ASSIGNMENT 1

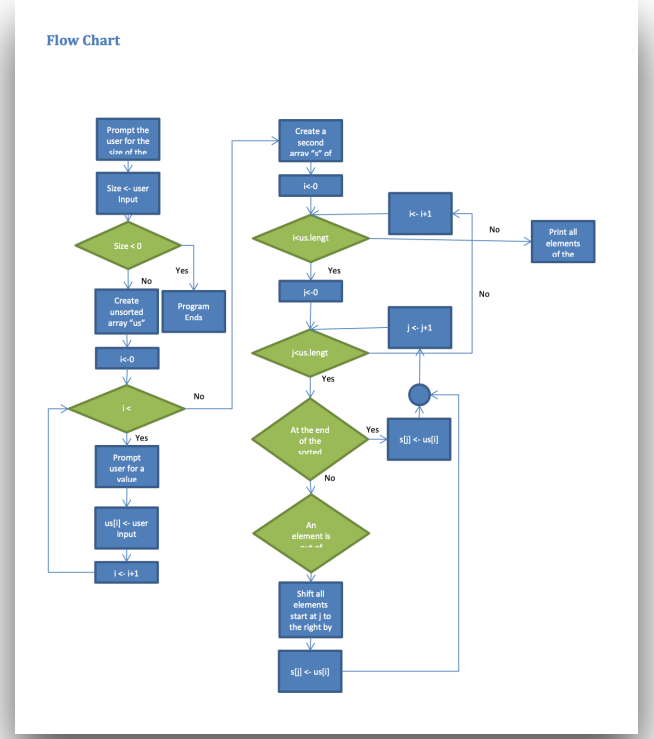
NAME :- KEVAL PANWALA
ROLL NO :- 21BCP041
GROUP :- G2

— — INSERTION SORT — —

○ AIM -

To write a program in which a user populated an unsorted array of integers and then it was sorted using insertion sort. Finally, the program printed out the sorted array to the console.

○ INTRODUCTION -



○ ALGORITHM -

STEP 1 - Prompt the user for the size of the array.
STEP 2 - Create an array of integers of size given in step 1.
STEP 3 - For each element in the array.
    A. Prompt the user to input a value.
    B. Store that value as that element of the array.
STEP 4 - For each element in the array, find a value that is less than the another value.

STEP 5 - Insert the value.
STEP 6 - Repeat until list is sorted.
STEP 7 - After that print the array using for loop.

- PROGRAM -

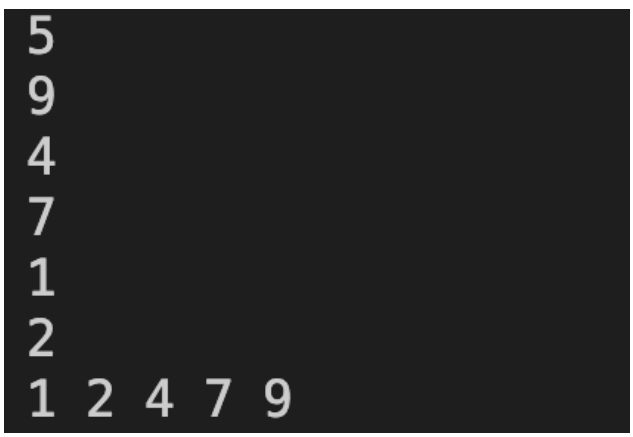
```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int main(int argc, char const *argv[])
{
    clock_t t, t1, t2;
    srand(time(0));
    int n;
    scanf("%d",&n);

    int arr[n];
    for (int i = 0; i < n; i++)
    {
        arr[i]=rand();
    }
    t = clock();
    for (int i = 1; i < n; i++)
    {
        int current = arr[i];
        int j = i - 1;
        while (arr[j]>current && j>=0)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = current;
    }
    t = clock() - t;
    double timet = ((double)t)/CLOCKS_PER_SEC;
    for (int i = 0; i < n; i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
    printf("Average time is %f\n", timet);
```

```
    t1 = clock();
    for (int i = 1; i < n; i++)
    {
        int current = arr[i];
        int j = i - 1;
        while (arr[j]>current && j>=0)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = current;
    }
    t1 = clock() - t1;
    double timet_1 = ((double)t1)/CLOCKS_PER_SEC;
    printf("Best time is %f\n", timet_1);

    t2 = clock();
    for (int i = 1; i < n; i++)
    {
        int current = arr[i];
        int j = i - 1;
        while (arr[j]>current && j>=0)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = current;
    }
    t2 = clock() - t2;
    double timet_2 = ((double)t2)/CLOCKS_PER_SEC;
    printf("Worst time is %f\n", timet_2);
    return 0;
}
```

- OUTPUT -



- ANALYSIS -

N	Best Case	Average Case	Worst Case
10	Time: 0.000001	Time: 0.000008	Time: 0.000002
100	Time: 0.000002	Time: 0.000033	Time: 0.000004
1000	Time: 0.000008	Time: 0.001681	Time: 0.000012
10000	Time: 0.000031	Time: 0.067611	Time: 0.000034
100000	Time: 0.000351	Time: 5.209319	Time: 0.000364

Worst Case Time Complexity -

Suppose, an array is in ascending order, and you want to sort it in descending order. In this case, worst case complexity occurs.

Each element has to be compared with each of the other elements so, for every nth element, `(n−1)` number of comparisons are made.

Thus, the total number of comparisons = `n∗(n−1) ~ n2.`

Best Case Time Complexity -

When the array is already sorted, the outer loop runs for `n` number of times whereas the inner loop does not run at all. So, there are only `n` number of comparisons. Thus, complexity is linear.

Average Case Time Complexity -

It occurs when the elements of an array are in jumbled order (neither ascending nor descending).

Space Complexity -

Space complexity is `0(1)` because an extra variable `key` is used.

- APPLICATIONS -

  The insertion sort is used when:
  - The array is has a small number of elements.
  - There are only few elements left to be sorted.

- REFERENCES -

  https://www.programiz.com/dsa/insertion-sort