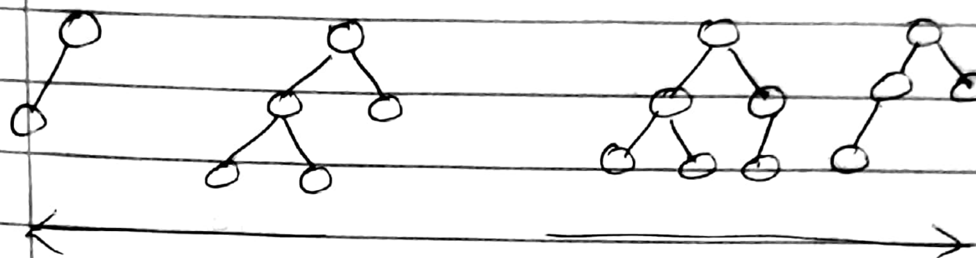
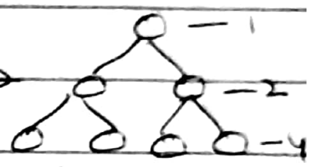
Heap:

- Heap is an almost complete binary tree.
- An almost complete binary tree is a tree in which every level except possibly the last level is completely filled ~~and~~ or ~~add~~ left nodes are present.



almost complete binary tree

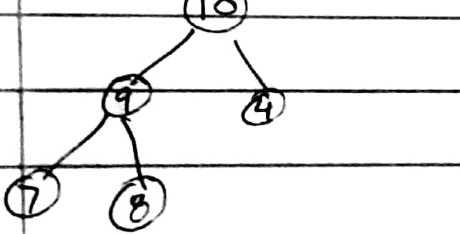


Complete binary tree

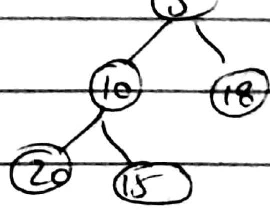
Types

Max heap | Min heap

In max heap the value of root node (Parent) is greater than value of its child nodes.

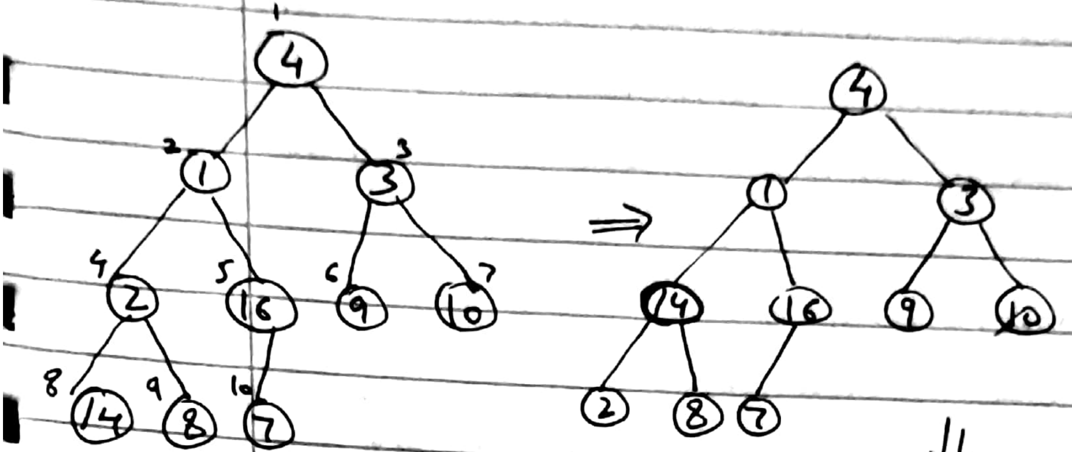


In min heap the value of root node is less than value of its child nodes.

