Roll NO- 2003042
Div:- C
Krishna.

1/6

Experiment 1

Aim:- Implement (a) Selection sort.
(b) Insertion sort.

Theory:-

(a) * **Selection sort:-** The selection sort Algorithm sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.
(1) The subarray which is already sorted.
(2) Remaining SubArray which is unsorted.
In every iteration of selection sort, the minimum element (considering Ascending order) from the unsorted subarray is picked and moved to the sorted subArray.

eg:-

	64	25	12	22	11
2nd →	11	25	12	22	64
3rd →	11	12	25	22	64
	11	12	22	25	64

Algorithm:-

Step 1:- Set MIN to location 0.
Step 2:- Search the minimum element in the list.
Step 3:- Swap with value at location MIN
Step 4:- INCREMENT MIN to point to next element.
Step 5:- Repeat until list is sorted.

Analysis:-
Time complexity → $O(n^2)$ → (Two nested for loops)
Auxiliary space- $O(1)$
Swaps → $O(n)$
Selection sort is by default unstable.
Logic:-
first pass → (n-1) comparisions
Second pass → (n-2) comparisions.
3rd pass - (n-3)

→ $(n-1) + (n-2) + (n-3) + \dots + 1$
Best case, worst case and Average case all are $O(n^2)$ in selection sort.

(b) ~~Insertion sort~~:-

Applications:-

The applications of selection sort is as follows:-

1] Selection sort almost always outperforms bubble sort and gnome sort.

2] Can be useful when memory write is a costly operation.

3] When selection sort is preferable to insertion sort in terms of number of writes $(\theta(n))$ swaps. versus $\theta(n^2)$ swaps.

4] It almost always far exceeds the number of writes that cycle sort makes, as cycle sort is theoretically ~~in the~~ optimal in the number of write.

(5) This can be important if writes are significantly more expensive than reads. such as with EEPROM or flash memory where every write ~~the~~ lessens the lifespan of memory.

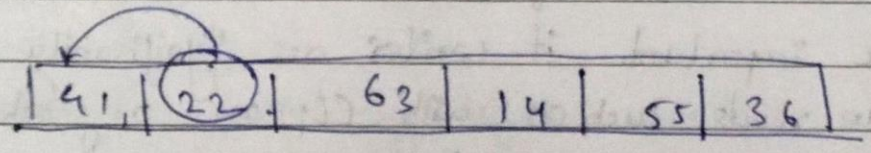
(b) Insertion sort:-

Insertion sort is a simple sorting Algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. values from the unsorted part are picked and placed at the correct position in the sorted part.

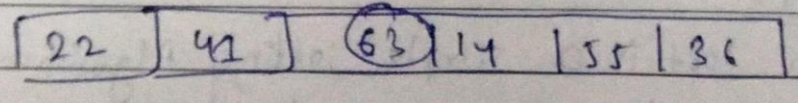
Algorithm of Insertion sort:-

1: for j = 2 to A.length.
2: key = A[j]
3: // Insert A[j] into the sorted sequence A[1..j-1].
4: i = j - 1.
5: while i > 0 and A[i] > key
6: A[i+1] = A[i]
7: i = i - 1
8: A[i+1] = key

