Regd NO :- | 1 | 9 | B | Q | 1 | A | 0 | 5 | K | 0 |

19BQ1A05K0

Name :- Shaik Heena Kousar

Branch :- CSE

Data Structures

Assignment - 1

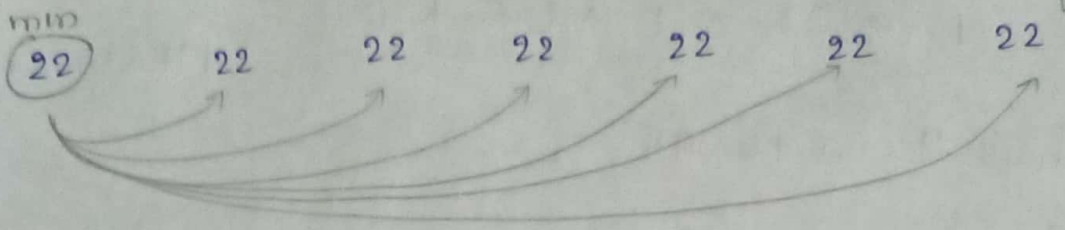
1) Assume that these is a list {22, 22, 22, 22, 22, 22, 22}. What happens when Selection Sort is applied on the list? Explain.

Given list is {22, 22, 22, 22, 22, 22, 22}

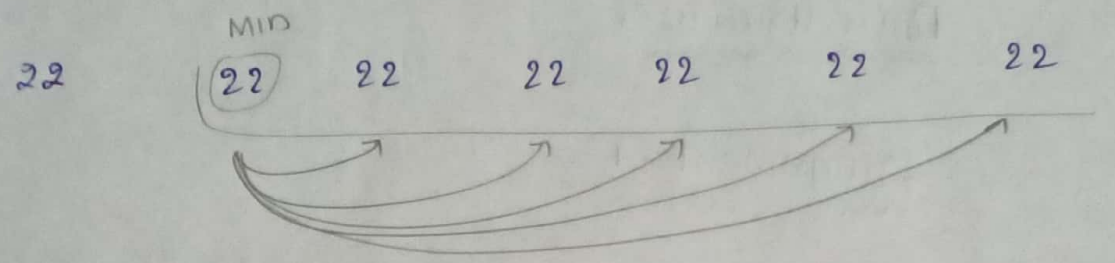
Here all elements are same in the list. So, the list remains same, when selection sort is applied on the list. Selection Sort follows human approach

22 22 22 22 22 22 22

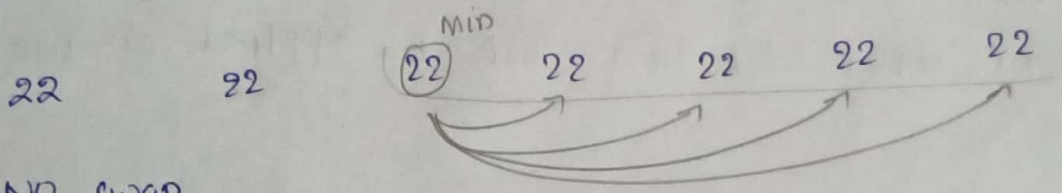
First consider, first element as minimum element and compare the minimum element with each element in the list. If you find any element which is smaller than the minimum element then replace the elements And the small element becomes minimum element. Now again compare the minimum element with the other elements and this process goes on until comparision reaches till the end of the list.



