Heap & its Identification

Identification
- → k
- → smallest / largest } Heap

Types of Heap

MinHeap
k + largest
→ means minimum wala hota hai

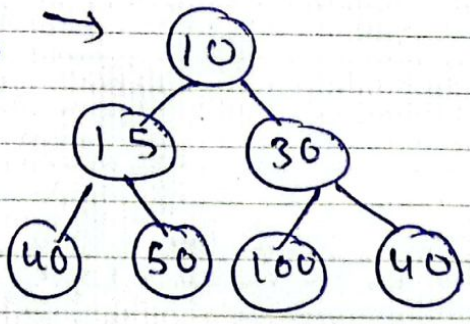
Max Heap
k + smallest
→ means maximum wala hota hai

- Binary Heap is Complet Binary Tree

$$left(i) = 2i + 1$$
$$Right(i) = 2i + 2$$
$$Parent(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$$

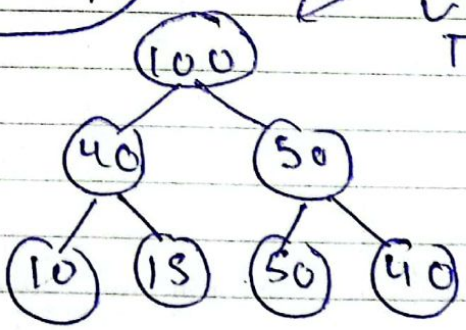
0 based Indexing

minimum value at top →



MIN Heap

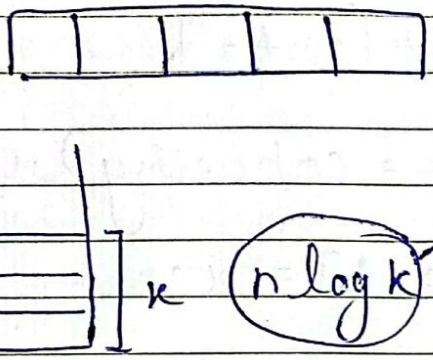
← maximum value at Top.



MAX Heap

sunday | 02

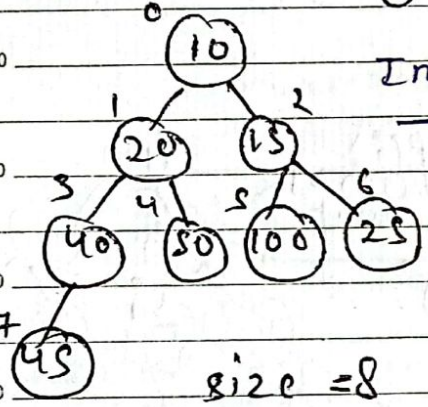
Heap ke Q -> sabse Sorting Q hote hai
↓
(n log n) TC



Heap ke mostly Q banate hai Sorting based hote hai

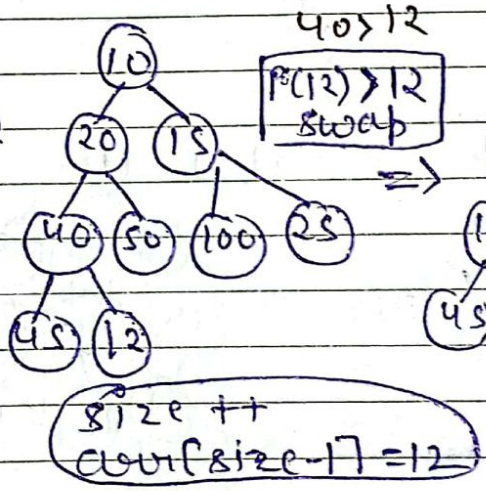
1) Basic solution of Heap Problem is Sorting.

