DS Assignment-1

NAME:- M.Sivajyothi Regd NO:- 19B01A05D3 Sension:- CSE-C

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.

A:- 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,

22	22	22	22	22	22	22

min

⇒ Here no swap

Alogorithm:-

step1:- set MIN to location 0.

step2:- search the minimum element in the list.

step3:- swap with Value at location MIN.

step4:- Increment MIN to point to next element.

step5:- Repeat until list is stored.

22	22	22	22	22	22	22

min

⇒ Here no swap

22	22	22	22	22	22	22.

min

⇒ Here no swap

22	22	22	22	22	22	22.

min

⇒ Here no swap

22	22	22	22	22	22	22.

Min

⇒ Here no swap

22	22	22	22	22	22	22

Here, no swap

=) In the above list all the elements are same so, there are no swappings at all.

Output:-

22	22	22	22	22	22	22.

2. Sort the following list using Insertion sort
Varun Amar karthik Ramesh Bhuvan Dinesh firoz Ganesh.

A:- Insertion list:- It is also a sorting algorithm. But it is more efficient because it replaces sorting 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.

Given array,

Varun	Amar	karthik	Ramesh	Bhuvan	Dinesh	firoz	Ganesh

Temp

Varun>Amar

so, shift Varun right and insert Amar at 0th position

Amar	Varun	karthik	Ramesh	Bhuvan	Dinesh	firoz	Ganesh

Temp

Varun>karthik

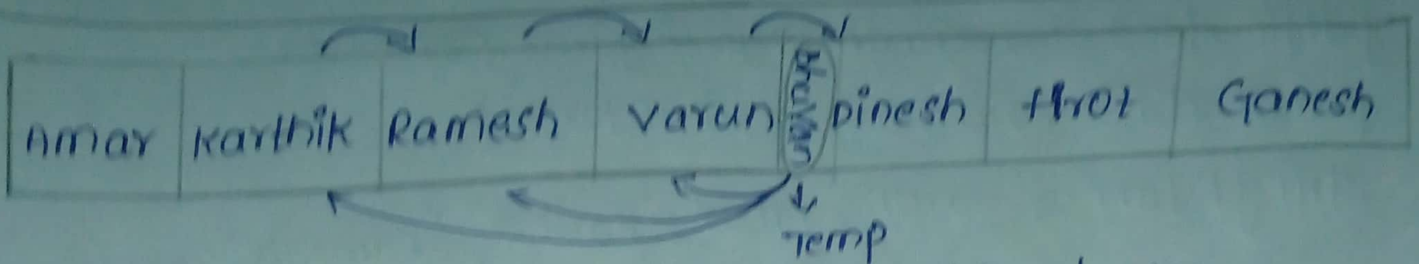
Shift Varun right and insert karthik at 1st position

Amar	karthik	Varun	Ramesh	Bhuvan	Dinesh	firoz	Ganesh

Temp

Varun>Ramesh

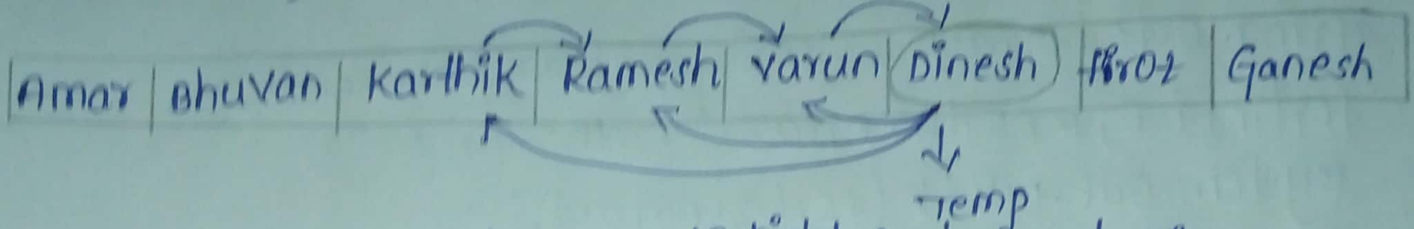
Shift Varun Right & insert Ramesh at 2nd position



