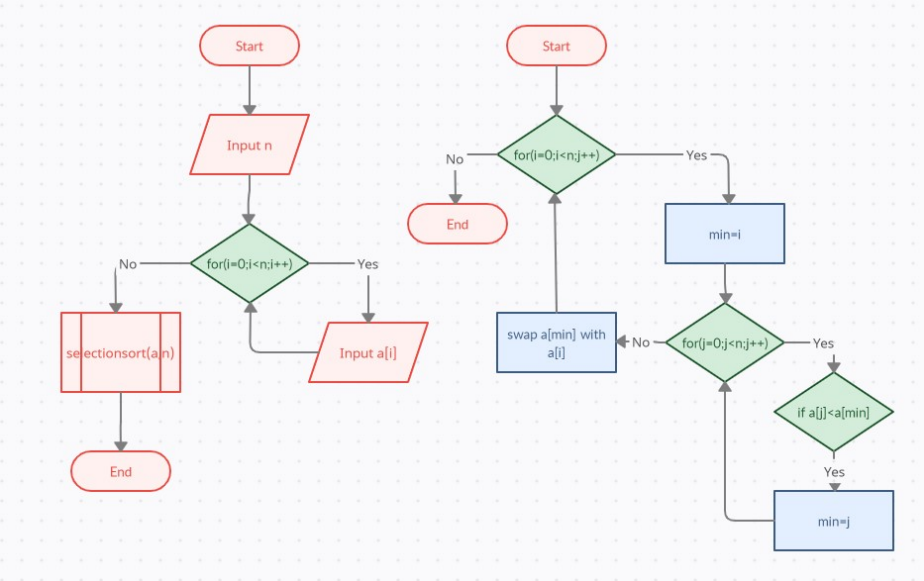
Name	**Hatim Yusuf Sawai**
UID no.	**2021300108**
Experiment No.	**5**

AIM:	Demonstrate the use of one-dimensional arrays to solve a given problem.
Program 1A	
PROBLEM STATEMENT:	The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.
ALGORITHM:	1. START (main) 2. Input n 3. For i=0 4. Input a[i] 5. Repeat step 4 till i<n 6. selectionsort(a,n) 7. for I=0 8. Output a[i] 9. Repeat step 8 till i<n 10. STOP 1. START (selectionsort) 2. For i=0 3. min =i 4. For j=i+1 5. if a[j] < a[min] 6. min = j 7. repeat steps 5-6 till j<n 8. swap(a[min],a[i]) 9. repeat steps 3-8 till i<n 10. STOP

FLOWCHART:



PROGRAM:

```
#include <stdio.h>
void selectionsort(int a[], int n);
void swap(int *, int *);
int main()
{
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int a[n];
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    selectionsort(a, n);
    printf("Sorted array is:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    return 0;
}
```

```c
void selectionsort(int a[], int n)
{
    int i, j, min, temp;
    for (i = 0; i < n; i++)
    {
        min = i;
        for (j = i + 1; j < n; j++)
        {
            if (a[j] < a[min])
                min = j;
        }
        swap(&a[min], &a[i]);
    }
}
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

RESULT:

```
PS D:\C Programming\C Practicals-SPIT\Experiment-5> cd
rog1 } ; if ($?) { .\prog1 }
Enter number of elements: 5
Enter 5 elements:
64 25 12 22 11
Sorted array is:
11 12 22 25 64
PS D:\C Programming\C Practicals-SPIT\Experiment-5>
```

Program 1B	
PROBLEM STATEMENT:	Perform search of a particular element on the above array using binary search. Binary Search will search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty. We basically ignore half of the elements just after one comparison.
ALGORITHM:	1. START (main) 2. Input n 3. For i=0 4. Input a[i] 5. Repeat step 4 till i<n 6. selectionsort(a,n) 7. Input key 8. Binarysearch(a,n,key) 9. STOP 1. START (binarysearch) 2. Low=0, high=n-1 3. While low<=high 4. mid = low+high/2 5. if a[mid]==key 6. Output key,mid+1 7. else if a[mid] > key 8. high = mid-1 9. else 10. low = mid+1; 11. Repeat steps 4-10 till low<=high 12. STOP
FLOWCHART:	

PROGRAM:

