Arr=[12,13,16,23,27,45,56,58,62,65,70]

Search number 13 how many comparisons will be there

1. Low=0, high=10

Mid=5 arr[5]==num 45==13

2. Low=0, high=4
   Mid=2 arr[2]==num 16=13
3. Low=0, high=1
   Mid=0 arr[0]==num 12==13
4. Low=1 high=1
   Mid=1 arr[mid]==num 13==13

Search 15 how many comparisons will be there

1. Low=0, high=10

Mid=5 arr[5]==num 45==15

2. Low=0, high=4
   Mid=2 arr[2]==num 16=15
3. Low=0, high=1
   Mid=0 arr[0]==num 12==15
5. Low=1 high=1
   Mid=1 arr[mid]==num 13==15
4. Low=2 high=1
   Not found

Search 56 how many comparisons -→ 3

1. Low=0, high=10

Mid=5 arr[5]==num 45==56

2. Low=6 high=10
   Mid=8 arr[8]=num 62=56

3. Low=6,high=7
   Mid=6 arr[6]=num 56=56

To search number 70 how many comparisons will be there

1. Low=0, high=10

Mid=5 arr[5]==num   45==56

2. Low=6 high=10
   Mid=8 arr[8]=num   62=56

3. Low=9, high=10
   Mid=9 arr[9]=num   65=70
4. Low=10, high=10
   Mid=10 arr[10=num 70= 70

   Time complexity

Time complexity --- (log n)

Binary search is faster than sequential search

For binary search, the data has to be in sorted order

Complexity of Sorting Algorithms

The efficiency of any sorting algorithm is determined by the time complexity and space complexity of the algorithm.

1. Time Complexity: Time complexity refers to the time taken by an algorithm to complete its execution with respect to the size of the input. It can be represented in different forms:

- Big-O notation (O)-→ worst case
- Omega notation (Ω) -→ best case
- Theta notation (Θ)-→average case

2. Space Complexity: Space complexity refers to the total amount of memory used by the algorithm for a complete execution. It includes both the auxiliary memory and the input.

The auxiliary memory is the additional space occupied by the algorithm apart from the input data. Usually, auxiliary memory is considered for calculating the space complexity of an algorithm.

Example : to=ime complexity O(n)   and space complexity O(1)

```
Int[] arr={65,23,24,25}

Int x=23;

For(int i=0;i<arr.length;i++){   4   10    100

If(arr[i]==x)---1

   Return i

}
```

Time complexity

Time complexity is usually calculated in terms of size of data, mostly considered as n

It is not number of seconds required to execute the code, but it is number of times main operation performed to complete the task

For every algorithm we have 3 types of time complexity

1. best case--- for best case we use Ω symbol
2. average case-→ for average case we use θ
3. worst case-→ for worst case we use O(big O) symbol

sequential search

1. To calculate time complexity of search, the main operation we perform is comparison so the number of comparisons happen, will help us to calculate time complexity

the best case is the value found at the first position – so the number of comparison are always 1

and hence the time complexity of best case is Ω(1)

In average case the value may be available at the centre of the array, hence the number of comparisons are n/2, in time complexity, we do not consider constants, hence the time complexity is θ(n)

In worst case, the value is found at the end, or not available, in both cases number of comparisons is same as array length hence the time complexity is O(n)

Similarly for every algorithm we calculate space complexity also

It is nothing but how much memory space is used by the algorithm-→ for any searching algorithm the space requirement will be n and hence space complexity O(n)

for(int i=0;i<n;i++)
  for(int j=0;j<n;j++)
	O(n^2)

for(int i=0;i<n;i++)	O(n)

for(int i=0;i<n;i++){}
for(int i=0i<n;i++{}
	O(n)

for(int i=0;i<n;i++){   n
   for(i=0;i<3;i++){
   }
}
	O(n)

a+b a*b	O(1)

Searching algorithm

1. sequential search
    a. It is usually used when the array is not sorted, so to search the value, every value is compared sequentially, so it is called as sequential search
    b. It is slower than binary search

2. binary search
    a. It is used only when the data is sorted,
    b. Its time complexity is O(log n)

Sorting Algorithm

1. Bubble sort
   We compare adjacent element to sort the data, this is simplest and easiest way of sorting data, its time complexity in O(n^2)
2. Selection sort
3. Insertion sort
4. Quick sort
5. Merge sort
6. Heap sort