Binary Heap Insert

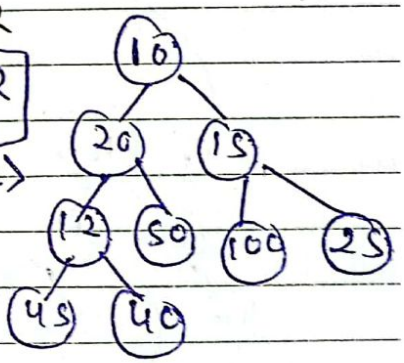


size = 8

Insert → 12

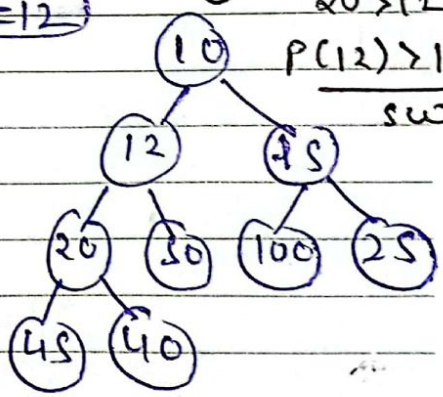


40 > 12
P(12) > 12 swap
=>



size ++
curr[size-1] = 12

20 > 12
P(12) > 12 swap



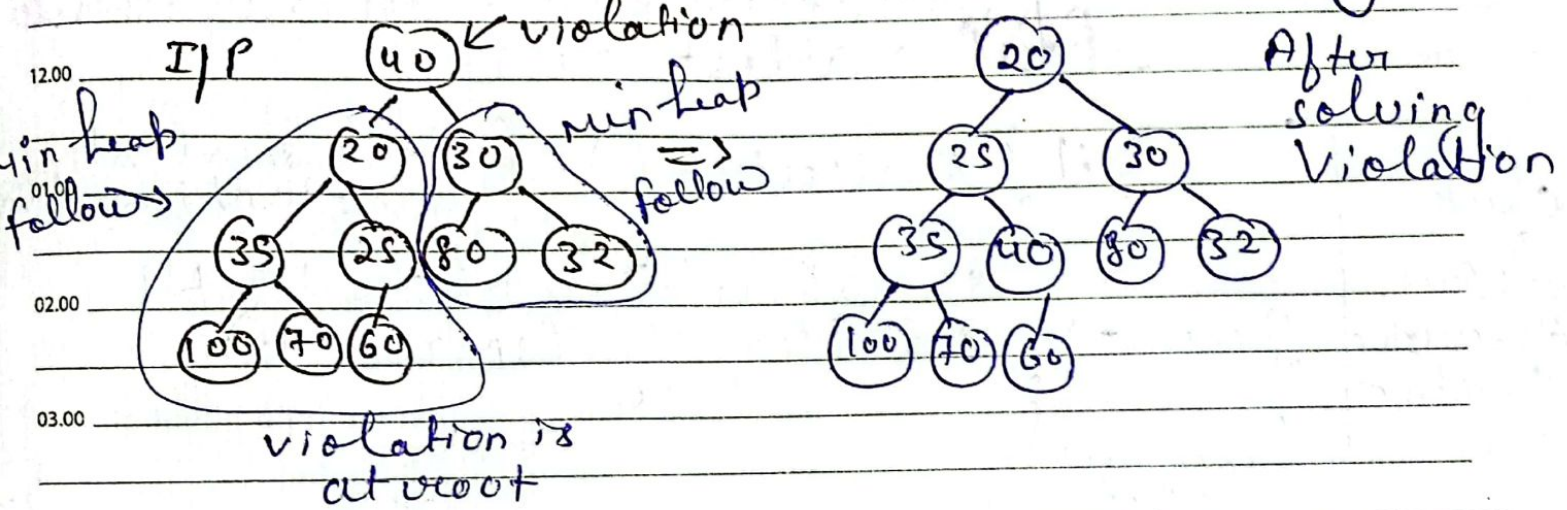
Stop Condition

-> if parent < current node

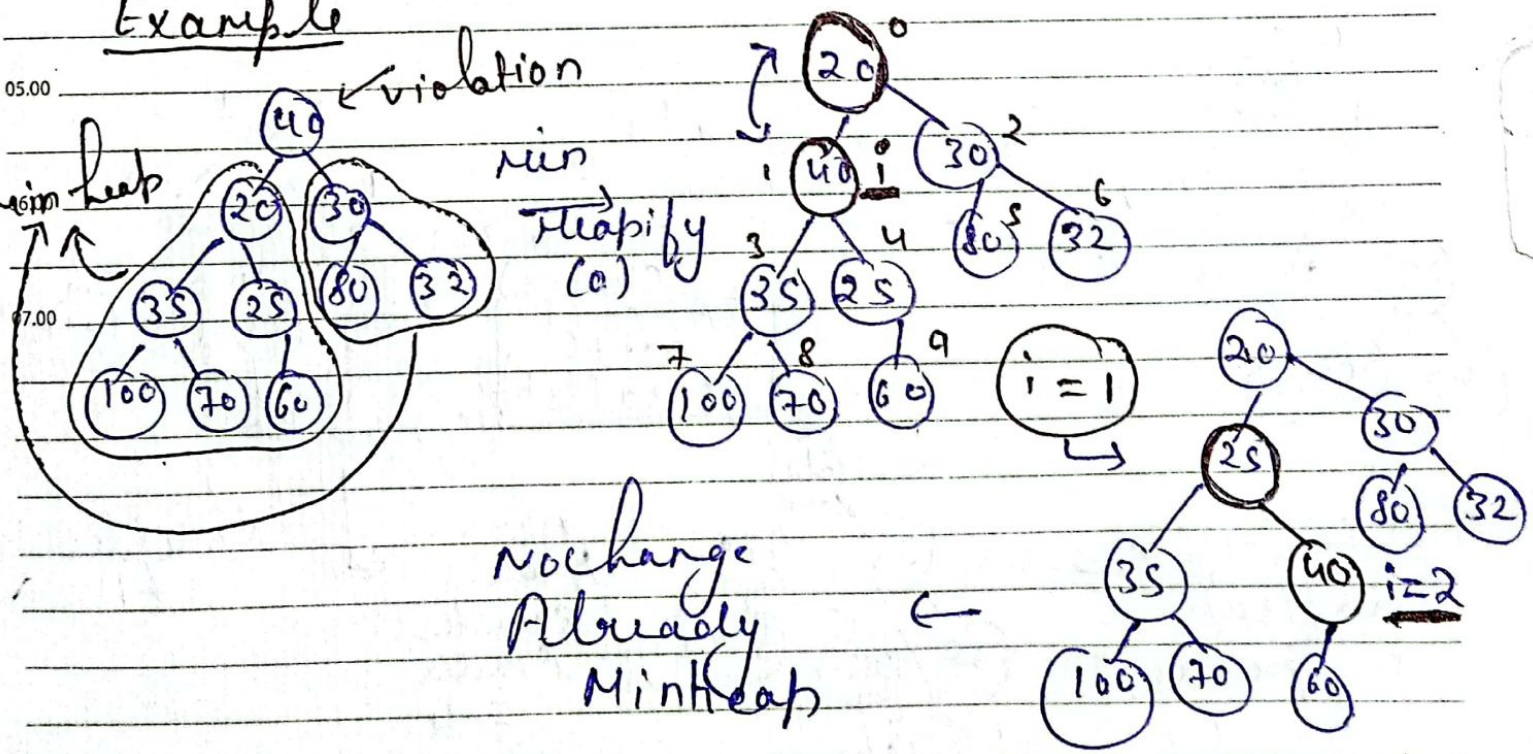
-> current node now at root.

Binary Heap (Heapify)

- Mainly used as subroutine in many standard operation.
