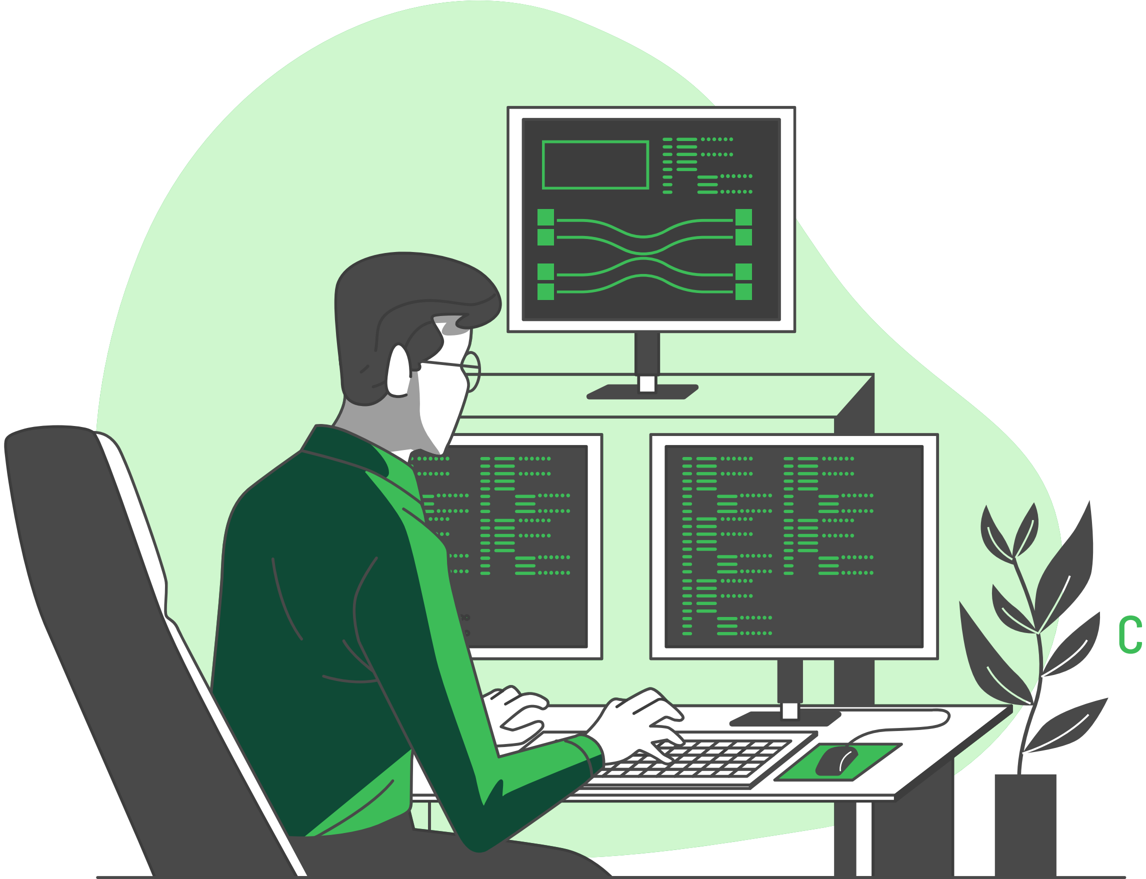
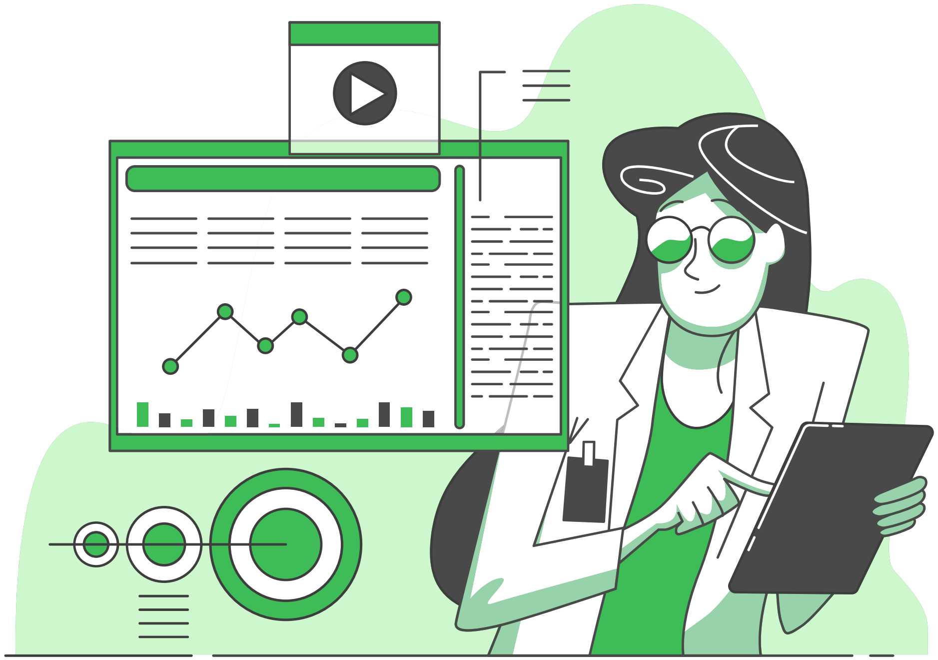


Binary Heap

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents



01
Binary Heap

Queue with priority

Think About This

We insert multiple random items into a data structure without following a specific order. We need to find the min or max item. How do we do it?

