IT 161_Lab11

Name: Dipean Dasgupta **Date: 23/03/2022**

STD ID: 202151188

Experiment 1: C program to Selection Sort algorithm with different sets of numbers (including negative numbers and zeros.

Software: Online compiler and debugger for C.

Algorithm:

Step 1: Start

Step 2: Define an array of size n after inputting n

from the user.

Step 3: Run a for loop to add elements into the

array.

Step 4: Set the first element of the array as minimum.

Step 5: Compare the minimum with the next element, if it is smaller than minimum assignthis element as minimum. Do this till the end of the array.

Step 6: Place the minimum at the first position (index 0) of the array.

Step 7: for the next iteration, start sorting from the first unsorted element i.e. the elementnext to where the minimum is placed.

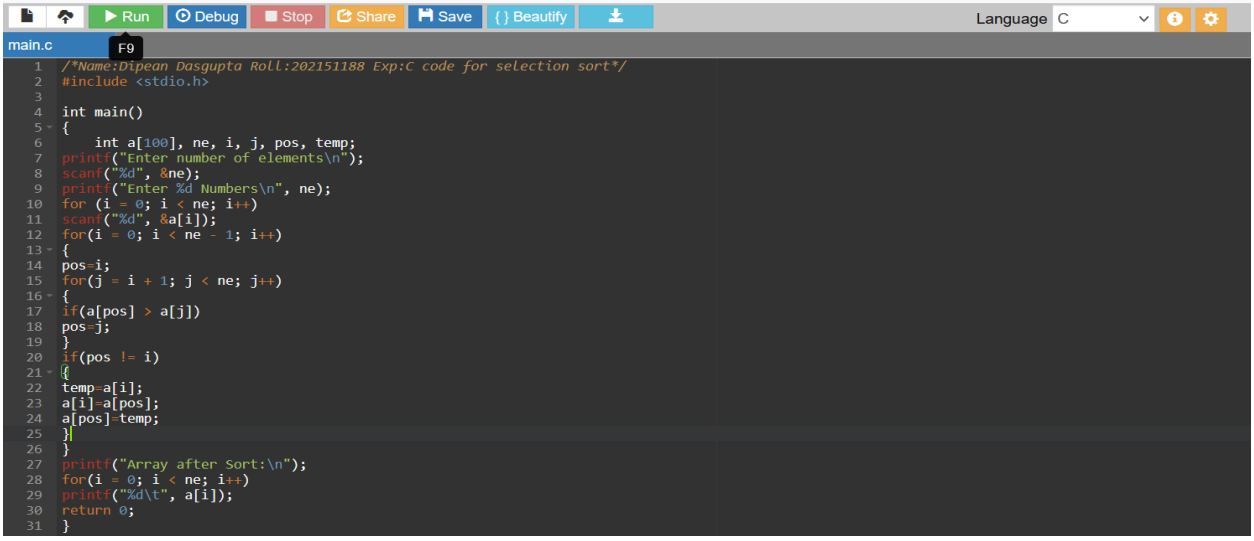
Step 8: To print the sorted array use

a for loop.

Step 9: End.

CODE:

```
#include <stdio.h>
int main()
{
   int a[100], ne, i, j, pos, temp;
printf("Enter number of elements\n");
scanf("%d", &ne);
printf("Enter %d Numbers\n", ne);
for (i = 0; i < ne; i++)
scanf("%d", &a[i]);
for(i = 0; i < ne - 1; i++)
{
pos=i;
for(j = i + 1; j < ne; j++)
{
if(a[pos] > a[j])
pos=j;
}
if(pos != i)
{
temp=a[i];
a[i]=a[pos];
a[pos]=temp;
}
}
printf("Array after Sort:\n");
for(i = 0; i < ne; i++)
printf("%d\t", a[i]);
return 0;
}
```