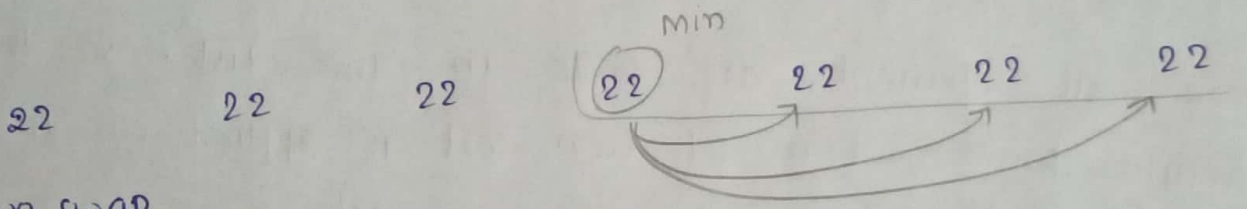
NO swap



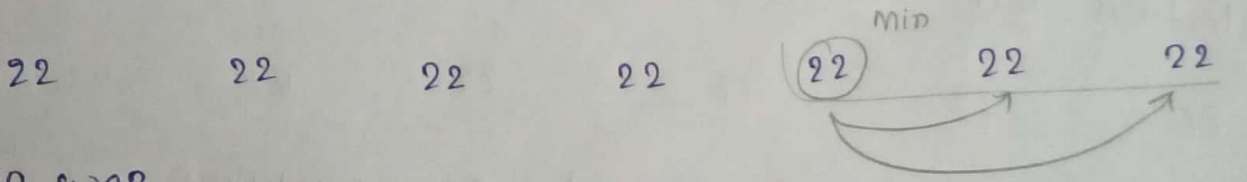
NO swap



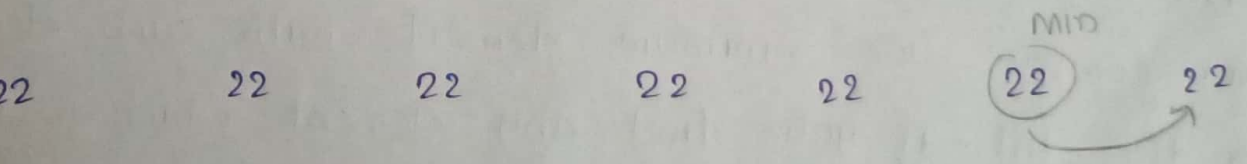
NO swap



NO swap



NO swap



NO swap

22 22 22 22 22 22 22

Here we are looking for the element which is smaller than 22, there is no such element in list so the list remains same.

2) Sort the following list of names using Insertion sort

Varun, Amar, Karthik, Ramesh, Bhuvan, Dinesh, Firoz, Ganesh

In insertion sort, we insert the elements. Every element will be compared with previous elements

	0	1	2	3	4	5	6	7
	Varun	Amar	Karthik	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
temp-1	Amar	Varun	Karthik	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
temp-2	Amar	Karthik	Varun	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
temp-3	Amar	Karthik	Ramesh	Varun	Bhuvan	Dinesh	Firoz	Ganesh
temp-4	Amar	Bhuvan	Karthik	Ramesh	Varun	Dinesh	Firoz	Ganesh
temp-5	Amar	Bhuvan	Dinesh	Karthik	Ramesh	Varun	Firoz	Ganesh
temp-6	Amar	Bhuvan	Dinesh	Firoz	Karthik	Ramesh	Varun	Ganesh
temp-7	Amar	Bhuvan	Dinesh	Firoz	Ganesh	Karthik	Ramesh	Varun

Steps we followed :

1) Assume that first element in the list is in sorted position and all the remaining elements are in unsorted position

2) Take first element from the unsorted postion and insert that element into the sorted postion in the order specified

3) Repeat the above process untill the list is sorted.

3) Sort the following list of numbers using Quick Sort

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

Quicksort :

1) It is a popular Sorting technique
2) It follows divide and concquer principle
3) It is the Quickest Sorting technique when compared to others

procedure

1) Take a list of elements
2) Make the first element as the KEY/PIVOT element
3) Compasision stasts from RIGHT to LEFT
4) Later on from LEFT to RIGHT
5) On demand, we divide the list into 2 halfs
6) We repeat the steps 3, 4, 5 until the list is sorted

key

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

From right to left, we seasch for smaller element than key ← R to L

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

→ L to R From left to right we search for bigger element than key

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

← R to L

→ L to R

From Right to left, there is no such element smaller than key and also from left to right, there is no such element greater than key. So, key element reaches its appropriate position and divide the lists into two sublists

