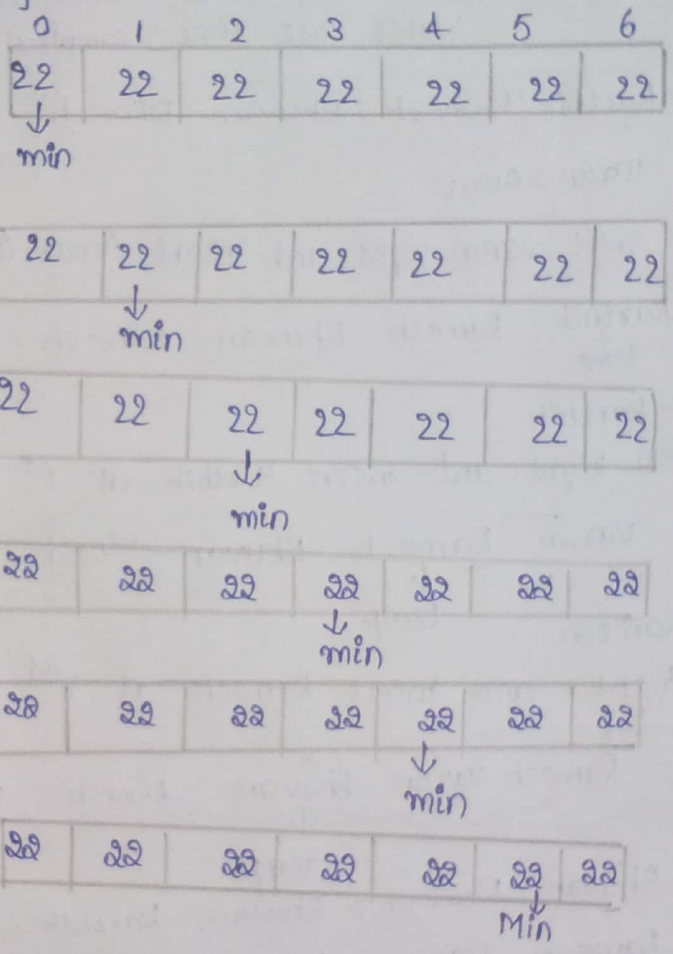
1. Assume that there is a list [22, 22, 22, 22, 22, 22, 22] What happens when Selection sort is Applied on the list? Explain?

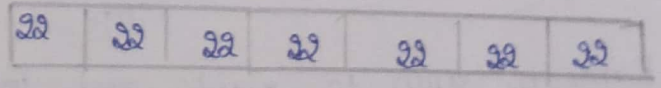
Selection Sort :- Selection sort is an algorithm that we select and search for the lowest element. Then the lowest element is swapped with current element.

Given array



In the above list all the elements are same. So there is no swapping at all.

Output :-



Each element with minimum element in unsorted elements are same. Hence no element is exchanged. Since time complexity of selection sort of given array is $O(n^2)$. When we search for small element we get number itself. It happens to hole array.

2. Sort the following list using insertion sort.

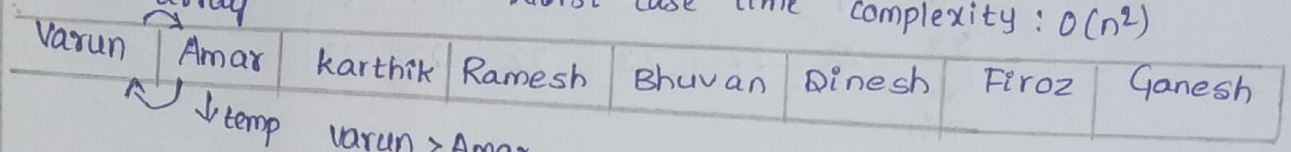
Varun Amar karthik Ramesh Bhuvan Dinesh Firoz Ganesh

Insertion Sort: It is also a sorting algorithm. But it is more efficient because it replaces swapping with shifting.

Here every element is compared to its previous element. If we found only bigger element before the key, then we shift their places. Best case time complexity : $O(n)$