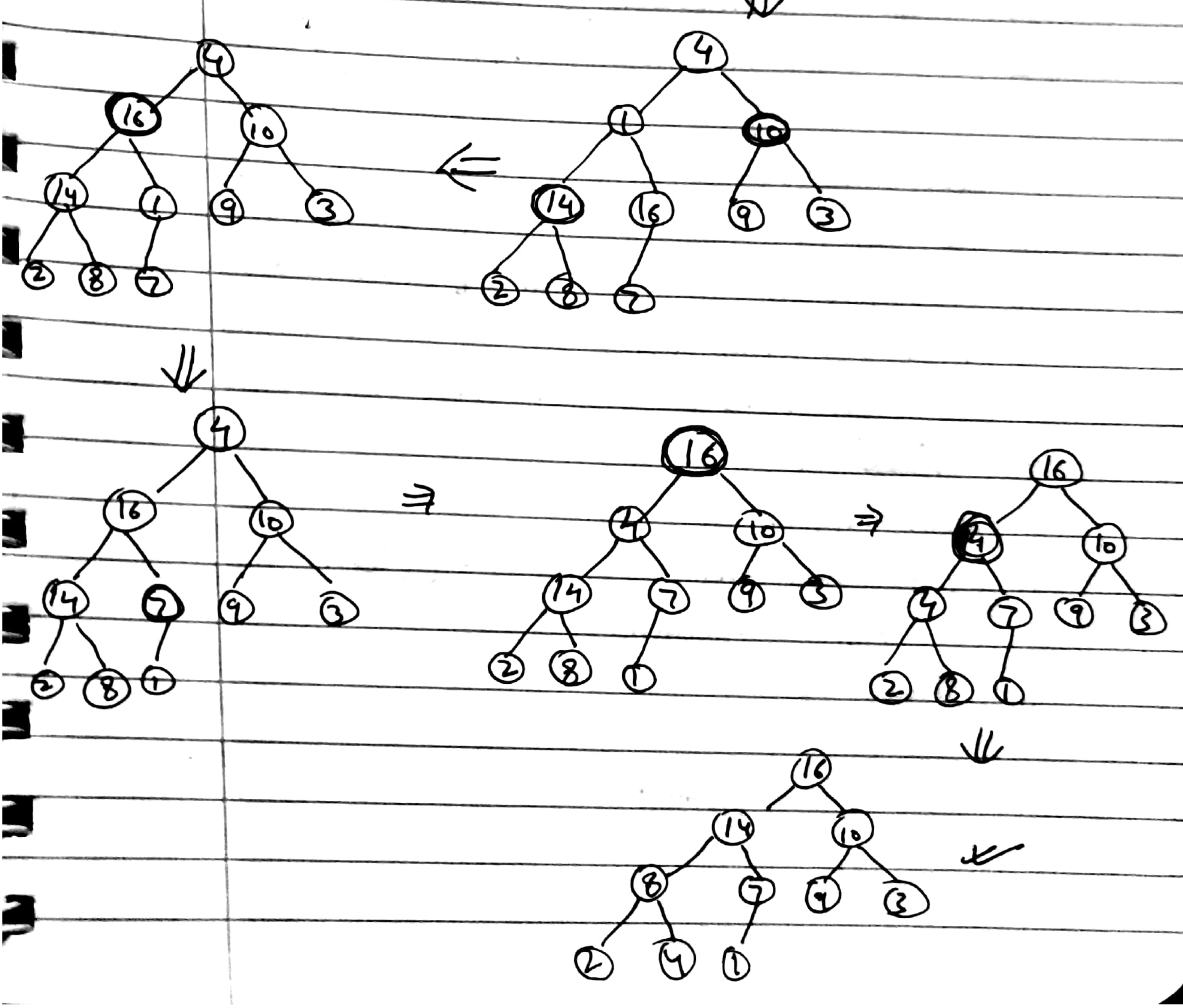


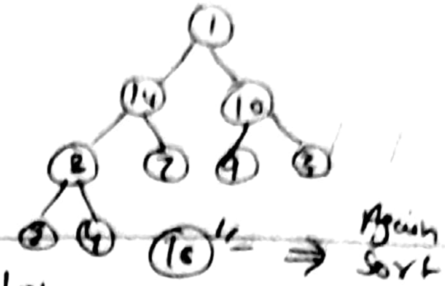
Heap Sort

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7



length/2 → Loop start





Heap Sort (A)

