# 01 Sorting

## Selection sort

| 13 | 46 | 24 | 52 | 20 | 9 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

9 13 20 24 46 52

[Select minimum and swap]

get min from array

S-1 | 9 | 46 | 24 | 52 | 20 | 13 | → swap at index 0, & min index [0 – n-1].

S-2 | 9 | 13 | 24 | 52 | 20 | 46 | → swap at index 1, & min index [1 – n-1]

S-3 | 9 | 13 | 20 | 52 | 24 | 46 | → swap at index 2, & min index [2 – n-1].

S-4 | 9 | 13 | 20 | 24 | 52 | 46 |

S-5 | 9 | 13 | 20 | 24 | 46 | 52 |

⋮

n-2

- one element always be sorted.
- get the min and swap it.

```
swap
arr[i]      arr[mini]
 10          12
temp = arr[mini]
arr[mini] = arr[i]
arr[i] = temp.
```

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

$TC = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$ so, $TC = O(n^2)$

Best
average
Worst

02 **Bubble sort** [Push the max no to the last by adjacent swaps]

Eg. | 13 | 46 | 24 | 52 | 20 | 9 |

Now, done.

13 24 46 52 20 9

13 24 46 52 20 9

13 24 46 20 52 9

13 24 46 20 9 52.

→ after one complete round max will gone to last

→ Here we move 6 rounds, For swapping 6 nos in an array.

Now

13 24 20 46 9 52

13 24 20 9 46 52

done.

→ If we did some optimization in this to

Best TC = O(n)

↳ in case of already sorted array

Now

13 24 20 9 46 52

13 20 24 9 46 52

13 20 9 24 46 52

done.

Now

13 9 20 24 46 52

done

Now

9 13 20 24 46 52

Full done.

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

TC = n + n-1 + n-2 + ... + 2 + 1

So, $\frac{n \cdot (n+1)}{2}$

So, $\frac{n^2 + n}{2}$

So, $TC = O(n^2)$.

→ worst

→ average

## Insertion sort → takes an element and place it in current position.



arr[] = | 14 | 9 | 8 | 12 | 6 | 8 | 13 |

↳ Isko apne jagaho pe phuchao.

yaha ↳

| 6 | 8 | 9 | 12 | 13 | 14 | 15 |

```
For ( int i=0 ; i<=n-1 ; i++){
    int j = i;
    while ( j>0 && arr[j-1] > arr[j]){
        int temp = arr[j-1];
        arr[j-1] = arr[j];
        arr[j] = temp;
        j--;
    }
}
```

Eg 5 4 3 2 1

4 5 3 2 1

4 3 5 2 1

3 4 5 2 1

3 4 2 5 1

3 2 4 5 1

2 3 4 5 1

2 3 4 1 5

2 3 1 4 5

2 1 3 4 5

1 2 3 4 5

$O(n \cdot \frac{n+1}{2})$

So, $O(n^2)$

So, TC = $O(n^2)$

- worst
- average.

But Best TC = $O(n)$

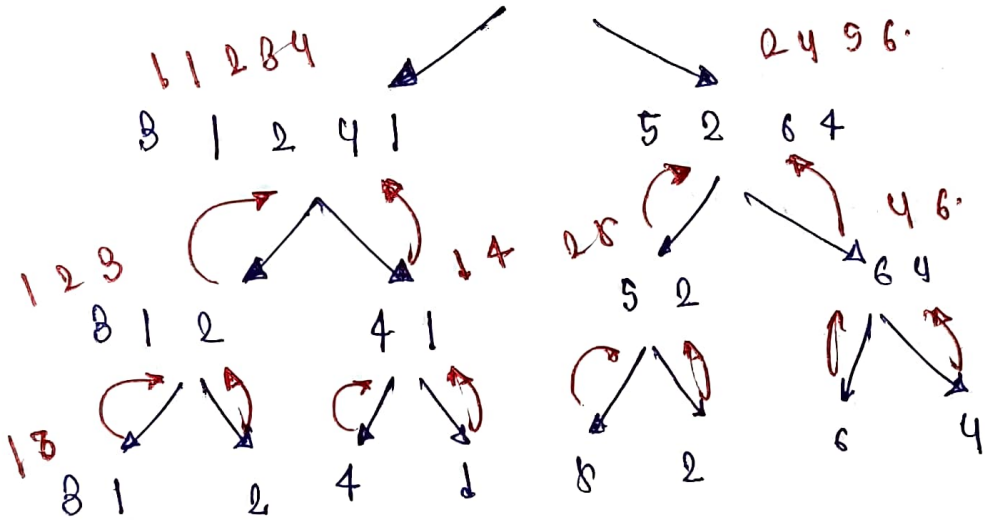
Q4 Merge sort → use merge function (sorting algorithm).