- Given Min/Max heap with one possible violation. Fix the Min/Max Binary Heap.

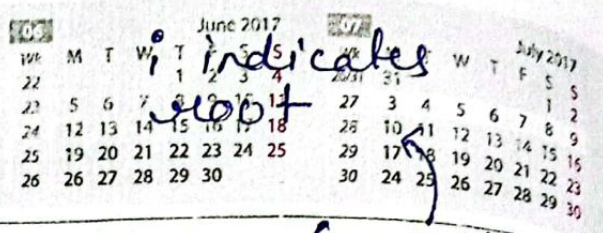


Example



2017-07-05 | 186-179 | 27

05 | 2017 july wednesday



Algo void ~~Heapify~~ MinHeapify(int i)

{

int li = left(i), ri = right(i)

int smallest = i

if(li < size && arr[li] < arr[smallest])
smallest = li

if(ri < size && arr[ri] < arr[smallest])
smallest = ri

find the minimum value →

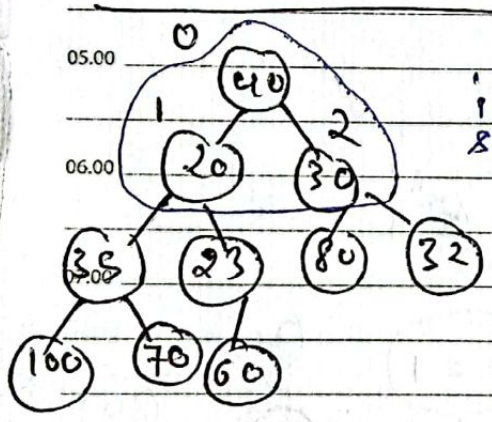
if(smallest != i) (check smallest same to nhi ka index)
{