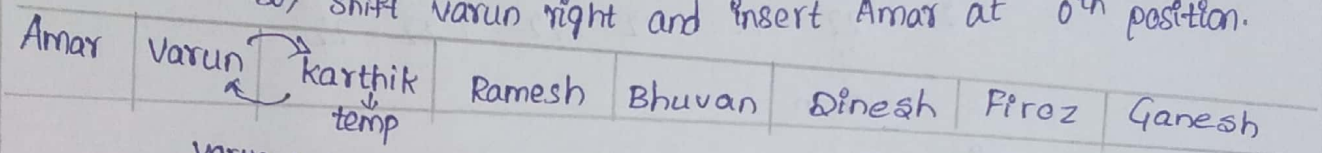
Worst case time complexity : $O(n^2)$

Given array



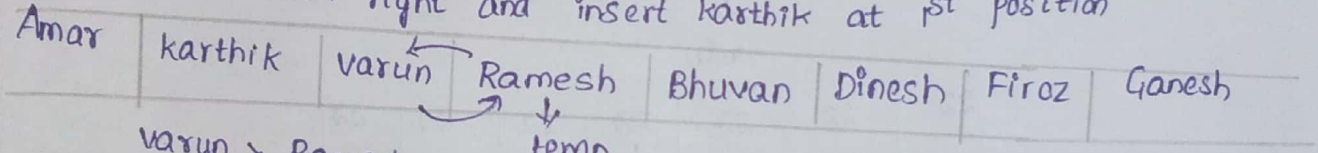
varun > Amar

So, shift varun right and insert Amar at 0th position.



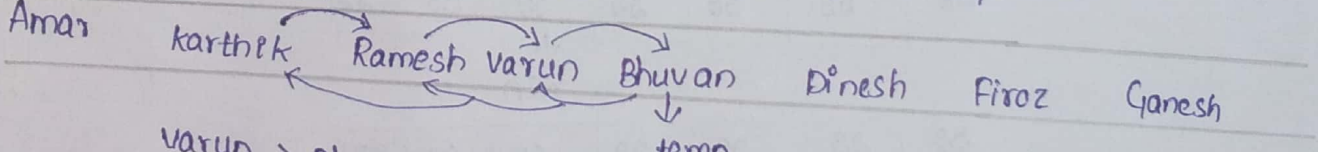
varun > karthik

So, shift varun right and insert karthik at 1st position



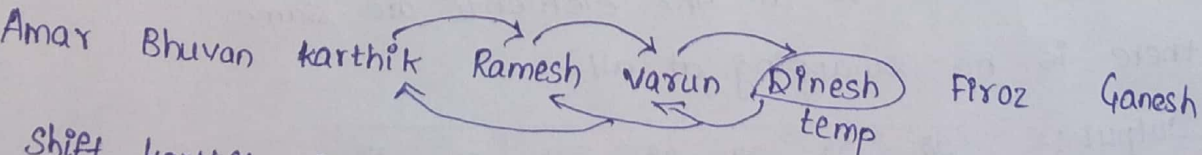
varun > Ramesh

Shift varun Right and insert Ramesh at 2nd position.

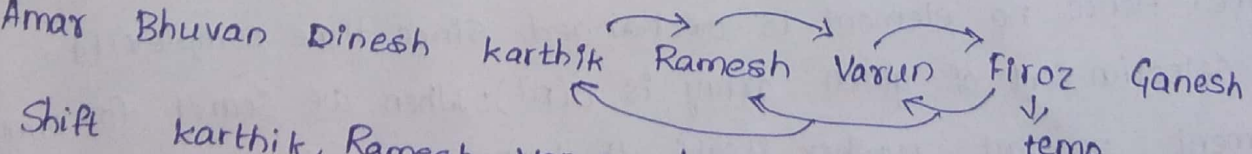


varun > Bhuvan, Ramesh > Bhuvan, karthik > Bhuvan

Shift karthik, Ramesh, varun to right and insert Bhuvan at position 1.



Shift karthik, Ramesh, varun to right and insert Dinesh at 2nd position.



Shift karthik, Ramesh, varun to right and insert Firoz at 3rd position.

Amar Bhuvan Dinesh Firoz karthik Ramesh Varun Ganesh

Shift karthik, Ramesh, Varun to Right and insert Ganesh at 4th pos.

Amar	Bhuvan	Dinesh	Firoz	Ganesh	karthik	Ramesh	Varun

This is the sorted list.

3. Sort the following numbers using Quick Sort:

67 54 9 21 12 65 56 43 34 79 70 45

Quick Sort: It is based on divide and conquer principle. Take first element of the list as pivot. Swapping are done untill Pivot element reaches its correct position. Then, again take the two sublists and repeat the process until we get a sorted list.

(67)	54	9	21	12	65	56	43	34	79	70	(45)

pivot

compare from right to left for Smaller Swap (67, 45)

45	54	9	21	12	65	56	43	34	(79)	70	(67)

Compare from left to right for Bigger element Swap (79, 67)

45	54	9	21	12	65	56	43	34	(67)	70	79

67 is correct position

Now divide left Sublist and Right sublist.