example

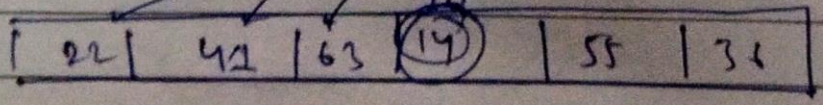


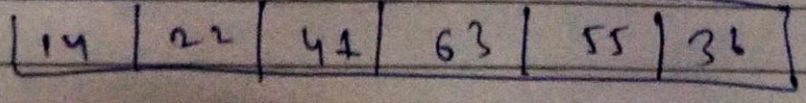
1st iteration key = 22



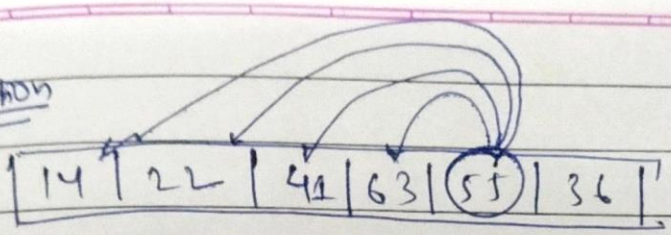
2nd iteration key = 63

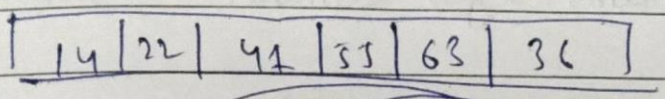
3rd iteration key = 14



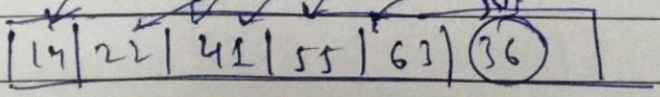


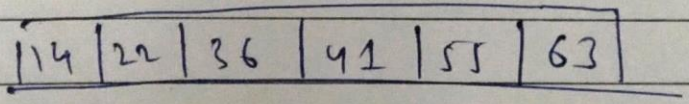
4th iteration





5th iteration





Complexity Analysis

1st iteration → (n-1) comparisons
2nd iteration → (n-2) comparisons

.
.
.

1

→ (n-1) + (n-2) + ... + 1

I → $O(n^2)$

→ Time complexity → $O(n^2)$
Space Complexity → $O(1)$
Best Time complexity → $O(n)$
Average time Complexity → $O(n^2)$
Worst Time Complexity → $O(n^2)$

Applications:-

(i) When Array contains a few elements Insertion sort is preferable.

(ii) When there exist few elements to sort.

—

Program :