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

Again → Call with smallest value → minHeapify(smallest)

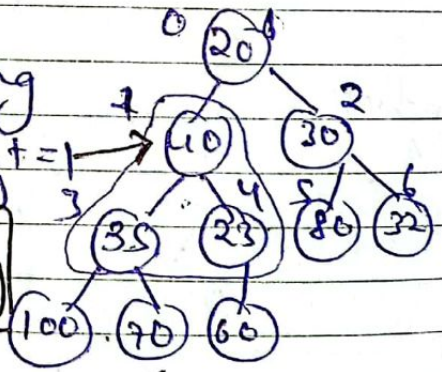
}

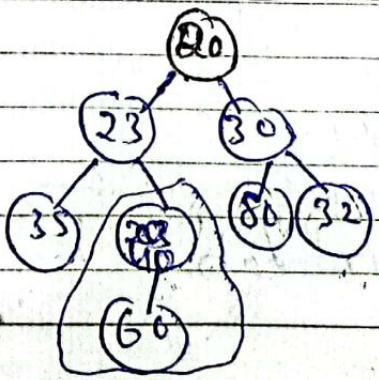
Check & call Recursive Function



i=0
smallest=0

After processing smallest = 1 (index)
(Ab call with smallest)





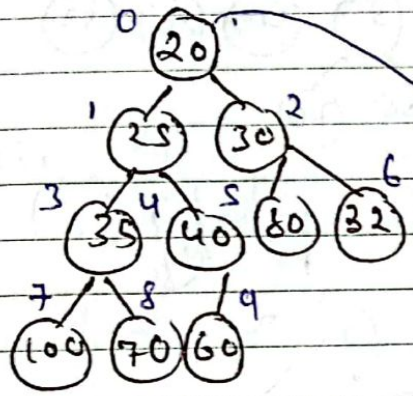
All satisfy ← min Heap property.

After process smallest (index) = 4

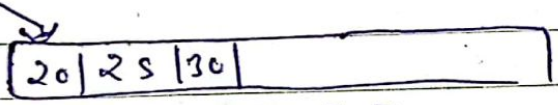
smallest = 1

TC → (log N) → Height of Tree

Binary Heap (Get Min & Extract Min)



getMin - It simply return the minimum element from Min Heap

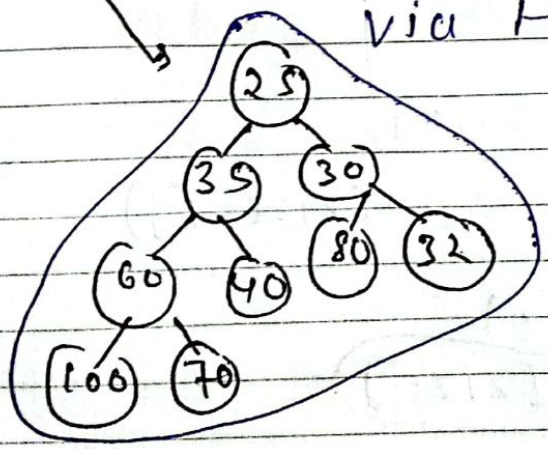


return arr[0]

TC → O(1) (It does not change the Binary Heap)

ExtractMin → remove minimum element from Heap & Return it

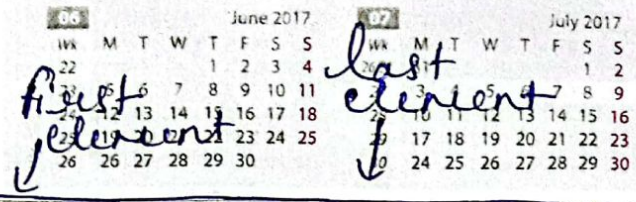
2 things Always remember after remove
→ size--
→ Binary Heap change → so u have to make MinHeap / MaxHeap via Heapify.



