Algorithms

- Linear Search —— 2
- Binary Search —— 3, 4
- Bubble Sort —— 5, 6
- Insertion Sort — 7, 8
- Selection Sort — 9, 10
- Quick Sort — 11, 12
- Merge Sort — 13, 14
- Randomized Quick Sort — 15

Linear Search

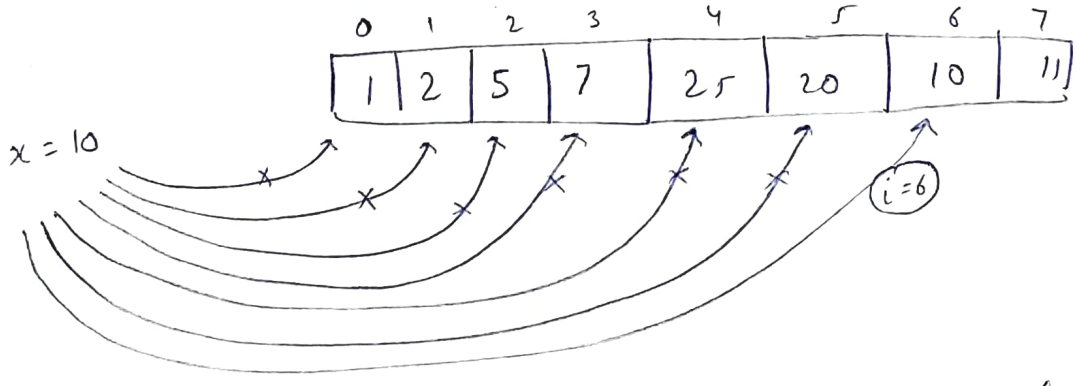
x → To search
arr[] → Search space

Approach :-

- Start from left most element of arr[] and one by one compare 'x' with elements of arr[];
- If x is found, return index;
- If arr[] is finished, return -1;

e.g x = 10
arr[] = {1, 2, 5, 7, 25, 20, 10, 11}

```
int search (int arr[], int n, int x)
{
    for(int i=0; i<n; i++)
    {   if (arr[i] == x)
        {
            return i;
        }
    }
    return -1;
}
```



To optimize :- → Make 2 searches simultaneously one from left & other from right.

Binary Search

arr() = Search space
x = key

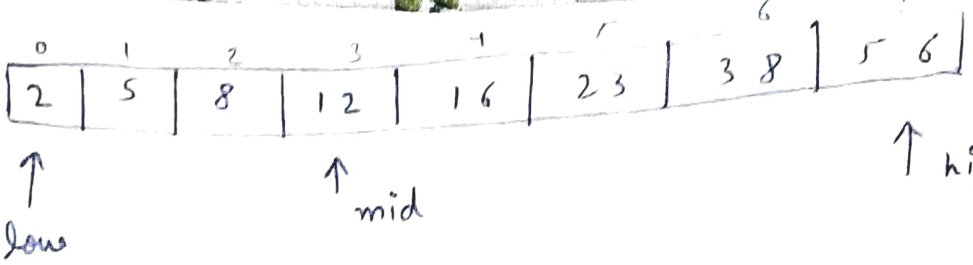
Search in (sorted) array repeatedly reducing (dividing) the search space by half.

Approach:

- Compare x with the middle element of search space; return if matches
- If x is smaller than middle, reduce the search space to left half.
- If x is greater " " " " " " right half
- If search space is finished, return -1

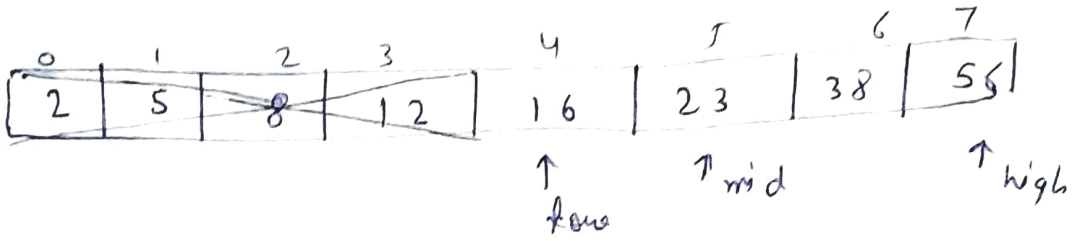
```
int binarySearch ( int arr[] , int low , int high , int x )
{
    if ( high >= low )
    {
        int mid = low + (high - low) / 2 ;

        if ( arr[mid] == x)
        {
            return mid ;
        }
        else if ( arr[mid] > x )
        {
            return binarySearch (arr, low, mid-1, x);
        }
        else
        {
            return binarySearch (arr, mid+1, high, x);
        }
    }
    return -1 ;
}
```