1) Build - Max heap (A) — n logn
2) for i = A.length down to 2 — n
{
3) exchange A[1] with A[i] — n
4) A.heapsize = A.heapsize - 1
5) Max - heapify (A, 1) — logn — n logn
}

Build - Max heap (A, i) — O(n logn)

1) A.heapsize = A.length
2) for j = ⌊A.length/2⌋ down to 1 — n/2
3) Max - heapify (A, i) — $\frac{n}{2} \cdot$ logn = n logn

Max - heapify (A, i)

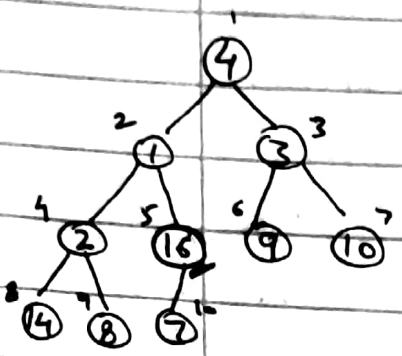
1) l = left(i)
2) r = Right (i)
3) if l ≤ A.length and A[l] > A[i]
4) largest = l
5) else largest = i
6) if r ≤ A.length and A[r] > A[largest] — O(logn)
7) largest = r
8) if largest ≠ i
9) { exchange A[i] with A[largest]
10) Max - heapify (A, largest)
}

O(nlogn)

1	2	3	4	7	8	9	10	14	16

⇒

4	1	3	2	16	9	10	14	8	7



Build Max-heap

1) A. heapsize = 10 ⇒ A. length

2) $\frac{10}{2}$ to 1 ⇒ 5 to 1

3) Max heapify (A, 5) i=5

1) l = 10 , r = —

3) l ≤ 10 and 7 > 16 false

5) largest = i ⇒ 5

Skip 6 & 7 becuz no right value

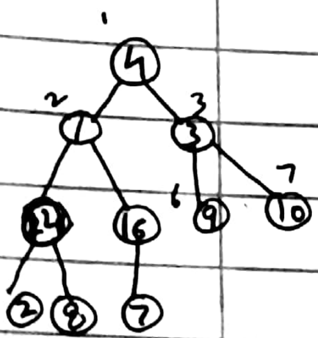
8) largest ≠ i false

~~Max heapify~~

i = 4

Max heapify (A, 4)