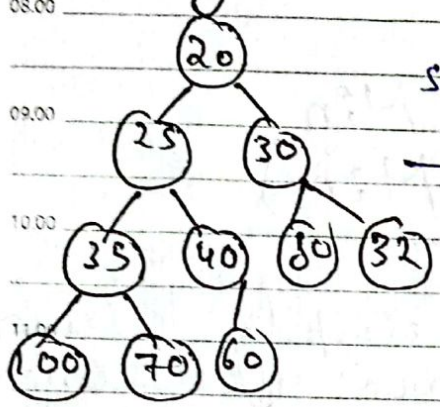
this result show after perform ExtractMin() function.
(It change the Binary Heap)

2017-07-07 | 188-177 | 27

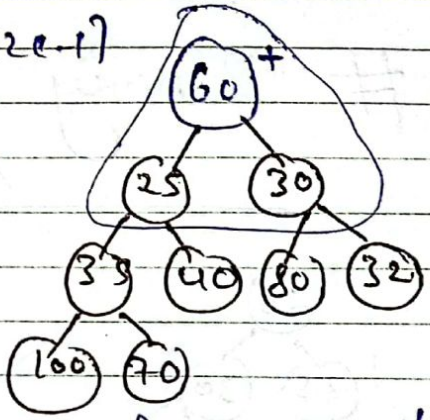
07 | 2017 july friday



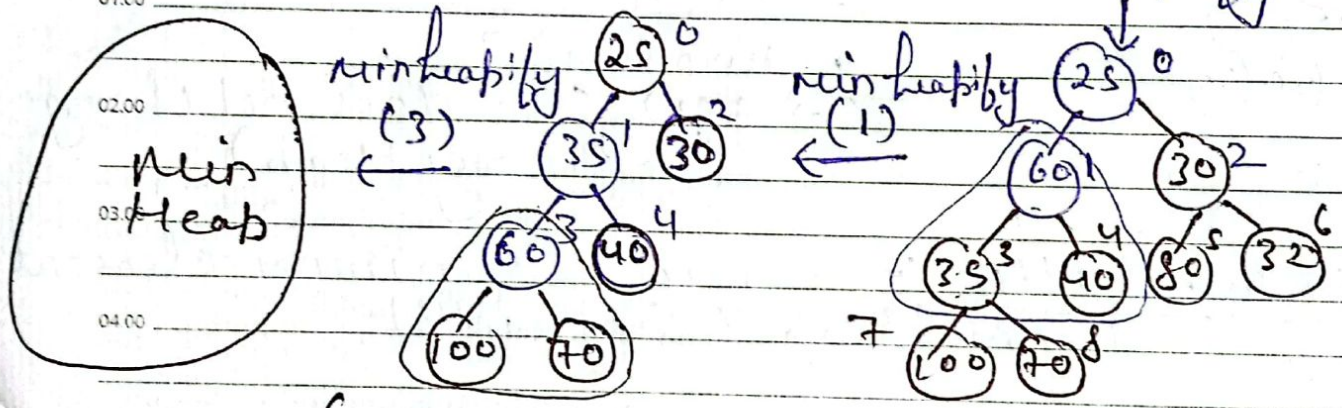
Algo - swap (arr[0], arr[size-1])
size--



swap(arr[0], arr[size-1])
size--



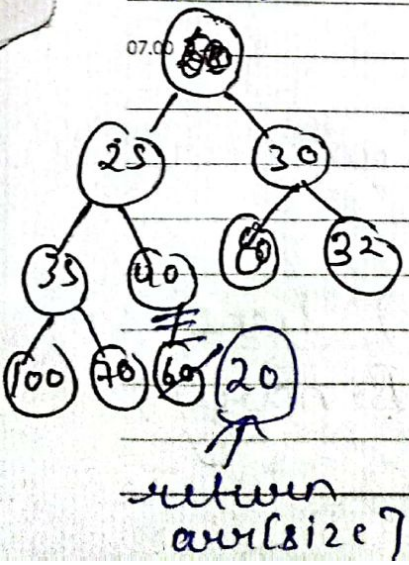
minheapify (0) ← index



Code

```
int extractMin()
{
  if (size==0) return INT_MAX
  if (size==1)
  { size--;
    return arr[0]
  }
  swap(arr[0], arr[size-1])
  size--
  MinHeapify(0)
  return arr[size]
}
```

~~2060~~

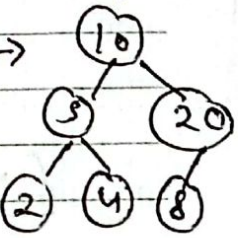


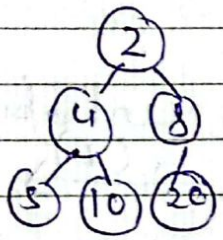
~~return~~ arr[size]

Binary Heap (Build Heap).