Amar	Karthik	Ramesh	Varun	Bhuvan	Dinesh	firoz	Ganesh

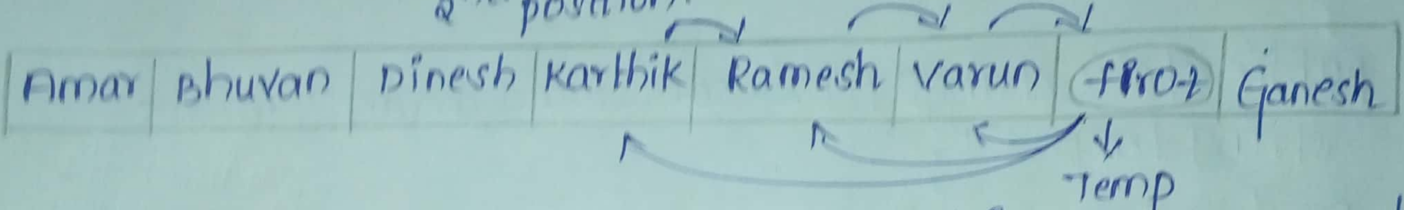
Varun > Bhuvan, Ramesh > Bhuvan, Karthik > Bhuvan

Shift Karthik, Ramesh, Varun to right & insert Bhuvan at 1 position



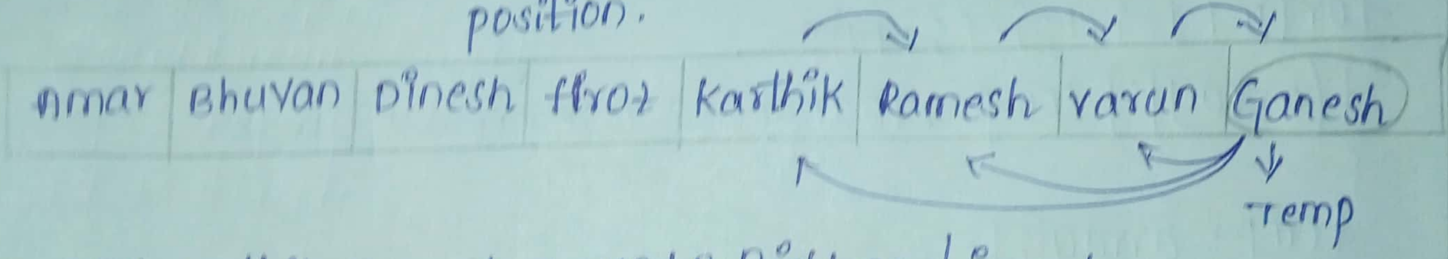
Amar	Bhuvan	Karthik	Ramesh	Varun	Dinesh	firoz	Ganesh

Shift Karthik, Ramesh, Varun to right and insert Dinesh at 2nd position.



Amar	Bhuvan	Dinesh	Karthik	Ramesh	Varun	firoz	Ganesh

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 position.

Amar	Bhuvan	Dinesh	firoz	Ganesh	Karthik	Ramesh	Varun

This is the sorted list.

```
public class sortstringArrayusingInsertion sort{
public static voidmain (string[]args){
string []arr ={"varun,"Amar","Karthik","Ramesh","Bhuvan",
                    "Dinesh","frot","Ganesh"};

int count =0;
stringsorted Array[] = sort_sub(arr, arr.length);
 for(int i=0; i<sorted Array.length; i++){
  system.out.println(sorted Array[i]);
  }
  }
public static string[] sort-sub(string array[].int f){
  string temp =" ";
for(int i=0; i<f; i++){
for(int j=i+1; j<f; j++){
if(array[i].comparTo Ingnore case(array[j]>0)
 {
  temp = array[i];
  array[i] =array[j];
   array[j] =temp;
   }
   }
   }
 return array;
  }
  }
```

Output:-

```
Amar
Bhuvan
Dinesh
froz
Ganesh
karthik.
Ramesh
varun.
```

3. sort the following numbers using Quick sort: 67, 54, 9, 21, 12, 65, 56, 43, 34, 79, 70 and 45.

A:- Quick sort :-

procedure:-

1. We take a list of elements.
2. We identify the first element on the key PNOT element.
3. comparsion starts from right to left (smaller element)
4. latter on from left to Right (bigger element)
5. on Demand, we devide this list into 2 halfs.
6. We repeat steps 3,5 on the left and right sublists.