arr[] = [3, 1, 2, 4, 1, 5, 2, 6, 4].

→ Recursive algorithm.

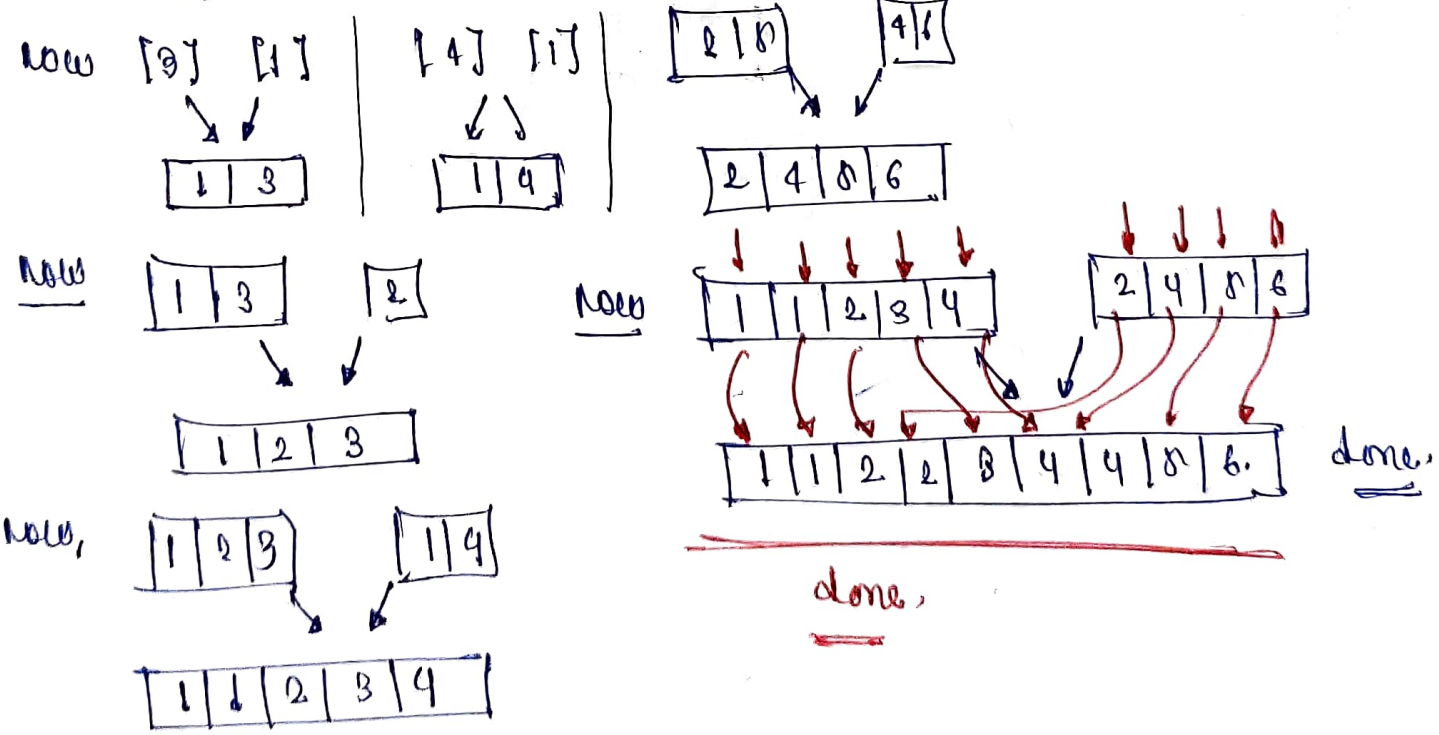
use divide and merge.