Given a random Array, rearrange its elements to form a Min/Max Heap

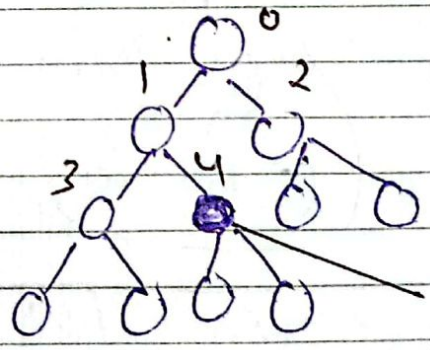
I/p arr[7] = { 10, 5, 20, 2, 4, 8 }
O/P arr[0] = { 2, 4, 8, 5, 10, 20 }





Build Heap → Pick (Bottom most, Right most internal node and do MinHeapify(i)

Find index (i)
then
MinHeapify(i)
MinHeapify(i-1)
MinHeapify(i-2)
⋮
i
MinHeapify(0)



from there we start heapify then for 3, 2, 1, 0

lastNode → (size-1)
lastNode Parent → $\frac{(size-1)-1}{2}$

2017-07-11 | 192-173 | 28

11 | 2017 july tuesday

06	June 2017						
Wk	M	T	W	T	F	S	S
22				1	2	3	4
23	5	6	7	8	9	10	11
24	12	13	14	15	16	17	18
25	19	20	21	22	23	24	25
26	26	27	28	29	30		

07	July 2017						
Wk	M	T	W	T	F	S	S
26/31	31					1	2
27	3	4	5	6	7	8	9
28	10	11	12	13	14	15	16
29	17	18	19	20	21	22	23
30	24	25	26	27	28	29	30