(45) 54 9 21 12 65 56 43 34 | (70) 79

pivot pivot

Compare left from right

Swap (45, 34)

Right to left No swap

70	79

34 (54) 9 21 12 65 56 43 (45)

left to right. Swap (54, 45) pivot

34 (45) 9 21 12 65 56 (43) 54

Pivot

Right to left. (Swap (45, 43))

34 43 9 21 12 65 56 45 54

pivot

left to right

Swap (65, 45)

34 43 9 21 12 45 56 65 54

pivot

Right to left (No swap). 45 is correct position

Now divide left Sublist & Right sub list.

L.S.L

34 43 9 21 12

pivot

R → L Swap (34, 12)

12 43 9 21 34

pivot

L → R

12 34 9 21 43

pivot

R → L

12 21 9 34 43

pivot

34 is correct position

R.S.L

56 65 54

pivot

Right to left

Swap (56, 54)

54 65 56

pivot

left to right

Swap (65, 56)

54 56 65

left Sublist

12 21 9

pivot

Right sublist

43

Right to left Swap (12, 9)

9 21 12

Pivot

Left to Right Swap (21, 12)

9 12 21

The final sorted list is

9	12	21	34	43	45	54	56	65	67	70	79

4. Implement Linear and Binary Search using recursion

Linear Search:

It is used to find position of an element in the given list. It is also called as Sequential Search.

Algorithm:

Take a list of elements

Compare the key with all the elements in the list Sequentially.

Program:

```
class Test
{
   Static int arr [ ] = { 12, 34, 54, 2, 3};
   /* RecursiveMethod to search x in arr [1..r] */
   Static int recSearch (int arr[], int l, int r, int x)
   {
       if (r<l)
           return -1;
       if (arr [l] == x)
           return l;
       if (arr [r] == x)
           return r;
       return recsearch (arr, l+1, r-1, x);
   }
   // Driver method
   Public Static void main (String [] args)
   {
      Int x = 3;
      int index = // Method call to find x
                  recsearch (arr, 0, arr.length-1, x);
      if (index != -1)
         System.out.println ("Element" + x "is present at index" +
                                                        index);
      else
         System.out.println ("Element " + x " is not present at index");
   }
}
```

Output: Element 3 is present at index 4.

Binary Search: It is also used to find position of an element. In the given list. It is based on divide and conquer principle.
It reduces number of comparisions when compared to linear Search.
Algorithm:
Take array of elements
Find mid position
If key is found at mid, return mid
If key is greater than mid element and repeat procedure (1 to 3)
If key is less than mid element then take left sublist and repeat the procedure (step 1 to 3)
Program:

```
class BinarySearch{
    int binarysearch (int arr[], int l, int r, int x)
    {
      if (r>=l){
          int mid = l+ (r-l) /2;
      // If elements is present at middle itself
       if (arr[mid] ==x)
            return mid;
      // If element is smaller than mid, it can only
      If (arr[mid] >x)
          return binarySearch (arr, l, mid-1, x);
      return binary Search (arr, mid+1,r, x);
    }

    return -1;      // when element is not present
}
// Driver method
public static void main (String args[])
{
   BinarySearch ob = new BinarySearch ();
   int arr[] = {2, 3, 4, 10, 40};
```

```
        int n = arr.length;
        int x = 10;
        int result = ob.binarysearch(arr, 0, n-1, x);
        if (result == -1)
            System.out.println("Element not present");
        else {
            System.out.println("Element found at index" +result);
        }
}
```

5. Explain in brief the various factors that determine the selection of an algorithm to solve a computational problem.

In computer science, a computational problem is a problem that a computer might be able to solve, or a question that a computer may be able to answer.

A computational Problem can be viewed as a infinite collection of instances together with a possible empty, set of solutions for every instance.

i) To solve a computational problem, first we have to choose a datastructures to solve the problem.

ii) Write down the steps to solve the problem

iii) Then try to analyse the time complexity and space complexity taken by the algorithm.

iv) Then try to optimize the algorithm. By thinking can we do better or not?

v) If possible then we should try to optimize the algorithm.

vi) Hence we can solve any computational problem by following above steps.

Time:- The time complexity of an algorithm quantifies amount of time taken by an algorithm to run as a function of length of input.

Space:-

The Space complexity quantifies amount of Space taken by an algorithm to run as a function of length of input.

Ex:-

```
int Add ( int A[ ], int n)                          1        1
{
    int s=0, i;                                     1        1
    for (i=0; i<n; i++)                             n+1+1    2n+2
        s=s+A[i];                                   2        2n
    return s;                                       1        1
}                                                            4n+4
```

In the above code n*2 bytes of memory to store array A[]
2 bytes for integer parameter 'n'.
4 bytes for Sum variable 's' and 'i' (2 bytes each)
2 bytes for return variables.

Totally it takes 4n+4 units of time to compile an algorithm

By following above steps we can solve any algorithm. Mainly we have to solve time and Space Complexity of an algorithm.