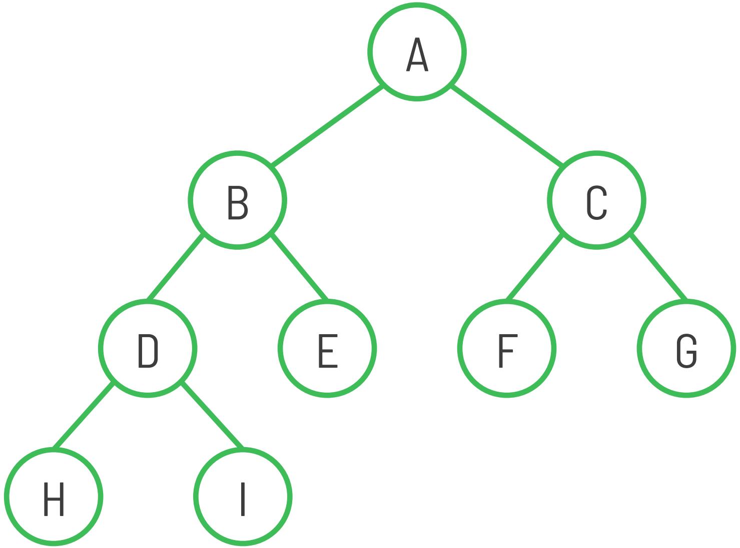
Sorting and then get min/max?

$$O(n \log(n)) + O(1) \in O(n \log(n))$$

Ordered insertion then get min/max?

$$O(n) + O(1) \in O(n)$$

Binary Heap

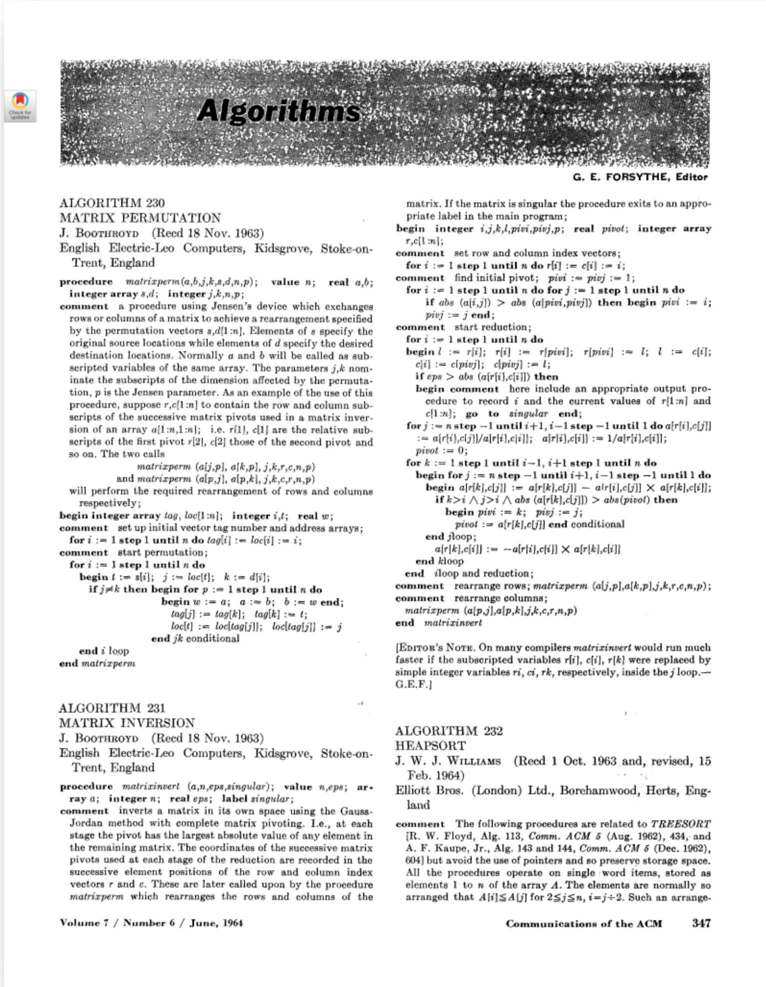


A binary tree-based data structure such that:

- The binary tree is **complete**.
- It is **heap-ordered**:
- Min-heap order: The parent's item is less than the children's items.
- Max-heap order: The parent's item is greater than the children's items.

Note: Items must be comparable to answer questions of the form: is $A < B$?