Start from last Node parent

08.00

09.00

10.00

11.00

12.00

01.00

02.00

03.00

04.00

05.00

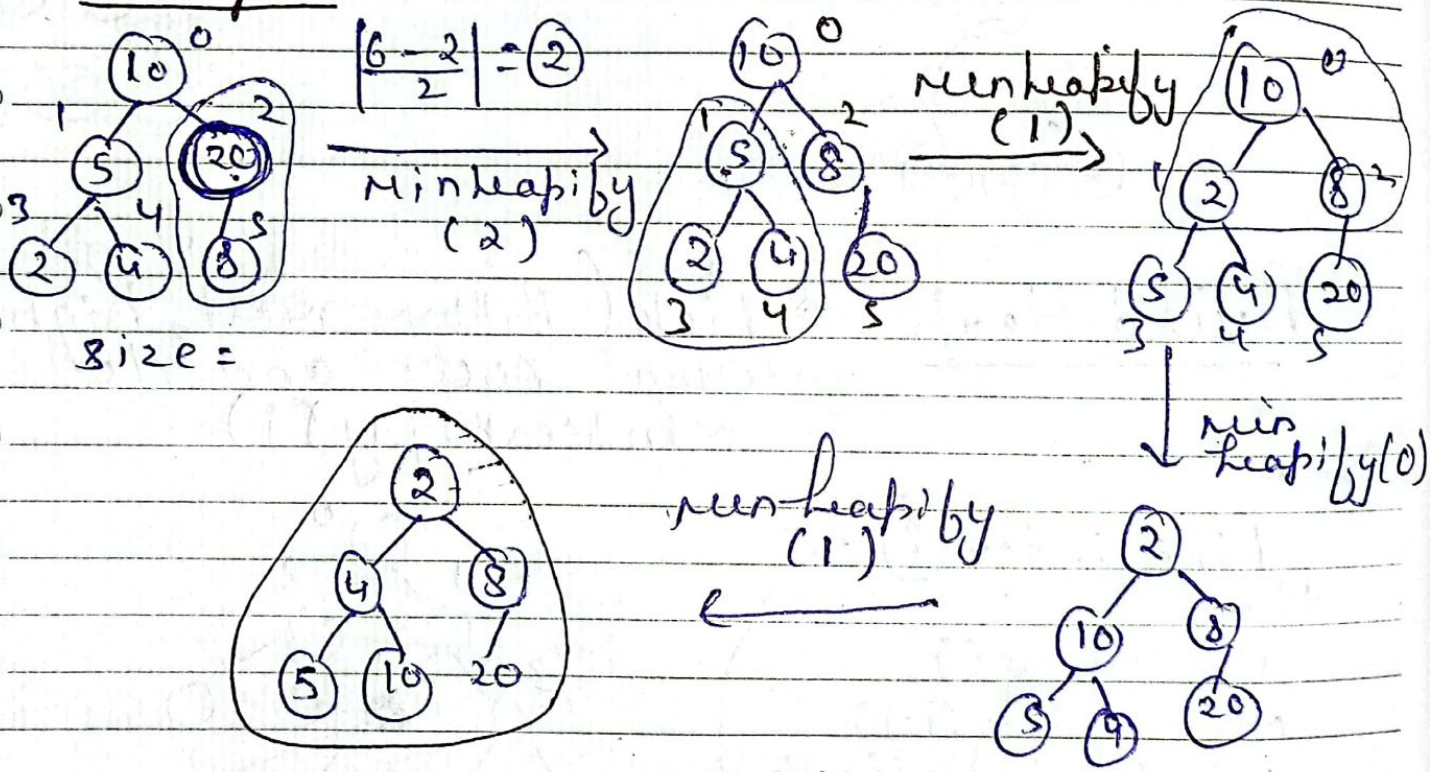
06.00

07.00

Algo

```
void buildHeap()
{
    for(int i = |size-2 / 2| ; i>=0 ; i--)
    {
        minheapify(i)
    }
}
```

Example



Time Complexity → O(N)

Heap Sort

- It is based on selection sort Algo

Find out maximum element in group & swap with last Element

In selection Sort → we do linear search to find maximum element.
TC → $O(N^2)$

In Heap Sort → we reconstruct the given Array and form max Heap.
Means, root se maximum value uthay array ke last me daal dia.
TC → $O(\log N)$ → maximum height of tree.

Code

```
void buildHeap(int arr[], int n)
{
  for(int i = (n-2)/2 ; i>=0; i--)
  {
    maxheapify(arr, n, i)
  }
}
```

↑ his index pe karna hai

(2) void Maxheapify (int arr[], int n, int ind)

{

int largest = ind[index]

int left = 2i + 1

int right = 2i + 2

Find largest

if (left < n && arr[left] > arr[largest]

largest = left;

if (right < n && arr[right] > arr[largest]

largest = right

here do swap & call heapify fun

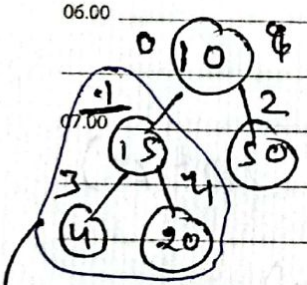
if (largest != i)

{

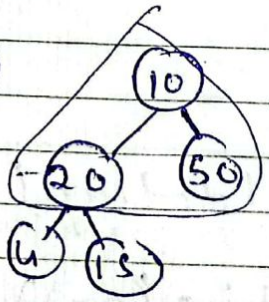
swap (arr[largest], arr[i]

maxheapify (arr, n, target)

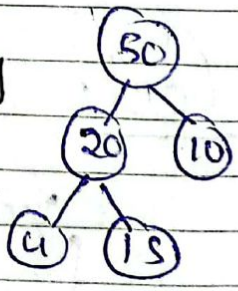
}}



heapify(1) →



heapify (0) →



| 10 | 15 | 50 | 4 | 20 |

| 10 | 20 | 50 | 4 | 15 |

| 50 | 20 | 10 | 4 | 15 |

maxheap hogye. then Apply Heap sort

```
void HeapSort(int arr[], int n)
{
    buildHeap(arr, n)
    for(int i=n-1; i>=1; i--)
    {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0)
    }
}
```

size, index

work on visualise on Array

- Repeatedly swap root with last Element
- Reduce Heap size by 1 and do Heapify().

heapify always from 0 index (Root) i esize) always decrease by 1

Time Complexity - O(N log N)

15

2017
july
saturday