⇒ Bubble Sort

Swap largest integer to right after iterations.

```
for (i = 0; i < n-1; i++)
 {
    for (j = 0; j < n-i-1; j++)
     {
       if (A[j] > A[j+1])
        {
           temp = A[j];
           A[j] = A[j+1];
           A[j+1] = temp;
        }
     }
 }
```

→ Space complexity : $\Theta(1)$ (3 variables)

→ Time Complexity : $\Theta(n^2)$

→ On place sorting

→ Stable Algo (Relative order of equal elements is maintained)

→ Bubble sort optimization

```
flag = 1
for (i = 0; i < n-1; i++)
{
    for (j = 0;  j < n-i-1; j++)
    {
        if (A[j] > A[j+1])
        {
            swap(A[j], A[j+1])
            flag = 0;
        }
    }
    if (flag = = 1)
    {
        break;
    }
    flag = 1;
}
```

⇒ Selection sort (Min swapping among all algo)

(One swapping per iterations)

Put min element at the start, start element at the min-element's index.

→
```
for (i=0; i<n-1; i++)
{
    min = i;
    for( j = i+1; j<n-1; j++)
    {
        if (A[j] < A[min])
            min = j;
    }
    temp = A[i];
    A[i] = A[min];
    A[min] = temp;
}
```

→ Runtime Complexity : $O(n^2)$

→ Space complexity : $O(1)$

→ Max. swap operations : $n-1$

→ In-place : Yes

→ Stable : No

⇒ Insertion Sort

Sort array by inserting elements from 1st position to ~~at~~ whole array

```
for (i=1; i<=n-1 ; i++)
{
  temp = A[i];
  for (j=i-1; j >=0 && A[j]>temp; j--)
  {
    A[j+1] = A[j];
  }
  A[j+1] = temp;
}
```

→ Time Complexity : $O(n^2)$

Best case = $O(n^2)$

→ In-place : Yes

→ Stable : Yes

→ Space Complexity : $O(1)$

3 5 4 6

3 5 4 6

3 5 4 6

⇒ Heap Sort

We sort array using binary heap.

Complete B.T. → next level only if prev. level is full

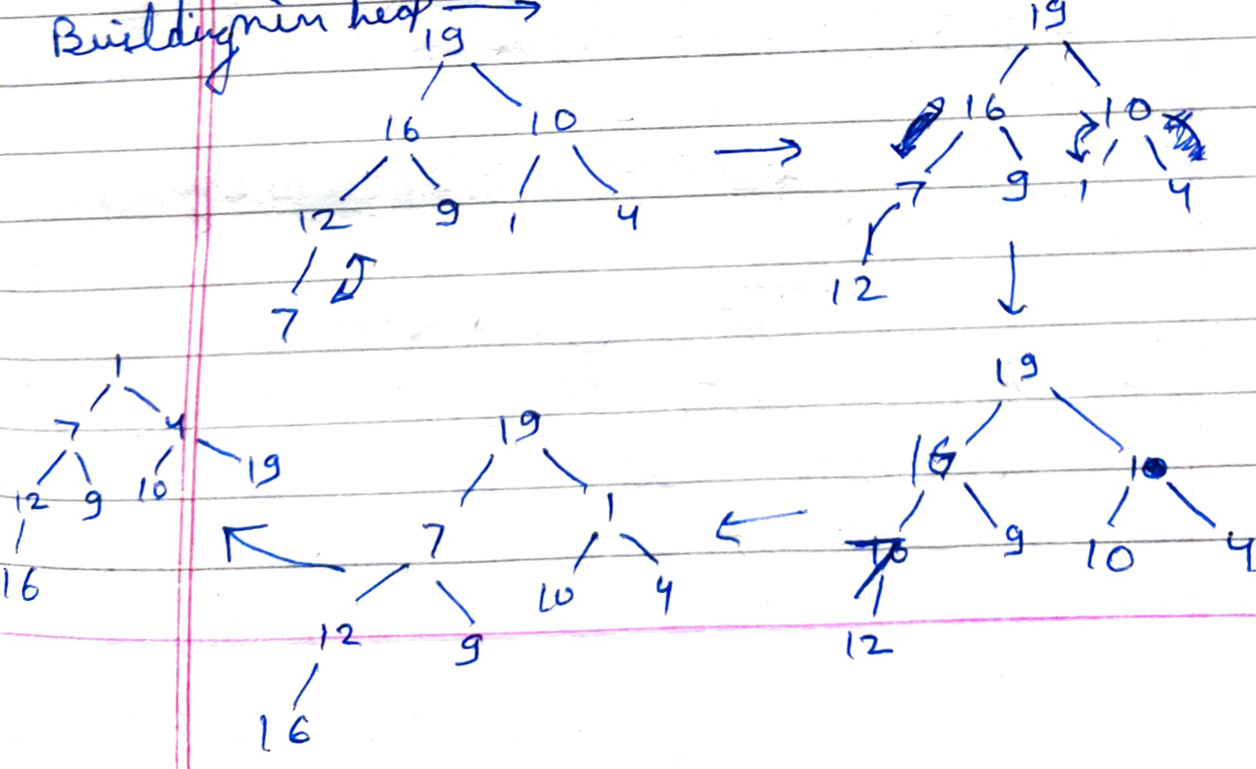
→ Min heap (parent is smaller)