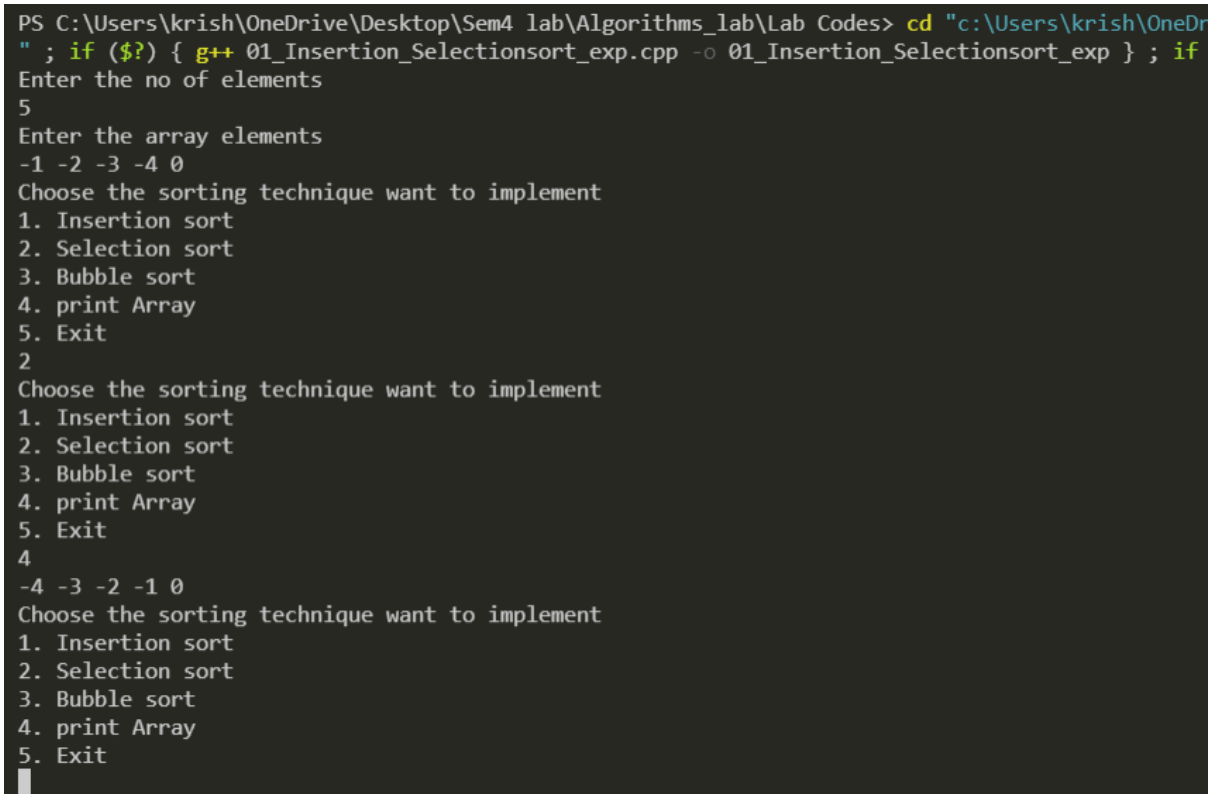
```
#include<bits/stdc++.h>
using namespace std;
void swap(int *a , int *b){
   int temp = *a ;
   *a = *b;
   *b = temp;
}

void bubbleSort(int *arr, int n){
   int i, j ;
   for(i = 0; i < n - 1; i++){
      for(j = 0; j < n - i - 1 ; j++){
         if(arr[j] > arr[j + 1]){
            swap(&arr[j], &arr[j + 1]);
         }

      }
   }
}
void selectionSort(int *arr, int n){
    int i, j;
   for (i = 0; i < n-1; i++){
      for( j = i + 1; j < n ; j++){
         if( arr[i] > arr[j]){
            swap(&arr[i], &arr[j]);
         }
      }
   }

}
void insertionSort(int *arr  , int n){
   int i , key , j ;
   for(i = 1 ; i < n   ; i ++){
      key = arr[i];
      j = i - 1 ;
      while( j >= 0  && arr[j] > key ){
         arr[j + 1] = arr[j];
         j --;
      }
      arr[j + 1] = key ;
```

```
    }
}
void printarray(int *arr , int n){

     for(int i = 0 ; i < n ; i++){
         cout<<arr[i]<<" ";
     }

}
int main(){
    int n;
    cout<<"Enter the no of elements"<<endl;
    cin >> n;
    cout<<"Enter the array elements"<<endl;
    int arr[n];
    for(int i = 0 ; i < n; i++){
        cin>>arr[i];
    }

    do{
        cout<<"Choose the sorting technique want to implement"<<endl;
        cout<<"1. Insertion sort"<<endl;
        cout<<"2. Selection sort"<<endl;
        cout<<"3. Bubble sort"<<endl;
        cout<<"4. print Array "<<endl;
        cout<<"5. Exit "<<endl;
    int ch ;
    cin >> ch;
    switch (ch)
    {
    case 1:
        insertionSort(arr,n);
        break;

    case 2:
        selectionSort(arr,n);
        break;

    case 3:
        bubbleSort(arr,n);
        break;
```

```
    case 4:
        printarray(arr,n);
        cout<<endl;
        break;

    case 5:
        return 0;
        break;

    default:
        break;
    }
    }while(1);

    return 0;

}
```

Output :

```
PS C:\Users\krish\OneDrive\Desktop\Sem4 lab\Algorithms_lab\Lab Codes> cd "c:\Users\krish
" ; if ($?) { g++ 01_Insertion_Selectionsort_exp.cpp -o 01_Insertion_Selectionsort_exp }
Enter the no of elements
5
Enter the array elements
1 3 2 4 5
Choose the sorting technique want to implement
1. Insertion sort
2. Selection sort
3. Bubble sort
4. print Array
5. Exit
1
Choose the sorting technique want to implement
1. Insertion sort
2. Selection sort
3. Bubble sort
4. print Array
5. Exit
4
1 2 3 4 5
Choose the sorting technique want to implement
1. Insertion sort
2. Selection sort
3. Bubble sort
4. print Array
5. Exit
```



Conclusion :

Thus we learned two sorting techniques Insertion sort and selection sort we first learned algorithm then implemented the program and also analysed the Time and space complexity of these algorithms and applications too .