```
#include<stdio.h>
void selectionsort(int a[], int n);
void swap(int *, int *);
void binarysearch(int a[],int n,int key);
int main()
{
   int n, i,key;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   int a[n];
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++)
      scanf("%d", &a[i]);

   selectionsort(a, n);
   printf("Which element do you want to search?\n");
   scanf("%d", &key);
   binarysearch(a,n,key);
   return 0;
}
void binarysearch(int a[],int n,int key)
{
   int low = 0, high = n-1, mid;
   while(low <= high)
   {
      mid = (low+high)/2;
      if(a[mid] == key)
      {
         printf("%d is found at index %d\n", key, mid+1);
         return;
      }
      else if(a[mid] > key)
         high = mid-1;
      else
         low = mid+1;
   }
```

```
    printf("%d is not found\n", key);
}
void selectionsort(int a[], int n)
{
    int i, j, min, temp;
    for (i = 0; i < n; i++)
    {
        min = i;
        for (j = i + 1; j < n; j++)
        {
            if (a[j] < a[min])
                min = j;
        }
        swap(&a[min], &a[i]);
    }
}
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

RESULT:

```
PS D:\C Programming\C Practicals-SPIT\Experiment-5> cd '
prog1b } ; if ($?) { .\prog1b }
Enter number of elements: 5
Enter 5 elements:
64 25 12 22 11
Which element do you want to search?
12
12 is found at index 2
PS D:\C Programming\C Practicals-SPIT\Experiment-5>
```

Program 2	
PROBLEM STATEMENT:	Write a program in C to calculate frequency count of each number in ARRAY. Suppose input array is [1 2 1 4 5 2 3 1 4 5 6 2 2 3] output : 1 is appearing 3 times , 2 is appearing 4 times , 4 is appearing 2 times….5 is appearing 2 times , 3 is appearing 2 times , 6 is appearing 1 times
ALGORITHM:	1. START 2. Input n 3. For i=0 4. Input a[i] 5. Repeat step 4 till i<n 6. selectionsort(a,n) 7. For i=0 8. Count=1 9. For j=0 10. If a[i]==a[i+1] 11. Count++ 12. I++ 13. Repeat steps 10-12 till j<n 14. Output I,count 15. Repeat steps 8-14 till i<n 16. STOP
FLOWCHART:	

PROGRAM:

```
#include<stdio.h>
void selectionsort(int a[], int n);
void swap(int *, int *);
int main()
{
   int n, i,count,j;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   int a[n];
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++)
      scanf("%d", &a[i]);

   selectionsort(a, n);
   for(i=0;i<n;i++)
   {
      count=1;
      for(j=0;j<n;j++)
      {
         if(a[i]==a[i+1])
         {
            count++;
            i++;
         }
      }
      printf("%d occurs %d times\n",a[i],count);
   }
   return 0;
}
void selectionsort(int a[], int n)
{
   int i, j, min, temp;
   for (i = 0; i < n; i++)
   {
      min = i;
      for (j = i + 1; j < n; j++)
```

```
        {
            if (a[j] < a[min])
                min = j;
        }
        swap(&a[min], &a[i]);
    }
}
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

RESULT:

```
PS D:\C Programming> cd "d:\C Programming\C Practica
Enter number of elements: 14
Enter 14 elements:
1 2 1 4 5 2 3 1 4 5 6 2 2 3
1 occurs 3 times
2 occurs 4 times
3 occurs 2 times
4 occurs 2 times
5 occurs 2 times
6 occurs 1 times
PS D:\C Programming\C Practicals-SPIT\Experiment-5>
```

CONCLUSION:	In this experiment, we learnt how to sort and search 1D arrays using various search/sorting algorithms and perform operations on elements of an array.