→ Max heap (parent is ~~st~~ larger)

→ Building heap

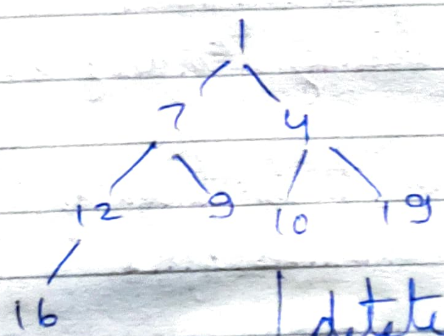
- → make complete B. tree
- → Adjust values to create heap
  - ↓ start from lowest internal node

Building min heap →

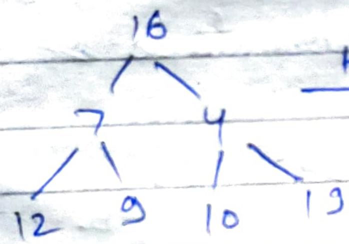


→ deletion in heap [Root node is deleted]

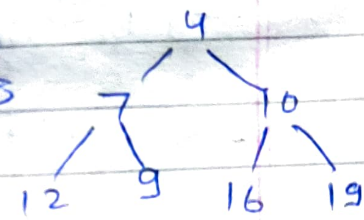
↓ last node sent to root



delete



Heapify



Heap sort → ① Build Heap
② Delete root one by one and store in array

→ Building heap => $O(n)$

→ one by one deletion => $O(n \log n)$

∴ Total $O(n \log n)$

→ Space complexity = $O(n)$

→ In-place = No

→ Stable = No

⇒ Merge Sort

- → Divide list into 2 parts
- → Sort both parts
- → Merge them

Space Complexity :- $O(n)$

Time Complexity :- $O(n \log n)$

In-place - No

Stable - Yes

⇒ Quick Sort

Partition Algorithm

pivot

Ex   6   9   2   3   5   8   1

i →   ← j

>6 → stop

↓ swap

6   1   2   3   5   8   9

→ i

j ←

↳ When ~~i>j~~ i > j , swap pivot element with element on index j

→ Quicksort (n, start, end)
{
k = partition (n, start, end)
Quicksort (k, start, k-1)
Quicksort (n-k-1, k+1, end)
}

Ex

i j

(6) 8 3 1 4 5 7

i→ ←j

↓

(6) 5 3 1 4 8 7

i j

i j

i j

(4) 5 3 1 6 8 7

i j

↓

(4) 1 3 5 6 8 7

i i j

j j i

(3) 1 4 5 6 (8) 7

i j i j

$\boxed{1 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8}$

Best
→ Runtime Complexity – $O(n \log n)$

→ Worst → $O(n^2)$

→ avg. $O(n \log n)$

→ Space Complexity $O(1)$

→ In-place → yes

→ Stable → no

⇒ <u>Time Complexity</u>

O — Upper bound

If $O_{f(n)} < O_{g(n)}$

↳ $f(n) = O(g(n))$

growth rate of $g(n)$ is greater or equal to $f(n)$

Ω — lower bound

$f(n) = \Omega(g(n))$

$f(n) > g(n)$

↳ growth rate of $g(n)$ is smaller or equal to $f(n)$

⇒ If multiplication of growth changes the growth, then the growth is same.

Θ

$f(n) = \Theta(g(n))$

↳ growth rate of both functions are equal

⇒ Small 'O' (o)

$f(n) = o(g(n))$

↳ growth of $g(n)$ should be greater than $f(n)$.