l = 8 , r = 9



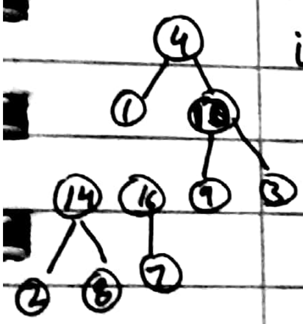
3) l ≤ 10 and ~~AE~~ 14 > 2 True

largest = l ⇒ largest = 8

6) r ≤ 10 and 8 > 14 false

8) ~~lar~~ 8 ≠ 4 ~~fa~~ True

enchange 2 & 14



i=3

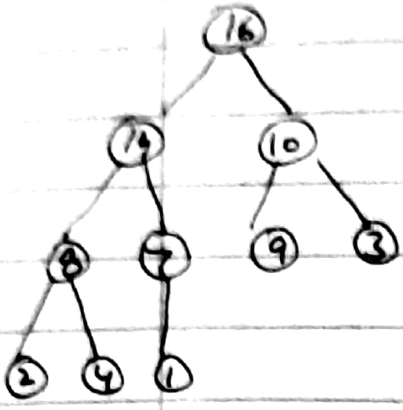
l = 6, r = 7

3) l ≤ 10 and 9 > 3 True

largest = l ⇒ largest = 6

6) r ≤ 10 and 10 > 9 True

largest = r ⇒ largest = 7



Build Max heap

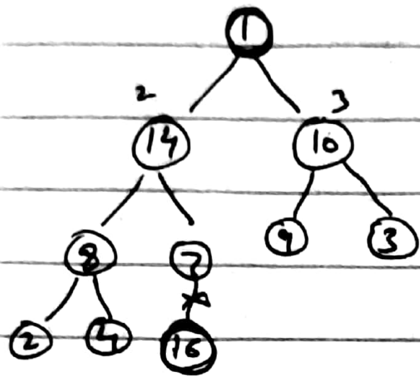
$i = 10$ to 2

exchange $A[1]$ & $A[10]$

$A.heapSize = A.heapSize - 1$

Maxheapify $(A,1)$

⇒ Again find second largest root element



Discard 16

Max-heapify $(A,1)$

$l = 2$, $r = 3$

$l \le A.length$ and $A[2] > A[1]$ True

largest $= l \Rightarrow$ largest $= 14$

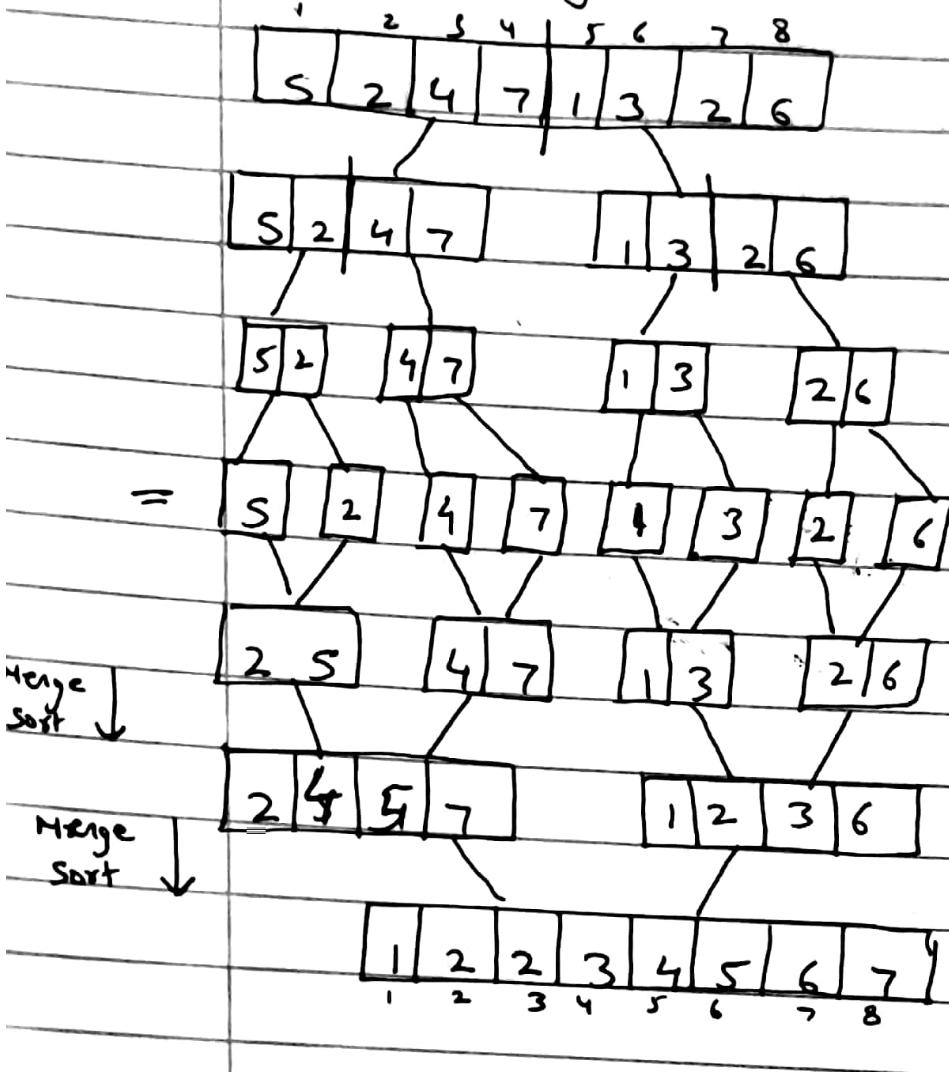
$r \le A.length$ and $A[r] > A[2]$ false