"Algorithm 232 – Heapsort“, J. W. J. Williams, "Communications of the ACM“, 1964



Algorithms

G. E. FORSYTHE, Editor

ALGORITHM 230
MATRIX PERMUTATION
J. Boothroyd (Recd 18 Nov. 1963)
English Electric-Leo Computers, Kidsgrove, Stoke-on-Trent, England

```
procedure matrixperm(a,b,j,k,s,d,n,p); value n; real a,b;
  integer array s,d; integer j,k,n,p;
comment a procedure using Jensen's device which exchanges
  rows or columns of a matrix to achieve a rearrangement specified
  by the permutation vectors s,d[1:n]. Elements of s specify the
  original source locations while elements of d specify the desired
  destination locations. Normally a and b will be called as sub-
  scripted variables of the same array. The parameters j,k nom-
  inate the subscripts of the dimension affected by the permuta-
  tion, p is the Jensen parameter. As an example of the use of this
  procedure, suppose r,c[1:n] to contain the row and column sub-
  scripts of the successive matrix pivots used in a matrix inver-
  sion of an array a[1:n,1:n]; i.e. r[1], c[1] are the relative sub-
  scripts of the first pivot r[2], c[2] those of the second pivot and
  so on. The two calls
              matrixperm (a[j,p], a[k,p], j,k,r,c,n,p)
          and matrixperm (a[p,j], a[p,k], j,k,c,r,n,p)
  will perform the required rearrangement of rows and columns
    respectively;
begin integer array tag, loc[1:n]; integer i,t; real w;
comment set up initial vector tag number and address arrays;
  for i := 1 step 1 until n do tag[i] := loc[i] := i;
comment start permutation;
  for i := 1 step 1 until n do
    begin t := s[i]; j := loc[t]; k := d[i];
      if j≠k then begin for p := 1 step 1 until n do
                        begin w := a; a := b; b := w end;
                          tag[j] := tag[k]; tag[k] := t;
                          loc[t] := loc[tag[j]]; loc[tag[j]] := j
                      end jk conditional
    end i loop
end matrixperm
```

ALGORITHM 231
MATRIX INVERSION
J. Boothroyd (Recd 18 Nov. 1963)
English Electric-Leo Computers, Kidsgrove, Stoke-on-Trent, England

```
procedure matrixinvert (a,n,eps,singular); value n,eps; ar-
  ray a; integer n; real eps; label singular;
comment inverts a matrix in its own space using the Gauss-
  Jordan method with complete matrix pivoting. I.e., at each
  stage the pivot has the largest absolute value of any element in
  the remaining matrix. The coordinates of the successive matrix
  pivots used at each stage of the reduction are recorded in the
  successive element positions of the row and column index
  vectors r and c. These are later called upon by the procedure
  matrixperm which rearranges the rows and columns of the
  matrix. If the matrix is singular the procedure exits to an appro-
  priate label in the main program;
begin integer i,j,k,l,pivi,pivj,p; real pivot; integer array
  r,c[1:n];
comment set row and column index vectors;
  for i := 1 step 1 until n do r[i] := c[i] := i;
comment find initial pivot; pivi := pivj := 1;
  for i := 1 step 1 until n do for j := 1 step 1 until n do
      if abs (a[i,j]) > abs (a[pivi,pivj]) then begin pivi := i;
      pivj := j end;
comment start reduction;
  for i := 1 step 1 until n do
  begin l := r[i]; r[i] := r[pivi]; r[pivi] := l; l := c[i];
    c[i] := c[pivj]; c[pivj] := l;
    if eps > abs (a[r[i],c[i]]) then
    begin comment here include an appropriate output pro-
      cedure to record i and the current values of r[1:n] and
      c[1:n]; go to singular end;
  for j := n step −1 until i+1, i−1 step −1 until 1 do a[r[i],c[j]]
    := a[r[i],c[j]]/a[r[i],c[i]]; a[r[i],c[i]] := 1/a[r[i],c[i]];
    pivot := 0;
  for k := 1 step 1 until i−1, i+1 step 1 until n do
    begin for j := n step −1 until i+1, i−1 step −1 until 1 do
      begin a[r[k],c[j]] := a[r[k],c[j]] − a[r[i],c[j]] × a[r[k],c[i]];
        if k>i ∧ j>i ∧ abs (a[r[k],c[j]]) > abs(pivot) then
          begin pivi := k; pivj := j;
            pivot := a[r[k],c[j]] end conditional
      end jloop;
        a[r[k],c[i]] := −a[r[i],c[i]] × a[r[k],c[i]]
    end kloop
  end  iloop and reduction;
comment rearrange rows; matrixperm (a[j,p],a[k,p],j,k,r,c,n,p);
comment rearrange columns;
  matrixperm (a[p,j],a[p,k],j,k,c,r,n,p)
end matrixinvert
```

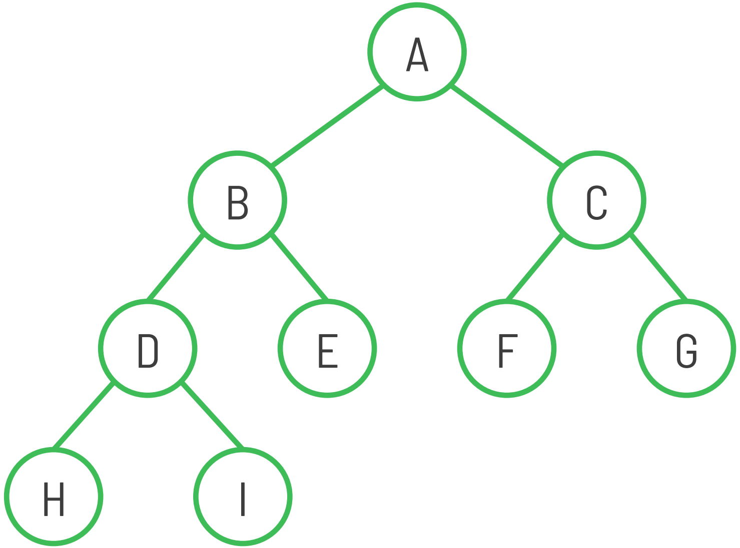
[Editor's Note. On many compilers *matrixinvert* would run much faster if the subscripted variables r[i], c[i], r[k] were replaced by simple integer variables *ri*, *ci*, *rk*, respectively, inside the *j* loop.—G.E.F.]