Given numbers:-

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

key

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

key

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

key

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

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

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

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

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

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

43, 12, 9, 21, 34, |45|, |54|, 65 key, 56, |67|, 70, 79.
key

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

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

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

21, 12, 9, |34|, 43, 45, 54, 56, 65, 67, 70, 79.
key

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

Given, list is sorted using Quick sort

```
public class Quicksort{
publicstaticvoidmain(string[]args){
   inti;
```

```
int[] arr ={67, 54, 9, 21, 12, 65, 56, 43, 34, 79, 70, 45};
   quicksort(arr,0,9);
   system.out.println("\n The sorted array is:\n");
  for(i=0;i<10;i++)
   system.out.println(arr[i]);
}
public static int partition(int a[],int beg, int end){
   int left,right,temp, loc,flag;
   loc=left=beg;
    right=end;
   flag=0;
   While(flag!=1)
   {
     While((a[loc]<=a[right](logc!=right))
     right--;
     if(logc==right)
     flag=1;
     else if(a[loc]>a[right])
     {
      temp=a[loc];
      a[loc]=a[right];
      a[right]=temp;
       loc=right;
     }
     if(flag!=1)
     {
      While((a[loc]>=a[left])(loc!=left))
      left++;
      if(loc==left)
```

```
 flag=1;
 else if (a[loc]<a[left])
  {
   temp=a[loc];
   a[loc]=a[left];
   a[left]=temp;
    loe = left;
   }
  }
 }
 return loc;
}
static void quicksort(int a[],int beg,int end)
 {
  int loe;
  if(beg<end)
  {
   loc=partition(a,beg,end);
   quicksort(a,beg,loc-1);
   quicksort(a,loc+1,end);
   }
  }
}
```

output:-

```
 9
 12
 21
 34
```

```
43
45
54
56
65
67
79
```

4. Implement linear search and Binary search using Recursion.

A:- Linear search:-

```
public class Test {
        static int arr[] = {12,34,54,2,3};
        /* Recursive method to search x in arr[l..r] */
        static int research(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 research (arr, l+1, r-1, x);
      }
      // Driver method
      public static void main(string[] args) {
      int x=3;
     // Method call to find x
    int index = research (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");

   }
 }
```

output:-

```
element 3 is present at index 4.
```

Binary search:-

```
public class Binary search{
   // Returns index of x if it is present in arr[l, r], else
   // return-1
   int binary search(int arr[], int l, int r, int x)
   {
     if (r>=l) {
       int mid=l+(r-l)/2;
    // If the element is present at the middle itself
      if (arr[mid]==x)
       return mid;
  // If element is smaller than mid, then it can only
  // be present in left subarray
    if (arr[mid]>x)
        return binary search (arr, l, mid-1, x);
  // Else the element can only be present in right
   // subarray
   return binary search (arr, mid+1, r, x);
  }
//We reach here when element is not present in array
   return-1;
}
//Driver method to test above
public static void main(string args[]){
   Binarysearch ob=newBinarysearch();
   intarr[]={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);
   }
}
```

output:-

```
Element found at index 3
```

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

A:- The performance of an algorithm can be measured by two properties.

1. Time complexity.

2. space complexity.

1. Time complexity:—of an algorithm quantifies the amount

Of time taken by an algorithm to run as a function of the length of the input.

2. similarly, space complexity:- of an algorithm quantifies the amount of space(or)memory taken by an algorithm to run as a function of the length of the input.

space complexity:- An algorithm represents the amount of memory space needed the algorithm in its life cycle.

space needed by an algorithm is equal to sum of the following two components:

spaces(p) = fixed part(A) + variable part sp(I).

A fixed part that is a space required to store certain data and variables (i.e. simple variables and constants, program size etc--), that are not dependent of the size of the problem.

A variable part is a space required by variables, whose size is totally dependent on the size of the problem.

For Example, recursion stack space, dynamic memory allocation etc....

In other words:- space can be calculated based on amount of space required

1) To store program instructions.

2) To store constant values.

3) To store variable values.

4) And for few other things like function calls, jumping statements etc--,

Ex:-1:-

```
int sum(int A[], int n)
{
 int sum=0,i;
 for(i=0;i<n;i++)
   sum=sum+A[i]
 return sum;
 }
```

In the above piece of code it requires

⟹ 'n*4' bytes of memory to store array variable 'a[]'

⟹ 4 bytes of memory for integer parameter 'n'.

⟹ 8 bytes of memory for Local integer variables 'sum' and 'i' (4 bytes each).

⟹ 4 bytes of memory for return value.

In case of Time complexity, the running time of an algorithm depends upon the following...

⟹ Whether it is a 32 bit machine (or) 64 bit machine.

⟹ Read and write speed of the machine.

⟹ The amount of Time required by an algorithm to perform Arithmetic operators, logical operators, return value and assignment operations etc...,

⟹ Input data.