Here 9 nos so 4/5 or 5/4.

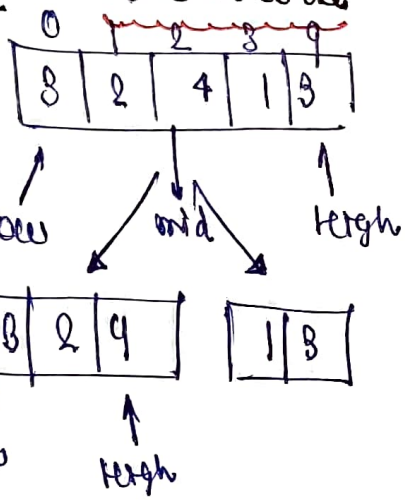



not divided further.

now



Low




Pseudocode

```
merge_sort (arr, low, high) {
    mid = (low + high) / 2 ;
    merge_sort (arr, low, mid) ;
    merge_sort (arr, mid+1, high) ;
    merge (arr, low, mid, high) ;
}
```

Now

Base case

[arr, 1, 1]

low high

Base case

```
if (low >= high)
    return ;
```


```
merge (arr, low, mid, high) {
temp ~~arr~~ → [ ]
    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]);
    left++ ;
}
while (right <= high) {
    temp . add (arr [right]);
    right++ ;
}
```

left

third

right

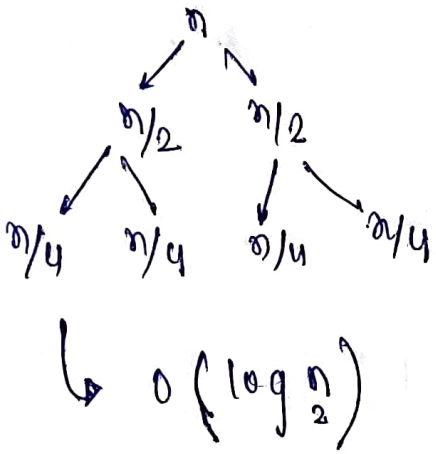
For ( i = low → high) {

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

}

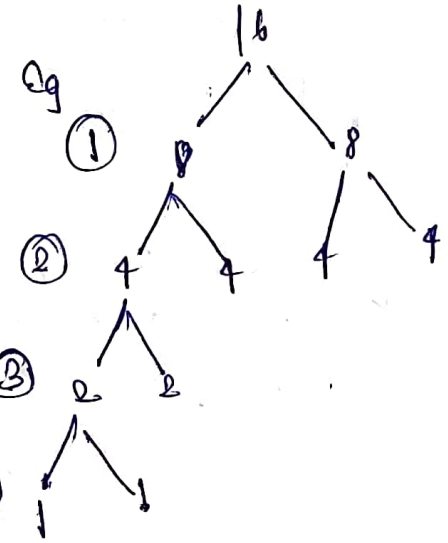
↳ done.

## Now time complexity




↳ $O(\log_2 n)$

merge sort




$\log_2 16 = 4$

Now, merge for 8 elements use 8 steps. (operations)

↳ $\theta(n)$

So, $TC = O(n \log_2 n)$.

↳ Best
- average
- worst.

$SC = \theta(n)$

↳ worst.

# Quick sort



arr[7] = [4 6 2 5 7 9 1 3] ⟶ TC = O(n log n)
SC = O(1).

better in comparison to merge sort in case of space.

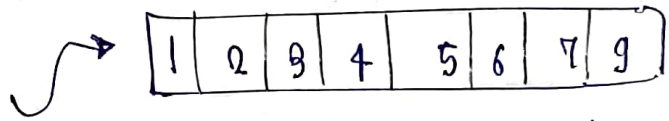
- Pick a Pivot and Place it in its correct Place in the sorted array.

Eg 4 6 2 5 7 9 1 3

1 2 3 4 5 6 7 9

Put it in on correct Place as sorted array.

- Smaller on the left and Larger on the right.




array is automatically get sorted.