ALGORITHM 232
HEAPSORT
J. W. J. Williams (Recd 1 Oct. 1963 and, revised, 15 Feb. 1964)
Elliott Bros. (London) Ltd., Borehamwood, Herts, England

```
comment The following procedures are related to TREESORT
  [R. W. Floyd, Alg. 113, Comm. ACM 5 (Aug. 1962), 434, and
  A. F. Kaupe, Jr., Alg. 143 and 144, Comm. ACM 5 (Dec. 1962),
  604] but avoid the use of pointers and so preserve storage space.
  All the procedures operate on single word items, stored as
  elements 1 to n of the array A. The elements are normally so
  arranged that A[i]≤A[j] for 2≤j≤n, i=j÷2. Such an arrange-
```


https://dl.acm.org/doi/10.1145/512274.512284

Binary Heap

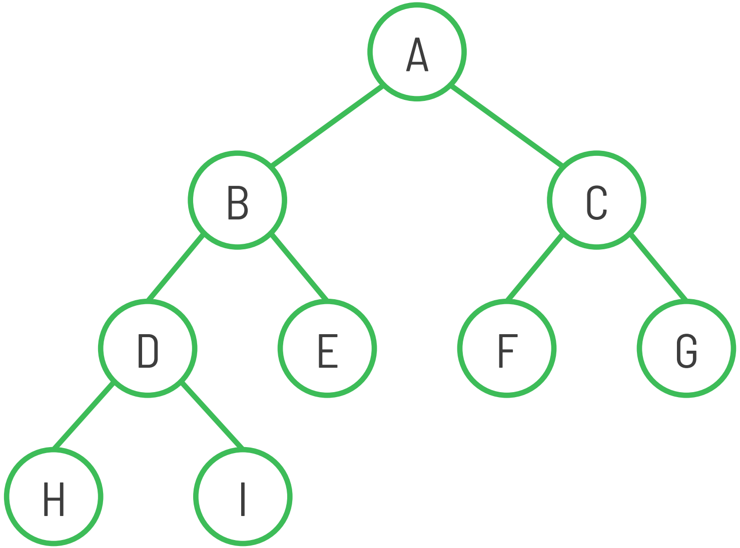


What did we say about **balanced binary trees**? Many things that lead to:

$$h \in \Theta(\log_2(n))$$

So, every **complete binary tree** is **balanced**!

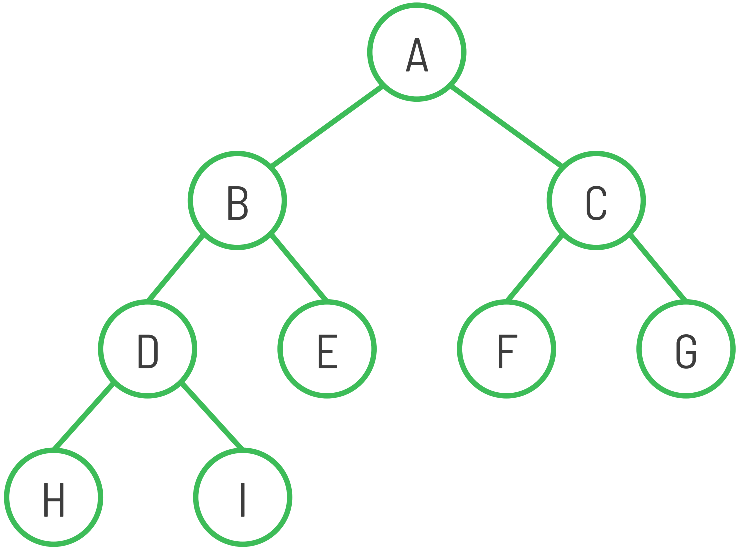
Binary Heap



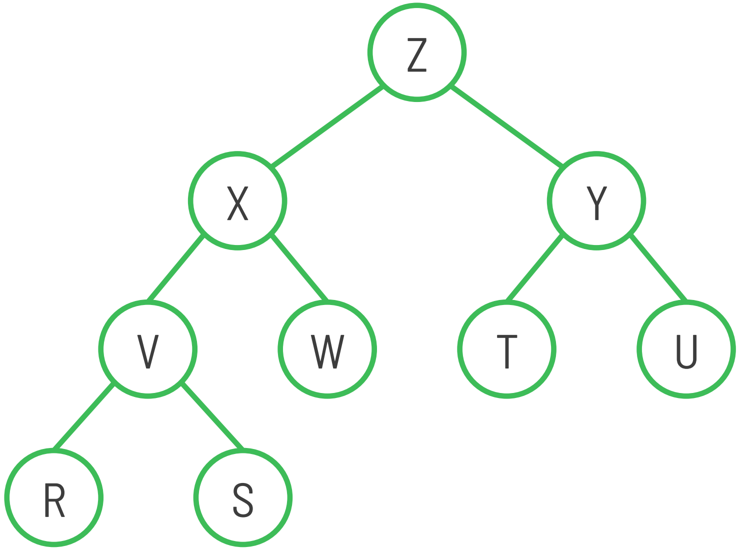
Max-heap (aka **Max Priority Queue**) if the key in each node is **larger than** or equal to the keys in that node's two children (if any).