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

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

← R to L

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

→ L to R

34 (45) 9 21 12 65 56 43 54

← R to L

34 43 9 21 12 65 56 (45) 54

→ L to R

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

← R to L

→ L to R

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

← R to L ← R to L

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

→ L to R → L to R

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

← R to L

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

→ L to R ← R to L

(12) key 21 9 43

← R to L

9 21 (12)

→ L to R

9 (12) 21

The sorted list is :

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

4) Implement Linear Search & Binary Search Using Recursion

Linear Search: A Linear Search or Sequential search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

program :

```
import java.util.Scanner;
public class LinearSearch
{
    public static void main(String args[])
    {
        int a[], n, i, key, pos;
```

```
Scanner sc = new Scanner(System.in);
System.out.println("Enter size of array");
n = sc.nextInt();
a = new Int[n]
System.out.println("Enter elements of array");
for(i = 0; i<n; i++)
    a[i] = sc.nextInt();
System.out.println("The array is : ");
for(i=0; i<n; i++)
    System.out.println(a[i]+ "  ");
System.out.println("Enter search key");
key = sc.nextInt();
pos = linearSearch(a, 0, n-1, key);
if (pos==-1)
    System.out.println("key not found");
else
    System.out.println("key found at " +(pos+1));
}
public static linearSearch(int a[], int lb, int ub, int key)
{
  if(lb>ub)
    return -1;
  else if (a[lb]==key)
    return lb;
```

```
        else
            return linearSearch(a, lb+1, ub, key);
    }
}
```

Output :-

```
Enter size of array
6
Enter elements of array
5    7    9    12    14    17
The array is
5    7    9    12    14    17
Enter Search key
7
key found at 2
```

Binary Search :

Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.

program

```
import java.util.Scanner;
public class binarySearch
{
  public static void main(String args[])
  {
    int a[], key, pos, n, i;
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the size of array");
    n = sc.nextInt();
    a = new int[n];
    System.out.println("Enter elements of array");
    for(i=0; i<n; i++)
        a[i] = sc.nextInt();
    System.out.print("The array is :");
    for(i=0; i<n; i++)
        System.out.print(a[i] + "  ");
    Sort(a,n);
    System.out.println("The sorted array is ");
    for(i=0; i<n; i++)
        System.out.println(a[i] + "  ");
    System.out.println("Enter search key");
    key= sc.nextInt();
    pos = binarySearch(a,0,n-1,key);
    if(pos==-1)
        System.out.println("No key found");
```

```
else
    System.out.println(" key found at " + (pos+1));
}
public static int binarySearch(int a[], int lb, int ub, int key)
{
    int mid = 0;
    if (ub >= lb)
    {
        mid = (lb+ub)/2;
        if (a[mid] == key)
            return mid;
    }
    if (a[mid] > key)
        return binarySearch(a, lb, mid-1, key);
    else
        return binarySearch(a, mid+1, ub, key);
}
public static void Sort(int a[], int n)
{
    int i, j;
    for(i = 0; i<n-1; i++)
        for (j=0; j<n-i-1; j++)
            if (a[j] > a[j+1])
            {
                int temp= a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
    }
}
```

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

The performance of an algorithm can be measured by two properties. They are Time and Space

Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Time Complexity : Time Complexity of an algorithm signifies the total time required by the program to run till its execution

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

Space $S(P)$ = 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

In other words

Space can be calculated based on amount of space required

→ To store program instructions

→ To store constant values

→ To store variable values

→ And for few other things like function calls, jumping statements

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'
- 4 bytes of memory for local integer variables 'sum' and 'i'
- 2 bytes of memory for return value

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

- Whether it is running on single processor machine or multi processor machine
- Read and write speed of the machine
- The amount of time required by an algorithm to perform Arithmetic operations, logical operations, return value etc
- Input data