e.g



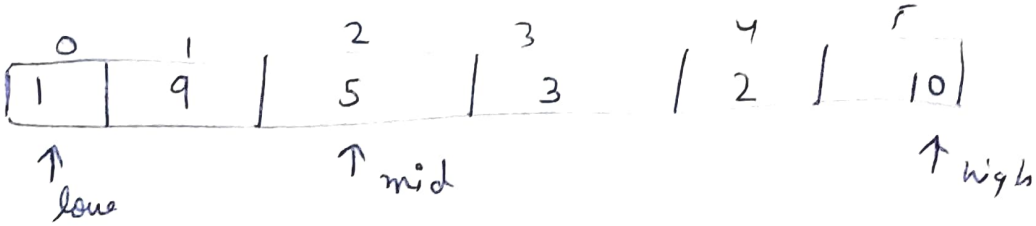
x = 23

∴ 23 > 12 Right Half to be searched



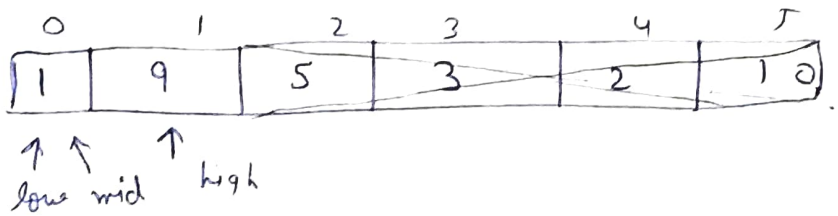
∴ 23 == 23 return (5)

e.g



x = 1

∴ 1 < 5 Left Half to be searched.

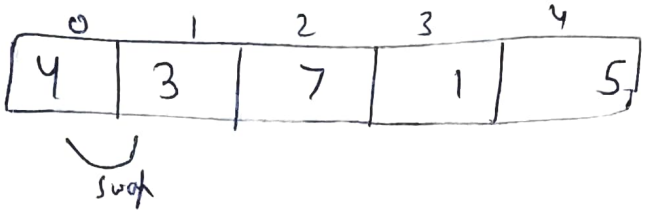


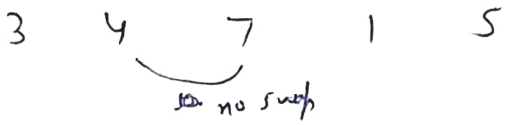
∴ 1 == 1 return (0)

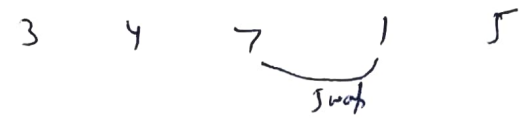
Bubble Sort

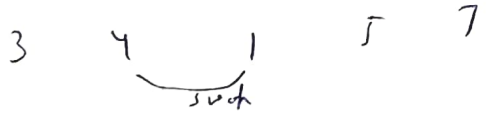
↳ Swap adjacent element if in wrong order.

↳ After one loop, max. will be at end.

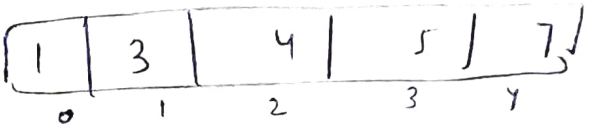








Sorted



Approach

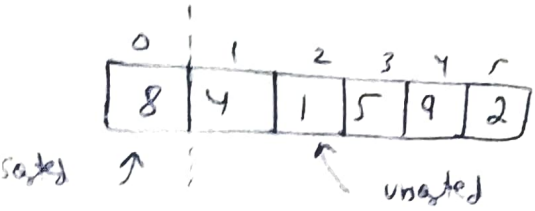
- Compare adjacent elements
- If left is bigger then swap
- Do above till all elements are at correct position