Min-heap (aka **Min Priority Queue)** if the key in each node is **less than** or equal to the keys in that node's two children (if any).

Min-Heap



Max-Heap



Binary Heap ADT

- **insert(item):** Inserts the item in the heap and moves it into its right place.
- **[del, get][min, max]():** Removes and returns the next item in the heap.
- **isempty():** Checks whether the heap is empty or not.
- **size():** Returns the number of items in the heap.
- **peek():** Returns the next item in the heap without removing it.

Idea

Insertion on Binary Heaps

Insert each item in the next available location in a heap, then swim up the item until it reaches its proper place according to the heap order.

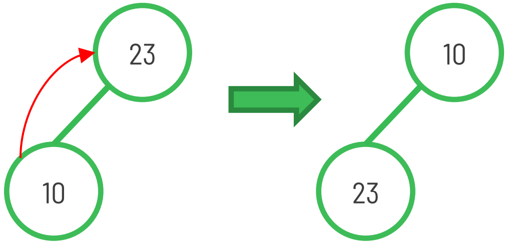
Each insertion is $O(\log_2(n))$
Why? It traverses a single path of a complete binary tree

…

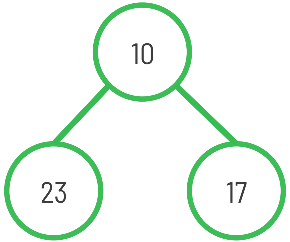
Insert into a Min-Heap: 23, 10, 17, 28, 34, 89, 22, 9

Insert 23:

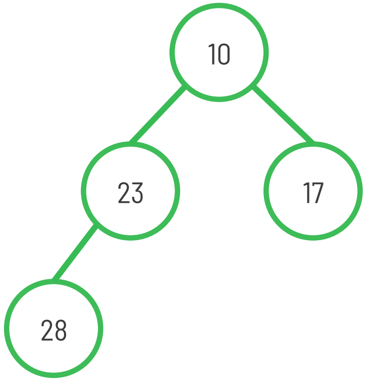
Insert 10:



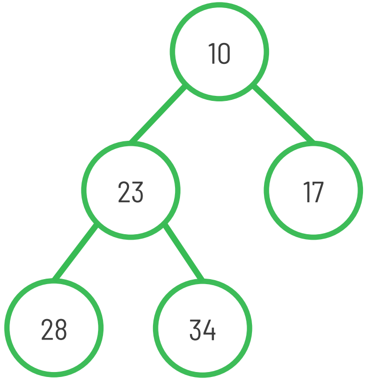
Insert 17:



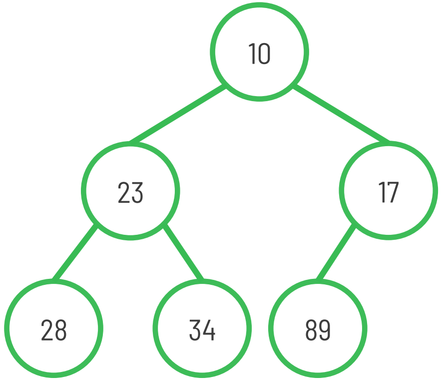
Insert 28:



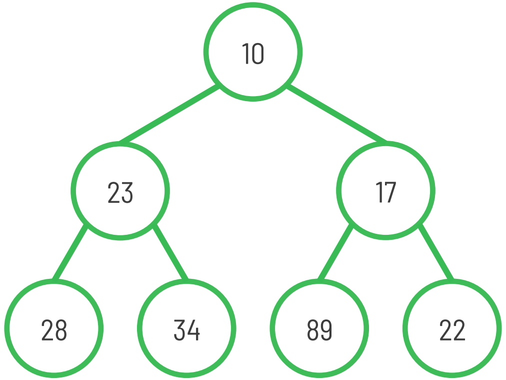
Insert 34:



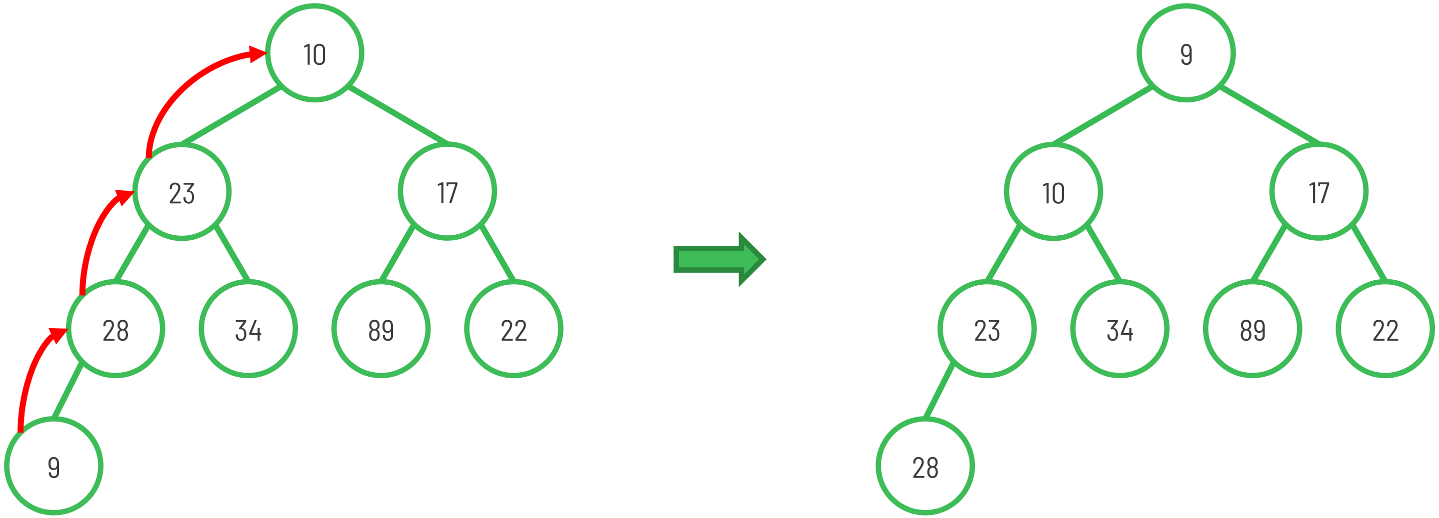
Insert 89:



Insert 22:



Insert 9:



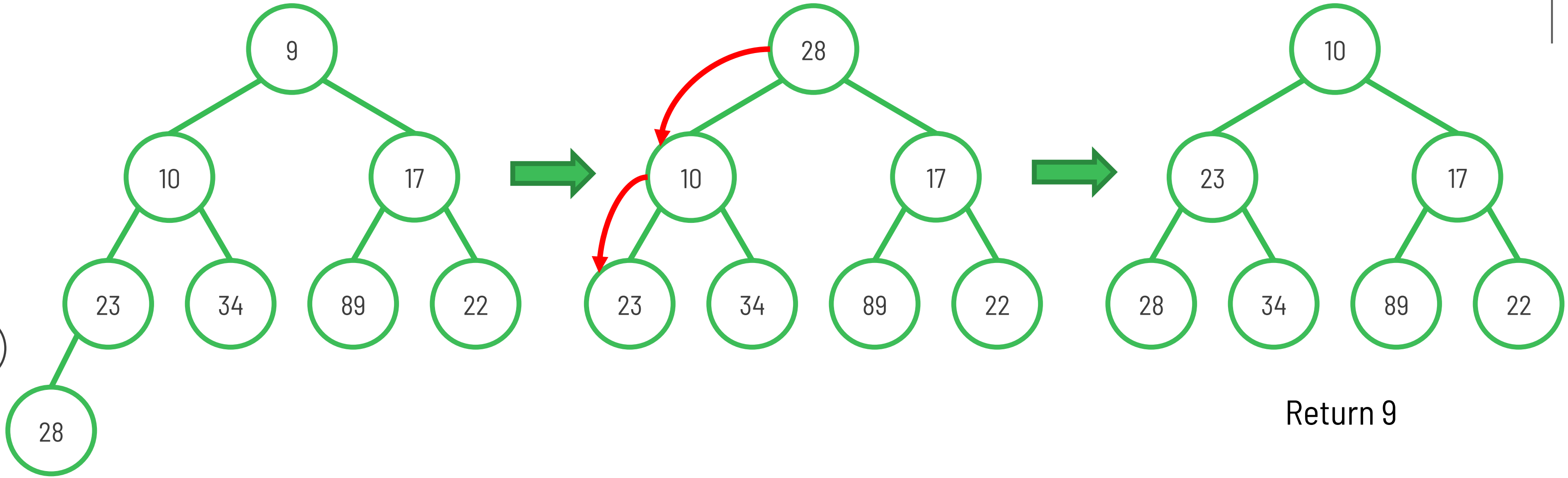
Idea

Delete/Get Min/Max

Put aside the item at the top of the heap. Move the last item in the heap to the top, then sink down the item until it reaches its proper place according to the heap order. Finally, return the previous top value.

Each deletion is $O(\log_2(n))$

Why? It traverses a single path of a complete binary tree getMin():



Return 9

Examples of Priority Queues

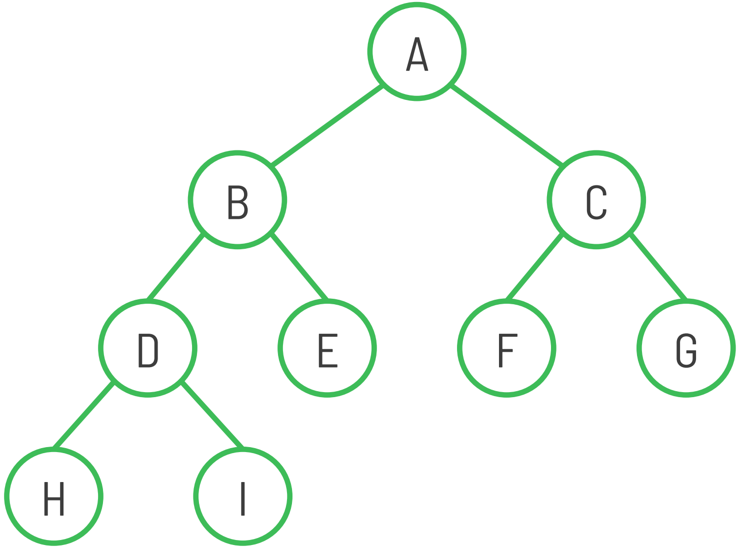
1. Emergency clinics process patients with different emergency levels. Some patients require immediate attention, while others may wait a bit longer.

