Recap :

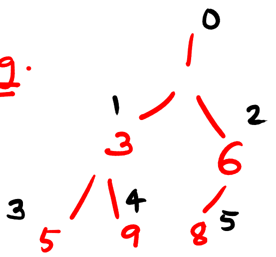
BT → how to deal with trees (structure)

BST → BT + 1 constraint : all nodes in left ST < root node < all nodes in right ST

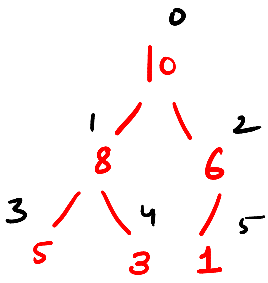
Heaps : BT* + 1 constraint : root node ≤ all nodes in left ST and right ST . minHeap *

≥ " " maxHeap

Eg.



minHeap



MaxHeap .

Represent .

class Node {
int data
Node left
Node right } ✗

Heaps are much better represented using arrays !!

new_node = new Node(4)

i parent

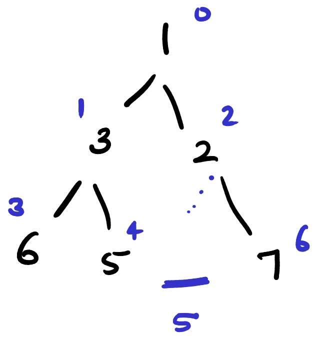
2i+1 2i+2

arr as an input. { minHeap = [1 3 6 5 9 8] ; maxHeap = [10 8 6 5 3 1] }

sorted

almost sorted BST : inorder

Heaps : level order



X complete .

minHeap = [1 3 2 6 5 $ 7]

No!! we have to create heap .

Ensure that our tree is complete.*

Hint: 10 elements [0 to 9] 10 elements

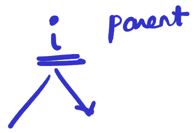
delete 3 elements .

[0 to 6 $$$] 7 elements

Interest : level order traversal arrays. arr = [ ]*.

Heaps → LoT → almost sorted array .

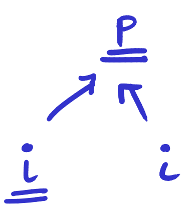
Heapsort $O(n\log n)$; Sort it



left (i) → returns 2i+1

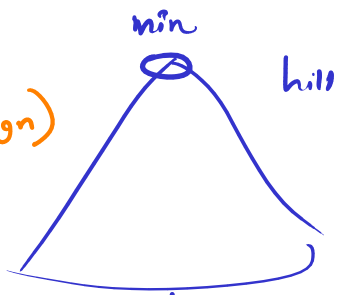
right(i) → return 2i+2

parent(i) → returns $\frac{i-1}{2}$



$\frac{4}{2} = 2$

$\frac{5}{2} = 2$

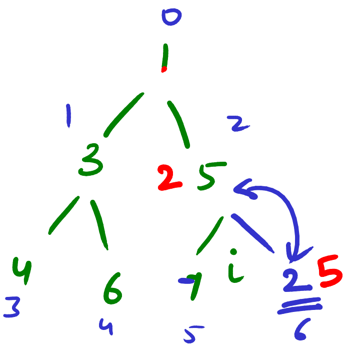


$2p+1 = i$ $p = \frac{i-1}{2}$

CRUD : 1. Create.

a) We have a heap and we want to perform insert. Can I do better? Yes.

Insert x n



minheap.

Insert 2. (key)

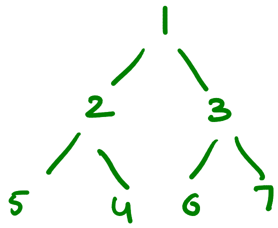
Heap Constraint.

i/p. [1 3 5 4 6 7 2] (leaf node)

[1 3 5 4 6 7] →

→ shift everyone.

thought: [1 2 3 5 4 6 7]



O(n)

fine!

index of last element = size of arr − 1

i = n−1

while (i != 0 and arr[parent(i)] > arr[i])

Swap (arr[parent (i)] , arr[i])

i = parent(i)

arr = [_________ key] n
heap ↑ n−1

$\frac{6-1}{2} = 2$ $\frac{2-1}{2} = 0$

T.C. ⇒ O(log n)

Insert key

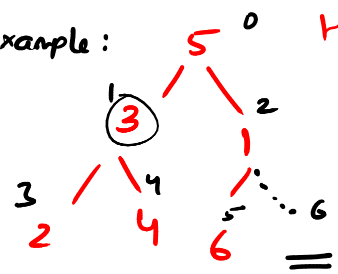
Time to create a heap : O(n log n)

You assume : input is given as a stream. 5 3 1 2 4 6

shuffled array / any array = [5 3 1 2 4 6] → n
↑ heapsize

array
make a heap / heapify ↓

example:



Heap? ✗

= 0

↓

minheapify (index)

T.C.