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

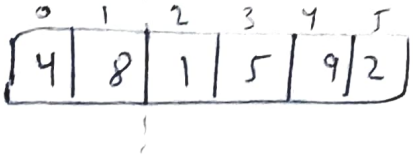
i = 0
j = 0, 1, 2, 3, 4

Insertion Sort

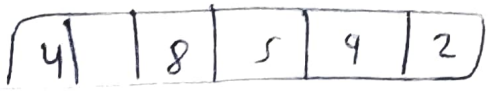
Pick an element & place it in it's order (sorted)



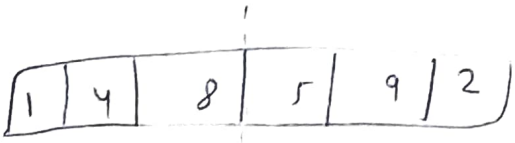
temp = 4
4 < 8
shift
arr[0] = 4



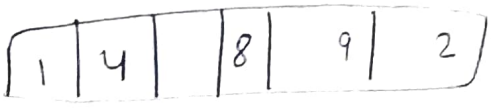
temp = 1
1 < 8
shift



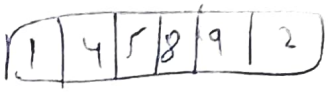
1 < 4
shift
arr[0] = 1

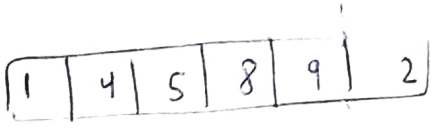


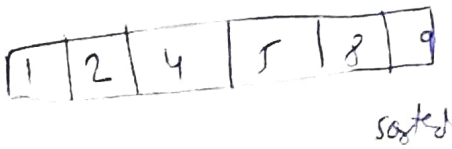
temp = 5
5 < 8
shift



5 > 4
arr[2] = 5



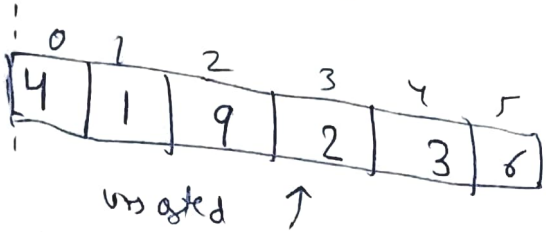




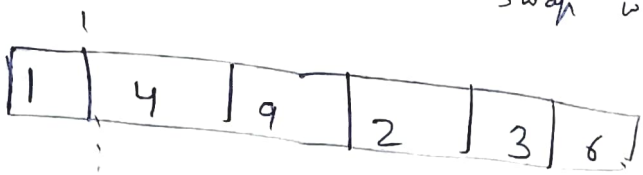
```
for (int i=1; i<n; i++)
{
    temp = arr[i];
    for (int j = i-1; j>=0; j--)
    {
        if (arr[j] > temp)
        {
            arr[j+1] = arr[j];
        }
        else {
            break;
        }
    }
    arr[j+1] = temp;
}
```

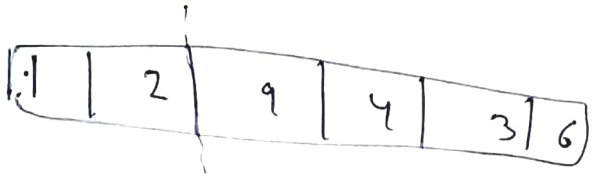
Selection Sort

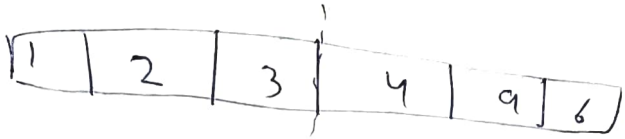
Select minimum element from the array and place it at the beginning.

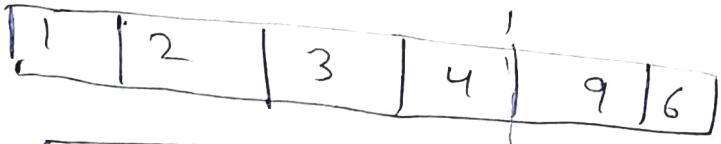


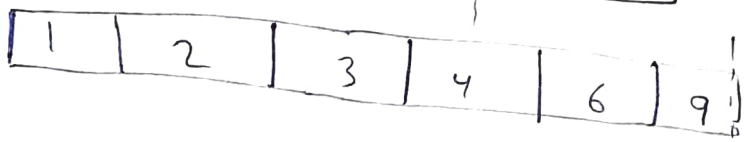
choose min i.e. 1 swap with 1st of unsorted









