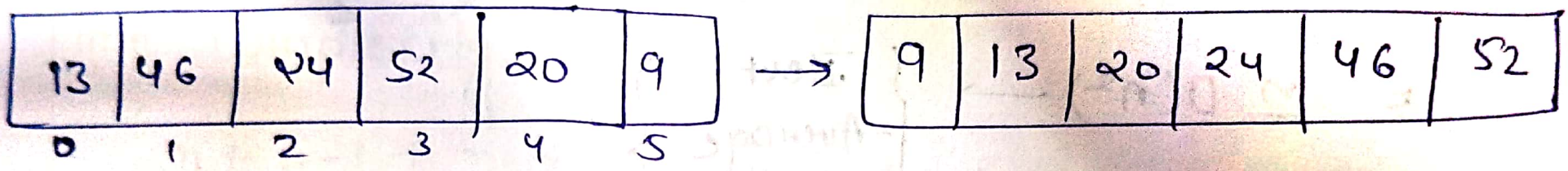
[Sorting]

1) Selection sort :- Select minimum & Swap



9	46	24	52	20	13	SI
9	13	24	52	20	46	S2
9	13	20	52	24	46	S3
9	13	20	24	52	46	S4
9	13	20	24	46	52	S5

⇒ Swap at index 0 & min index {0, n-1}
Swap at index 1 & min index {1, n-1}

| | |
n-2

⇒ Pseudo code :-

```
for(i=0 ; i<=n-2 ; i++)
  {  min=i
     for(j=i ; j<=n-1 ; j++)
       { if(arr[j] <arr[min])
           min=j;
       }
     swap(arr[j], arr[i]);
  }
```

⇒ Time Complexity :-

$$n+n-1+n-2+n-3+\cdots\cdots+2+1 \simeq \frac{n\times(n+1)}{2}$$

$$= \frac{n^2}{2}+\frac{n}{2}$$

⇒ $\simeq O(n^2) \longrightarrow$ { Best, Average, Worst }

8) Bubble sort :- push the max to the last by adjacent swaps

* 9, 13, 20, 24, 46, 52 → sorted order

⇒ 13, 46, 24, 52, 20, 9 → unsorted order

Sol:-

S1:- 13, 24, 52, 20, 9, 46 → {0, n-1}

S2:- 13, 24, 20, 9, 46, 52 → {0, n-2}

S3:- 13, 20, 9, 24, 46, 52 → {0, n-3}

S4:- 13, 9, 20, 24, 46, 52 → {0, n-4}

S5:- 9, 13, 20, 24, 46, 52 → {0, n-5} - - - {0, 1}

⇒ Pseudo code :-

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

optimization –

→ int didswap=0

→ didswap=1

→ if(didswap==0)
break;

⇒ Time Complexity :-

$n + n-1 + n-2 + \cdots 2+1$

$= \frac{n(n+1)}{2}$

$= O(n^2)$ → worst or Average

$= O(n)$ → Best (by optimization)

3) Insertion Sort :- Takes an element & Place it its correct order.

⇒ (14) 9, 15, 12, 6, 8, 13

S1:- 9, 14, 15, (12), 6, 8, 13

S2:- 9, 12, 14, 15, (6) 8, 13

S3:- 6, 9, 12, 14, 15, (8), 13

S4:- 6, 8, 9, 12, 14, 15, (13)

S5:- 6, 8, 9, 12, 13, 14, 15

⇒ Pseudo code :-

```
for (i=0; i<=n-1; i++)
  { j=i
    while (j>0 && a[j-1] > a[j]) {
        swap(a[j-1], a[j])
        j--;
    }
}
```

⇒ Time Complexity :-

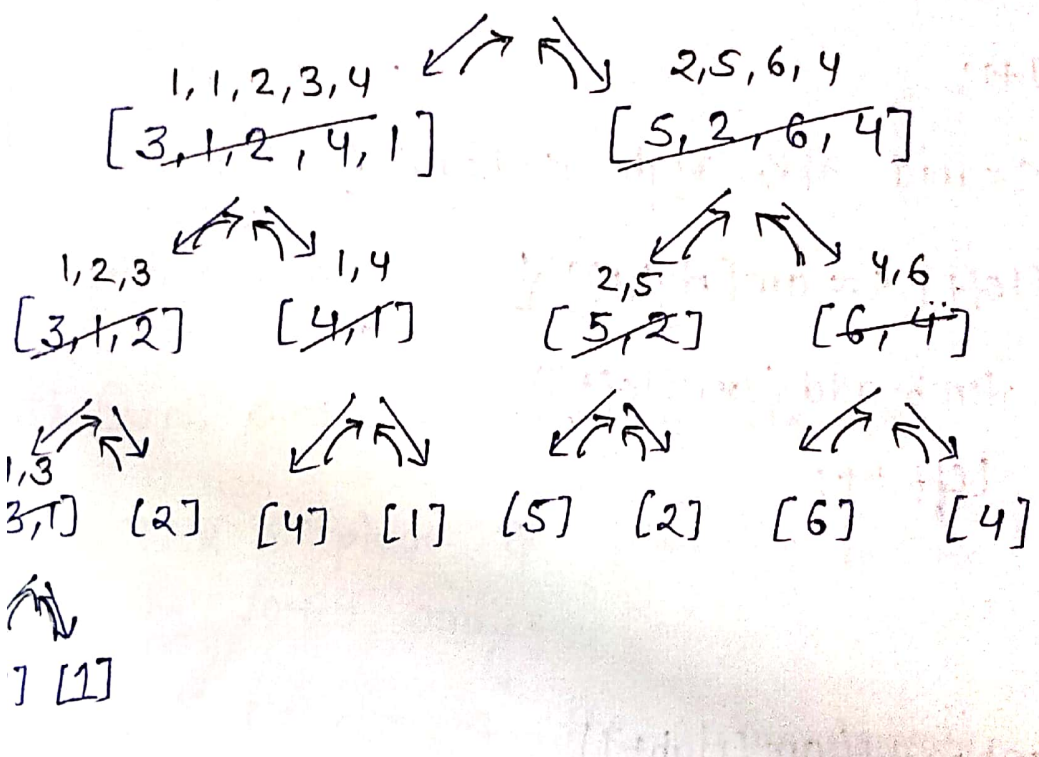
≃ $\frac{n(n+1)}{2}$ ≃ $O(n^2)$ (Worst or average)

