When we pass data to a function it can be done in 2 different ways

1. call By value
2. call by reference

1. Call by value-----In java when you pass primitive data type value to a function, then it is call by value, so the changes done to these values inside function, then the original data will not be affected
2. call by reference -→ In java when you pass objects as a parameter to a function, then it is call by reference, so the changes done within a function will be reflected in the original data.

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 csalculate 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 center 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 are 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)

Why time complexity of tree base algorithm is log(n)

It is not in general. The height of a *balanced* binary tree is of the order $\log_2 n$ $\log_2 n$, or to be more precise, $\lceil \log_2(n+1) \rceil$

The k'th level of a binary tree (k=0 for the level of the root node) can incorporate 2^k nodes. This makes a tree with l levels in total contain

$$\sum_{i=0}^{l-1} 2^i = 1 + 2 + 4 + \ldots + 2^{l-1} = 2^l - 1$$

nodes in total.

Now, for a full balanced binary tree,

$2l-1=n2l-1=n$

$\Leftrightarrow 2l=n+1 \Leftrightarrow 2l=n+1$

$\Leftrightarrow l=\log2(n+1) \Leftrightarrow l=\log2(n+1)$

To handle the case that the balanced binary tree is not full at the lowest level, we have to round up:

$l=\lceil \log2(n+1) \rceil$

Sorting algorithm

1. bubble sort

    a. It is simplest sorting algorithm, in which we will compare consecutive elements, and keep on moving either lowest value to the end, or highest value to end
    b. the time complexity of the algorithm is $O(n^2)$
    c. In the application if the sorting of data is required very rarely, then many times people prefer using bubble sort, to sort the data
2. insertion sort
3. merge sort
4. quick sort
5. selection sort
6. heap sort