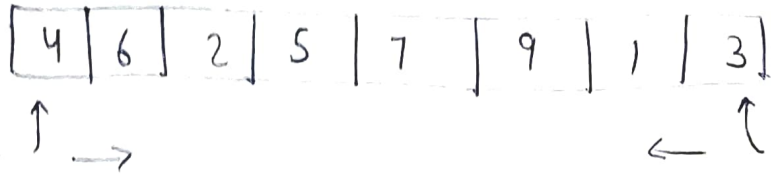


```
for (int i = 0; i <> n-1; i++)
{   int min=arr[i]; int index = i
    for(int j = i+1; j < n; j++)
        {
            if (arr[j] < min)
                { index = j;
                  min = arr[j];
                }
        }
    swap (arr[index], arr[i]);
}
```

Quick Sort

- Divide & conquer algorithm.
- Pick an element (pivot) and position the array around the pivot element

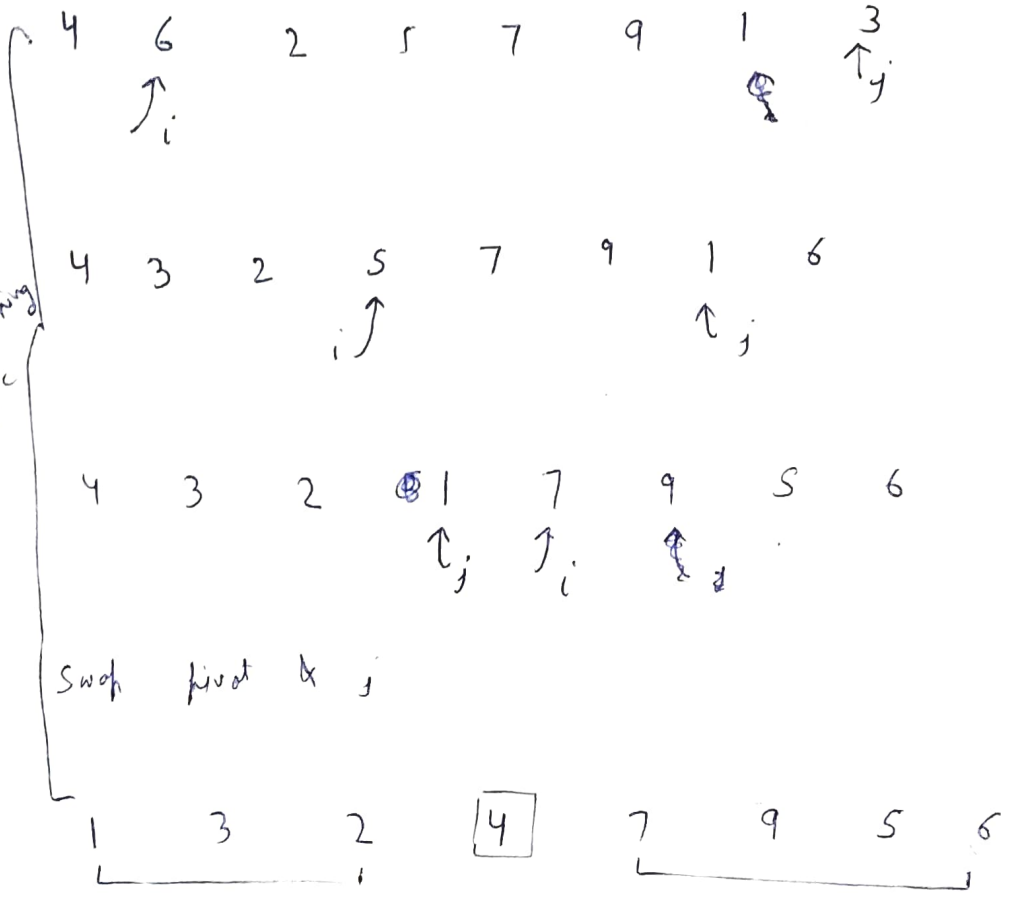
pivot



move i to right until found greater than pivot

move j to left until found smaller " "

Swap i, j



```
int partition (int arr(), int l, int h) {
    int pivot = arr[l]; int pivot_index = l;
    while ( l < h )
    {
        while ( arr[l] <= pivot)
        { l++; }
        while ( arr[h] > pivot )
        { h--; }
    if (l<h) -> swap ( arr[l], arr[h]);
    }
    swap ( arr[h], arr[pivot_index]); return h;
}
int arr(),
quickSort ( int l, int h) {
    if ( l < h ) {
        int pivot = partition (arr, l, h);
    }
    quickSort ( arr, l, pivot - 1),
    quickSort (arr, pivot +1, high),
    }
}
```

To optimise worst case, if sorted array given, we can make either middle element pivot or any random element pivot.