2. You are in line for the next cashier to pay for groceries. Someone with fewer items than you is behind you in line. You, out of the goodness of your heart, tell the person to go ahead of you in line.

3. Operating systems have process priority scheduling. Priorities based on technical quantities (memory usage, I/O operations, sleeping time), politics, or user preference.

How Do We Implement A Binary Heap?

Array Implementation



	0	1	2	3	4	5	6	7	8
BH	A	B	C	D	E	F	G	H	I

$$\mathrm{leftchild}(i \in \mathbb{Z}_{\geq 0}) := 2i + 1$$

$$\mathrm{rightchild}(i \in \mathbb{Z}_{\geq 0}) := 2i + 2$$

$$\mathrm{parent}(i \in \mathbb{Z}^{+}) := \left\lfloor \frac{i - 1}{2} \right\rfloor$$

For implementation purposes, you must handle parent(0) as a corner case.

Insertion in a Min-Heap

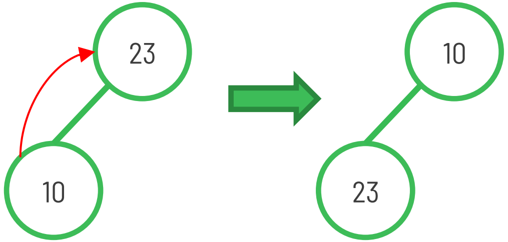
```
algorithm insert(A:array, X:item)

   let i be A’s next available index
   A[i] ← X
   p ← parent(i)

   while i > 0 and A[i] < A[p] do
      swap(A, i, p)
      i ← p
      p ←  parent(p)
   end while

end algorithm
```

Insert into a Min-Heap: 23, 10, 17, 28, 34, 89, 22, 9

Insert 23:

0	1	2	3	4	5	6	7
23							

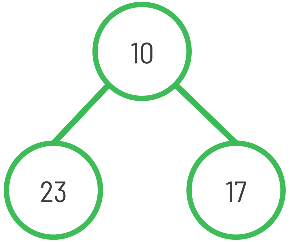
Insert 10:



0	1	2	3	4	5	6	7
23	10						

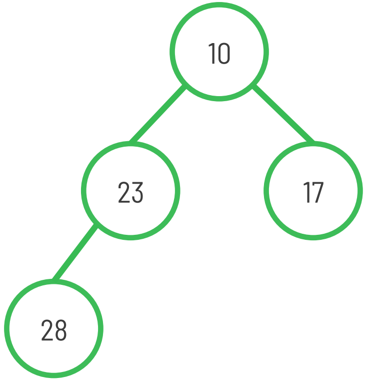
0	1	2	3	4	5	6	7
10	23						

Insert 17:



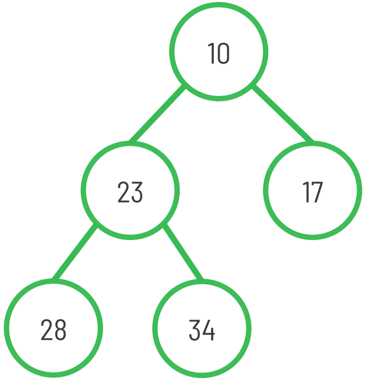
0	1	2	3	4	5	6	7
10	23	17					

Insert 28:



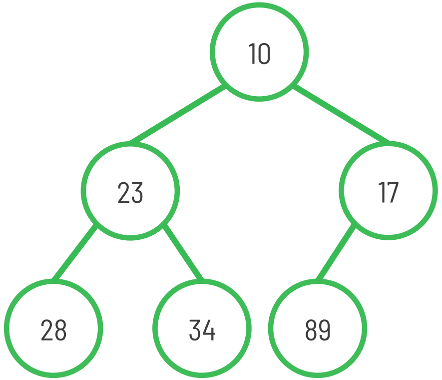
0	1	2	3	4	5	6	7
10	23	17	28				

Insert 34:



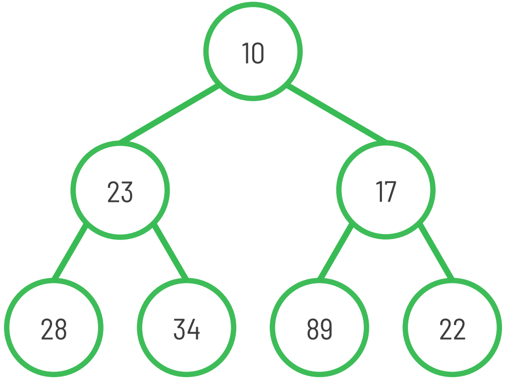
0	1	2	3	4	5	6	7
10	23	17	28	34			

Insert 89:



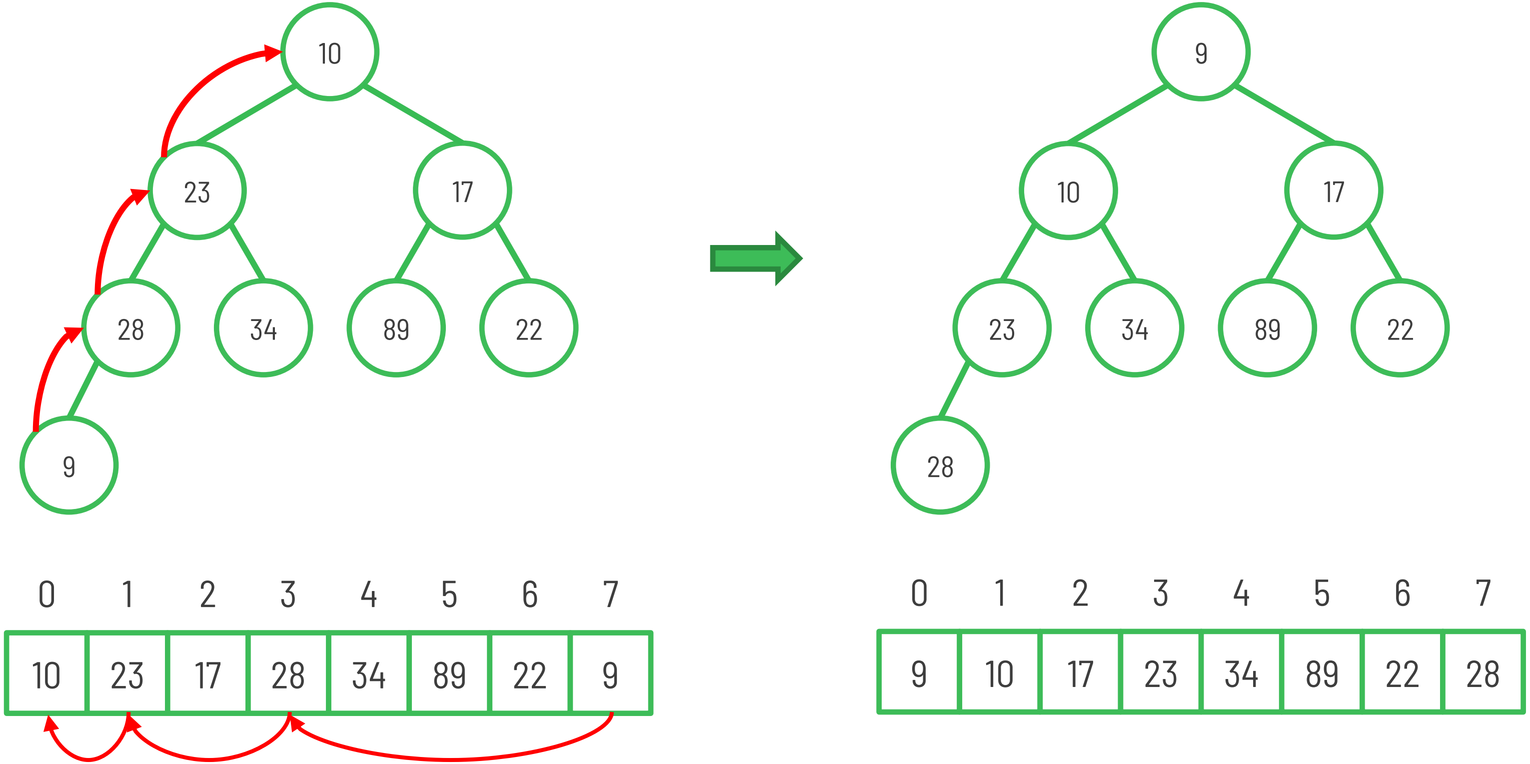
0	1	2	3	4	5	6	7
10	23	17	28	34	89		

Insert 22:



0	1	2	3	4	5	6	7
10	23	17	28	34	89	22	

Insert 9:


10
23
17
28
34
89
22
9
0 1 2 3 4 5 6 7
10 23 17 28 34 89 22 9
9
10
17
23
34
89
22
28
0 1 2 3 4 5 6 7
9 10 17 23 34 89 22 28

GetMin

```
algorithm getmin(A:array) → item
   throw an exception if A is empty
   t ← A[0]
   let n be size of A
   A[0] ← A[n-1]
   n ← n−1
   i ← 0
   min ← minchild(A, i)
   while min < n and A[i] > A[min] do
      swap(A, i, min)
      i ← min
      min ←  minchild(A, i)
   end while
   return t
end algorithm
```

Think carefully about minchild(A:array, i:$\mathbb{Z}_{\geq 0}$) $\longrightarrow$?. What value should it return if the left child of i does not exist?

Keep In Mind

Swim Up and **Sink Down** (AKA. **Sift Up** and **Sift Down**) functions are almost the same for Min-Heaps and Max-Heaps. The difference is the comparison signs to preserve the respective heap order.

Due to their **fast runtime complexity** $O(\log_2(n))$ for both insertion and deletion, Min/Max binary heaps are used as a fundamental data structure for more sophisticated data structures and algorithms.

Done!

Do you have any questions?