largest $\ne i$ true

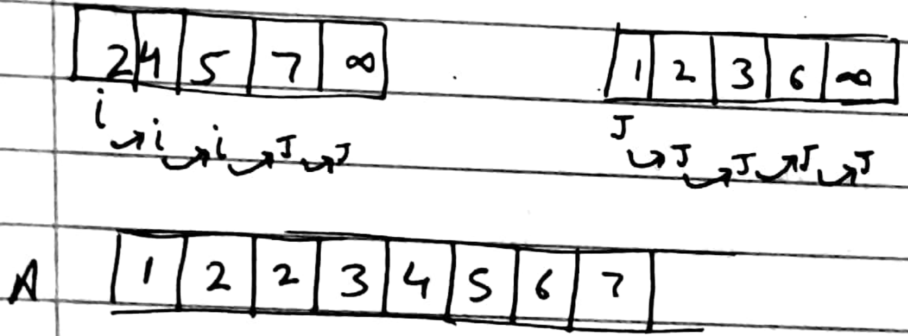
Max-heapify $(A, largest)$

Merge Sort

Merge Sort closely follow the divide and Conquer Paradigm.



$(1+8)/2 = \lfloor 4.5 \rfloor = 4$



```
MERG-SORT (A, P, r)                          ——  T(n)
if P < r
   then q ← ⌊(P+r)/2⌋
        MERG-SORT (A, P, q)      —— T (n/2)
        MERG-SORT (A, q+1, r)    —— T (n/2)
        MERGE (A, P, q, r)       —— O (n)
```

```
MERG (A P q r)
n1 ← q - P + 1
n2 ← r - q
Create array L[1....n1+1] & R[1....n2+1]
for i ← 1 to n1
   L[i] ← A[P+i-1]
for J ← 1 to n2
   R[J] ← A [q+J]
L [n1+1] ← ∞
R [n2+1] ← ∞                                  O(n)
i ← 1
J ← 1                                        T(n) = 2T(n/2) + n
for K ← P to r
      if L[i] ≤ R[J]
      then A[K] ← L[i]
           i ← i+1
      else A[K] ← R[J]
           J ← J+1
```

5	2	4	7	1	3	2	6

$p = 1$
$r = 8$
$q = 4$

2	4	5	7

1	2	3	6

$n_1 = 4 - 1 + 1$
$= 4$
$n_2 = 8 - 4 = 4$

$n_1 = 4$
$n_2 = 4$

L

1	2	3	4	5
2	4	5	7	∞

i → i

R J

1	2	3	4	5
1	2	3	6	∞

for K ← 1 to 8

K=1 L[1] ≤ R[1] false i=1 J=2

A[K] ← R[1]

J → J+1

1	2	2	3	4	5	6	7

K=2 L[1] ≤ R[2]

2 ≤ 2 True

A[K] ← L[i] i=2, J=2

i ← i+1

K=3 L[2] ≤ R[2] false

A[K] ← R[2] J=3, i=2

J → J+1

K=4 L[2] ≤ R[3] false

A[K] ← R[3] J=4, i=2

J → J+1

K=5 L[2] ≤ R[4] True

A[K] ← L[i] J=4, i=3

i=i+1

$L[3] \leq R[4]$ True

$A[k] \leftarrow A[i]$

$i = i+1$ $\quad$ $i = 4, J = 4$

$L[4] \leq K[4]$ False

$A[k] \leftarrow A[J]$

$J \rightarrow J+1$ $\quad$ $i = 4, J = 5$

$L[4] \leq R[5]$ True $\quad$ $p \rightarrow 1$

$r \rightarrow 8$

$A[k] \leftarrow A[i]$ $\quad$ $i = 5$.

$i = i+1$