Now 4 6 2 5 7 9 1 3

2 1 3 | 4 | 6 5 7 9

Now 2 1 3

1 2 3

left

Now 6 5 7 9

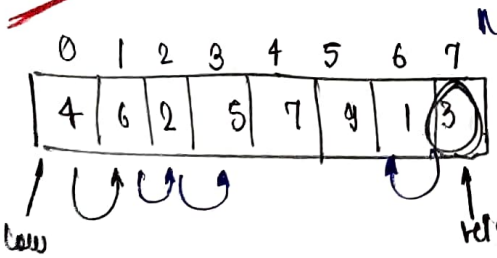
5 6 7 9

right

⇒ How to choose Pivot ?

- 1st element in the array.
- last element in the array.
- median of the array
- random element of the array.

Also




Now, 7 9

7 9

done

4 3 2 5 7 9 1 6

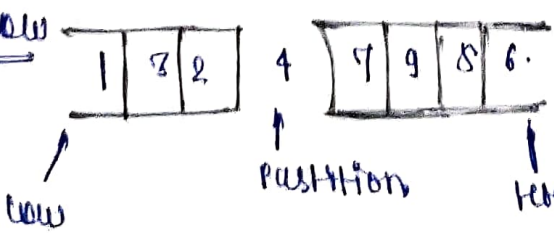
4 3 2 1 7 9 5 6

less than Pivot

Larger than pivot

- It is also divide and conquer algorithm

Now




Perform quick sort again.

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

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

→ Time complexity

- Not equal halves distribution.

$$TC = O(n \log n)$$

$$SC = O(1).$$

# SORTING ALGORITHMS

## 1.) SELECTION SORT

## CODE

```cpp
class Solution {
public:
    void selectionSort(vector<int>& arr) {
        int n = arr.size();

        for (int i = 0; i <= n - 2; i++) {
            int mini = i;
            for (int j = i; j <= n - 1; j++) {
                if (arr[j] < arr[mini]) {
                    mini = j;
                }
            }
            int temp = arr[mini];
            arr[mini] = arr[i];
            arr[i] = temp;
        }
    }
};
```

## 2.) BUBBLE SORT

## CODE

```cpp
class Solution {
public:
    void bubbleSort(vector<int>& arr) {
        int n = arr.size();

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

# SORTING ALGORITHMS

## 3.) INSERTION SORT

## CODE

```
class Solution {
public:
    void insertionSort(vector<int>& arr) {
        int n = arr.size();

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

# SORTING ALGORITHMS

## 4.) MERGE SORT

## CODE

```
void merge(vector<int> &arr, int low, int mid, int high){
    vector<int> temp;
    int left=low;
    int right=mid+1;
    while(left<=mid && right<=high){
        if(arr[left]<=arr[right]){
            temp.push_back(arr[left]);
            left++;
        }
        else{
            temp.push_back(arr[right]);
            right++;
        }
    }

    while(left<=mid){
        temp.push_back(arr[left]);
        left++;
    }

    while(right<=high){
        temp.push_back(arr[right]);
        right++;
    }

    for(int i=low;i<=high;i++){
        arr[i]=temp[i-low];
    }
}

void ms(vector<int> &arr, int low, int high){
    if(low==high)
    return;
    int mid=(low+high)/2;
    ms(arr, low, mid);
    ms(arr, mid+1, high);
    merge(arr, low, mid, high);
}

void mergeSort(vector < int > & arr, int n) {
    ms(arr, 0, n-1);
}
```

# SORTING ALGORITHMS

## 5.) QUICK SORT

## CODE

```
#include <bits/stdc++.h>

int partition(vector<int> &arr, int low, int high){
    int pivot=arr[low];
    int i=low;
    int j=high;

    while(i<j){
        while(arr[i]<=pivot && i<=high-1){
            i++;
        }
        while(arr[j]>pivot && j>=low+1){
            j--;
        }

        if(i<j){
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[low], arr[j]);
    return j;
}

void qs(vector<int> &arr, int low, int high){
    if(low<high){
        int pIndex=partition(arr, low, high);
        qs(arr, low, pIndex-1);
        qs(arr, pIndex+1, high);
    }
}

vector<int> quickSort(vector<int> arr)
{
    qs(arr, 0, arr.size()-1);
    return arr;
}
```

THANKYOU!