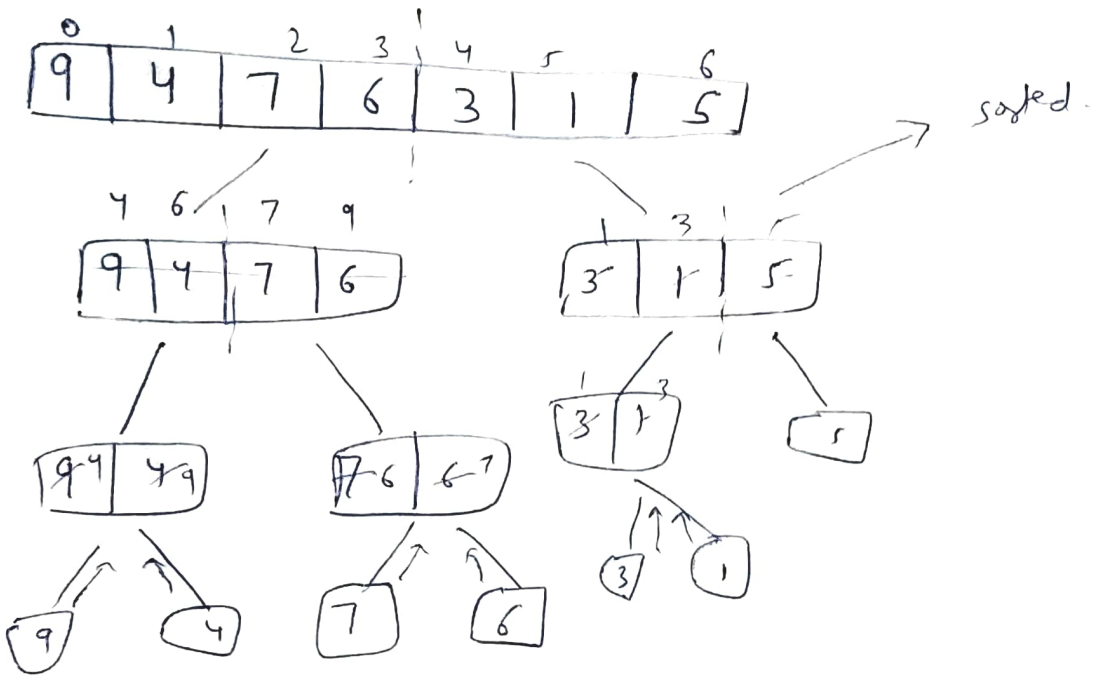
4 1 3 9 7

1 3 4 9 7

Merge Sort

↳ Divide & Conquer Algorithm

↳ Divide into smaller units, Sort them, Merge them in sorted order
  ↳ which can't be further divided.



```
mergeSort( arr[], left, right)
{
    if( left < right)
    {
        int mid = left + (right - left) /2;
        mergesort( a, left, mid);
        mergesort( arr, mid+1, right);
        merge( arr, left, mid, right);
    }
}

merge( arr[], left, mid, right)
{
    int temp[ right - left + 1];
    int i = left;
    int j = mid+1;
    int k = left;

    while( i <= mid && j <= right){
        if( arr[i] < arr[j])
            temp[k] = arr[i], i++
        else
            temp[k] = arr[j] ; j++

        k++
    }

    if( i > mid) {
        while( j <= right) {
            temp[k] = arr[j];
            j++;
            k++;
        }
    }else{
        while( i <= mid) {
            temp[k] = arr[i];
            i++;
        } k++;
    }
```

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

copy ele from temp to arr

Randomized Quick Sort

Pick a random element to be pivot.

↳ We will use the same program but we will just swap the low element with a random element

```
#include <time.h>
partition_random ( arr[], low, high) {
        srand ( time (NULL));
        int random = low + rand () % (high - low);
        Swap ( &arr[low] , &arr (random ]);
        return partition ( arr (@ , low , high);
}
```