≃ $O(n)$ (Best)

4) Merge Sort :- Divide & Merge

1, 1, 2, 2, 3, 4, 4, 5, 6

⇒ [3, 1, 2, 4, 1, 5, 2, 6, 4]



⇒ Pseudo code :-

```
mergeSort(arr, low, high) {
    if(low >= high) return;
    mid = (low + high) / 2;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid+1, high);
    merge(arr, low, mid, high);
}
```

⇒ merge (arr, low, mid, high) {

temp → [];

left = low;

right = mid+1;

while (left <= mid && right <= high) {

if (arr[left] <= arr[right]) {

temp.add (arr[left])

left++;

}

else {

temp.add(arr[right])

right++;

}

}

while (left <= mid) {

temp.add (arr[left]); } if on left

left++;

}

while (right <= high) {

temp.add (arr[~~high~~ right]); } if on right

right ++;

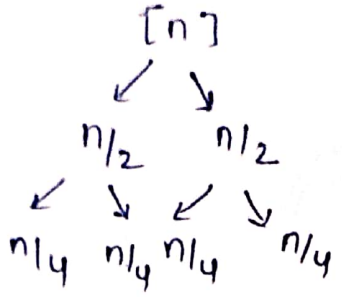
}

for (i = low → high) {

arr[i] = temp[i - low];

}

Time Complexity :-



$\simeq O(\log_2 n)$

So, $\simeq O(N \times \log_2 n)$

while → merge() $\simeq O(N)$

5) Quick sort :- Divide & Conquer

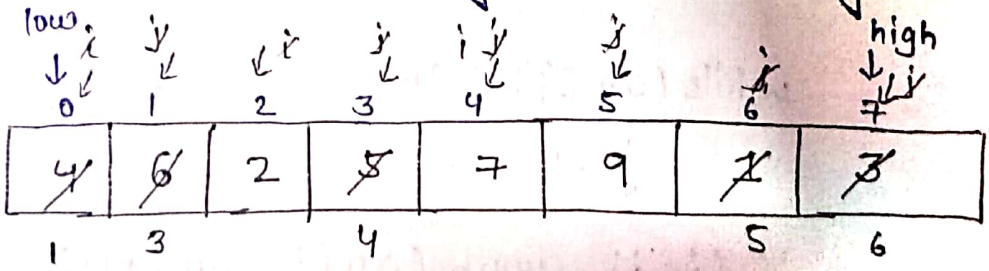
1) Pick a pivot & place it in its correct place in the sorted array.

a. 1st element in the array ~~(pivot)~~
b. ~~last~~ element of array
c. median of the array
d. random element of the array

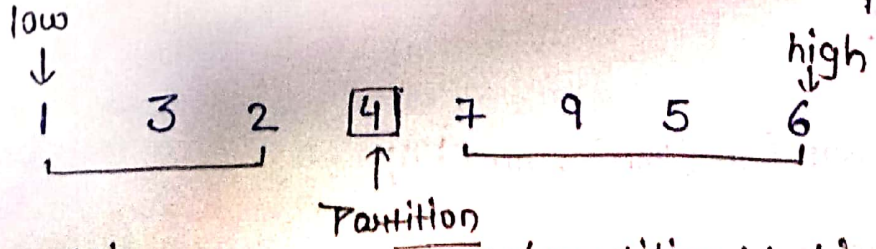
} (Pivot)

2) smaller on the left larger on the right

⇒



Pivot = a[low]



qs(low, partition−1)

qs(partition+1, high)

⇒ Pseudo code :-

```
qs (arr, low, high) {
    if(low < high) {
        PIndex = f(arr, low, high);
        qs (arr, low, PIndex-1);
        qs(arr, PIndex+1, high);
    }
}

int f(arr, low, high) {
    pivot = a[low];
    i = low;
    j = high;
    while (i<j) {
        while (arr[i] <= arr[pivot] && i<=high)
            i++;
        while (arr[j] > arr[pivot] && j>=low)
            j--;
        if (i<j) swap (arr[i], arr[j]);
    }
    swap (arr[low], arr[j]);
    return j;
}
```