O(n)

arr → global

[_________]
heap

void minheapify (int i) :
    l = left(i) , r = right (r)
    smallest = i
    if l < heap.size and arr[l] < arr[i] :
        smallest = l
    if r < heap.size and arr[r] < arr[smallest]:
        smallest = r

→ Corrects Branch.

2i+1

i 2. left(2) = 5

right(2) = 6

2i+2

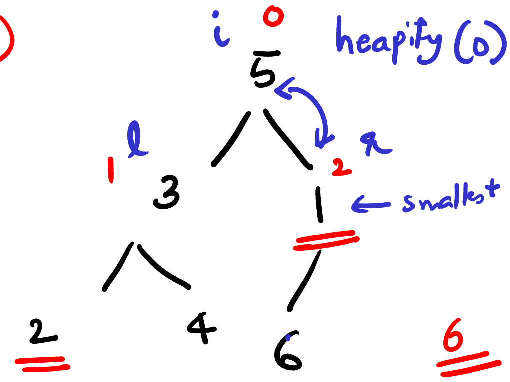
arr[i] arr[l] arr[r]

if smallest != i :

{ swap (arr[i], arr[smallest])

minheapify (smallest) ;

$O(\log n)$

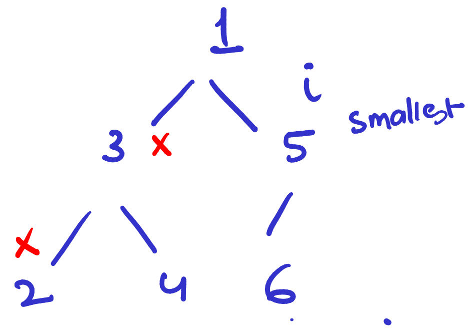


heap size = n

index of last non-leaf / internal

node : $n-1/2$ $\frac{6-1}{2} = 2$

min Heap ? No!



buildheap :

for $j = \frac{n-1}{2}$ $j >= 0$ $j--$

minheapify (j)

⇒ Ensure : Heap !!

Stackoverflow !!

loop : $\frac{n}{2} \times O(\log n)$ ⟶ $O(n \log n)$???

$O(n)$

Intuition : Once a branch is corrected, you dont make recursive call that often.

$O(2n)$

$O(n)$

not a heap.

minheapify

(don't call it that often)

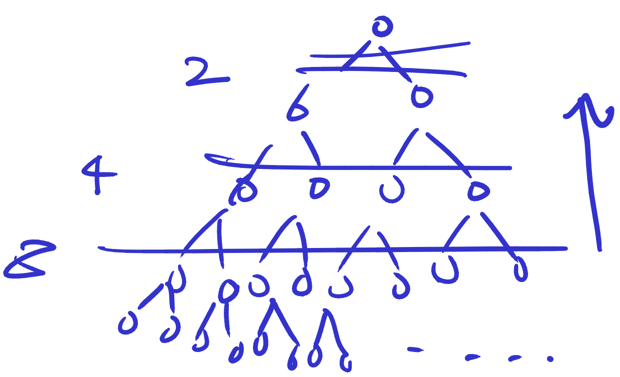
correct it.

assume heap.

$\frac{n}{2}$ minheapify (j)

$O(n/2)$ ⟶ $O(n)$

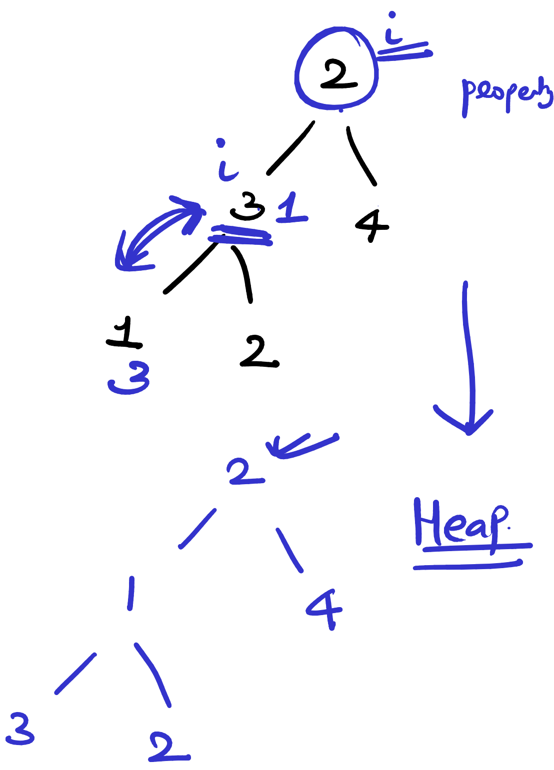
MIT:



Process backwards   5

$j = n/2 - 1$ $j >= 0$ $j--$

minheapify(j)   Ensures



Q. Read   what is there to read?

heap is already an array. [ ] $\xrightarrow{}$ O(n)

You read always the minimum (one by one).

heap: [1 3 2 5 4 7 8 6 9 10]

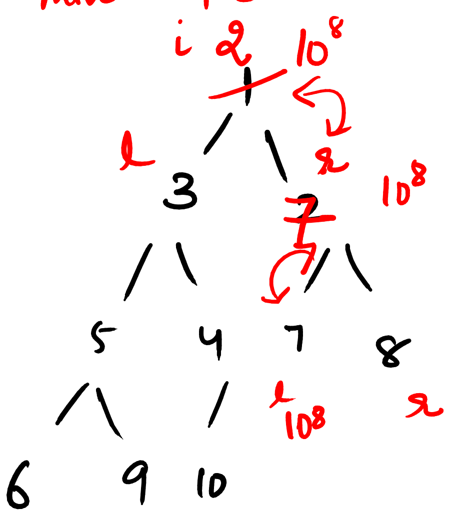
getmin( ) → heap[0]   O(1)

extractmin( ) ⇒ 1

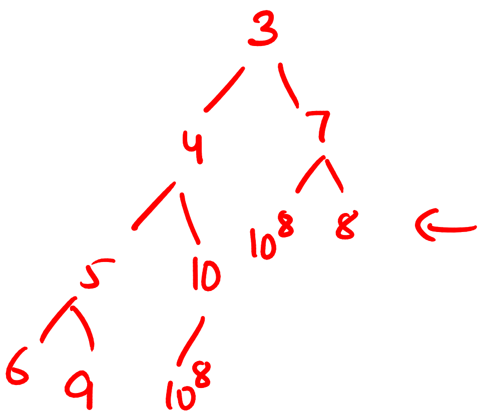
extractmin( ) ⇒ ~~2~~

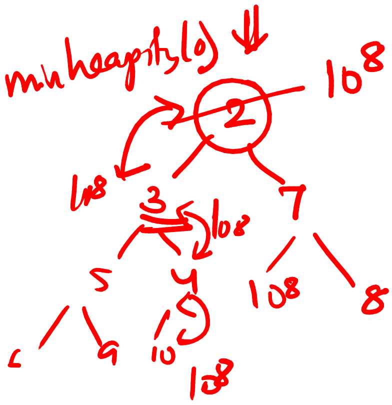
extractmin( ) ⇒ 3

:: ⇒ 4



$: 10^8





[3 4 7 5 10 10^8 8 6 9 10^8]   while (n--)

extractmin( )

Sorted order ⟶ Heapsort

any array → heap → extract min $O(\log n)$

$O(n)$ $\times n$

Total work : $O(n) + \underline{n \times O(\log n)}$

$\underline{O(n)}$ + $O(n\log n)$

$\times$

Heapsort = $O(n\log n)$

If you are doing extractMin( ) n times in sequence → heaps make no sense.

If you sort.

Task : bulk n numbers → heap $O(n)$ heap / array

$O(1)$ extract min( ) → $O(n\log n)$ heap

extract min( )

extractmin( )

$O(n)$ insert(x_1) 5

$O(n)$ insert (x_2) 1

extractMin( )

extractmin( )

insert(x_3)

any order

q

2 ops:

extractmin } any number of times.

insert

array. T.C

[~~1~~ ~~2~~ ~~3~~ 4 5 6 7 8 5]

i i i i

\$ \$ \$ 4 5 5 6 7 8

i

\$ \$ \$ 1 4 5 5 6 7 8

i

T.C $O(n\log n) + q \cdot O(n)$

$q \sim n$ → $O(n\log n + q \cdot n)$ worst case

$O(n^2)$

→ $O(n\log n)$ (sorting) Best case

Heap.

random array → heap $O(n)$

extract min ⇒ $O(\log n)$
insert ⇒ $O(\log n)$
} q

$q \sim n$ $O(n \log n)$

T.C. $O(n + q.\log n)$

AVL

BST vs. Heap

extract min $O(\log n)$

get min → $O(\log n)$
insert(x) → $O(\log n)$
} BST matches.