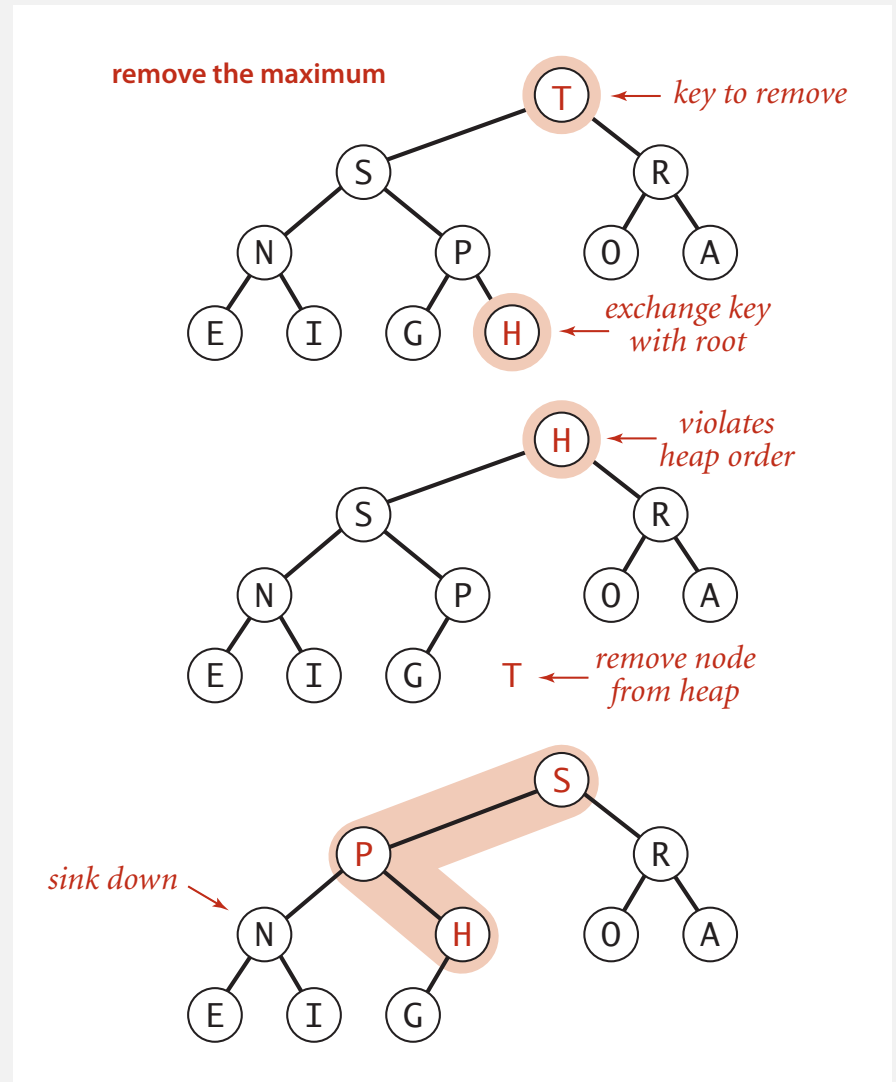
# Delete the maximum in a heap

**Delete max.** Exchange root with node at end, then sink it down.

**Cost.** At most  $2 \lg N$  compares.

```
public Key delMax()
{
    Key max = pq[1];
    exch(1, N--);
    sink(1);
    pq[N+1] = null;
    return max;
}
```

*index root* ← `exch(1, N--);`  
*Delet root* ← `Key max = pq[1];`  
*of last node* ← `N--`  
`prevent loitering` ← `pq[N+1] = null;`



# Binary heap: Java implementation

---

```
public class MaxPQ<Key extends Comparable<Key>>
{
    private Key[] pq;
    private int N;
```

```
    public MaxPQ(int capacity)
    { pq = (Key[]) new Comparable[capacity+1]; }
```

← fixed capacity  
(for simplicity)

```
    public boolean isEmpty()
    { return N == 0; }
    public void insert(Key key)
    public Key delMax()
    { /* see previous code */ }
```

*now match element we have.*

← PQ ops

```
    private void swim(int k)
    private void sink(int k)
    { /* see previous code */ }
```

← heap helper functions

```
    private boolean less(int i, int j)
    { return pq[i].compareTo(pq[j]) < 0; }
    private void exch(int i, int j)
    { Key t = pq[i]; pq[i] = pq[j]; pq[j] = t; }
```

← array helper functions

```
}
```

# Priority queues implementation cost summary

---

*one element*

implementation	insert	del max	max
<b>unordered array</b>	1	$N$	$N$
<b>ordered array</b>	$N$	1	1
<b>binary heap</b>	$\log N$	$\log N$	1

**order-of-growth of running time for priority queue with  $N$  items**