RESULT:

```
Enter number of elements
5
Enter 5 Numbers
23 78 56 34 12
Sorted Array:
12      23      34      56      78

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter number of elements
5
Enter 5 Numbers
-5 -27 -14 -23 -46
Array after Sort:
-46     -27     -23     -14     -5

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter number of elements
5
Enter 5 Numbers
45 0 76 -23 89
Array after Sort:
-23     0       45      76      89

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment 2: C program to Selection Sort algorithm with different sets of numbers (including negative numbers and zeros.

Software: Online compiler and debugger for C.

Algorithm:

Step 1: Start

Step 2: Define an array of size n after

inputting n from the user

Step 3: Run a for loop to add elements into the

array

Step 4: Using nested for loop and the if statement compare the elements from the 1st index.

Step 5: If the first element is greater than the second element, they are swapped

Step 6: Now we compare the second and third element and will swap them if necessary.

Step 7: The above process is carried until the last element

Step 8: To print the sorted array

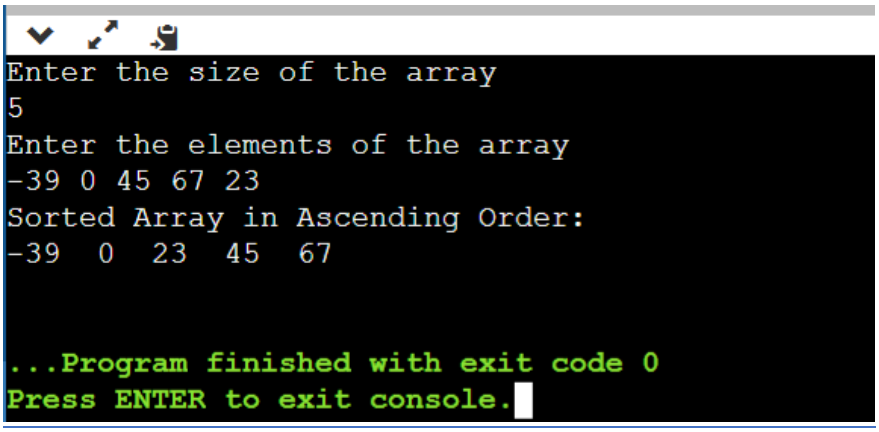
use a for loop

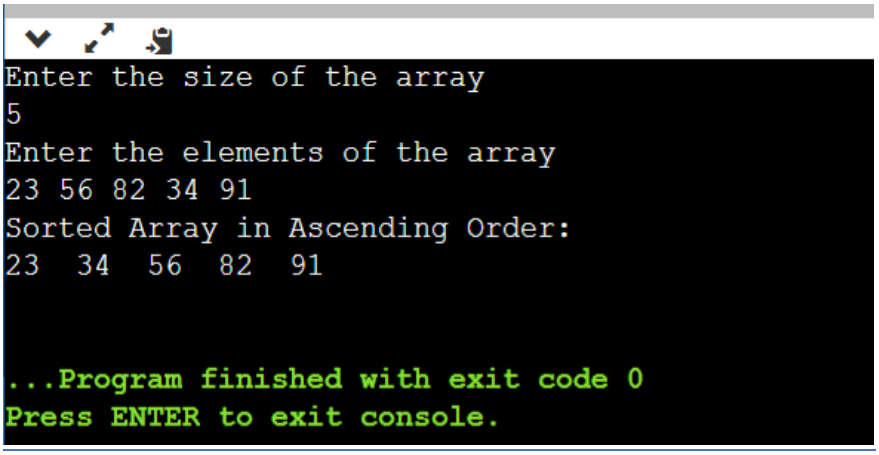
Step 9: End

CODE:

```
#include <stdio.h>
void sortbubble(int array[], int
size) {
   for (int s = 0; s < size - 1; ++s) {
   for (int i = 0; i < size - s - 1; ++i) {
    if (array[i] > array[i + 1]) {
      int swap = array[i];
      array[i] = array[i + 1];
      array[i + 1] = swap;
        }
     }
  }
}
void arrayprint(int array[], int
size) {
 for (int i = 0; i < size; ++i) {
  printf("%d  ", array[i]);
 }  printf("\n");
}
int main() {
  int size;
  printf("Enter the size of the array
\n");
  scanf("%d",&size);
  int data[size];
  int i;
  printf("Enter the elements of the
array \n");
  for(i=0;i<size;i++)
  scanf("%d",&data[i]);
  sortbubble(data, size);
  printf("Sorted Array in
Ascending Order:\n");
  arrayprint(data, size);
}
```

RESULT:


Enter the size of the array
5
Enter the elements of the array
-39 0 45 67 23
Sorted Array in Ascending Order:
-39 0 23 45 67
...Program finished with exit code 0
Press ENTER to exit console.


Enter the size of the array
5
Enter the elements of the array
23 56 82 34 91
Sorted Array in Ascending Order:
23 34 56 82 91
...Program finished with exit code 0
Press ENTER to exit console.

Experiment 3: Compare the computational complexity of the above two algorithms andmake a brief note about it, to the extent you understand it, in your report.

Objective – To compare the computational complexity of selection andbubble sort.

The efficiency of an algorithm depends on two parameters:

1. Time Complexity
2. Space Complexity

Time Complexity: Time Complexity is defined as the number of times a particular instruction set is executed rather than the total time is taken. It isbecause the total time taken also depends on some external factors like thecompiler used, processor's speed, etc.

Space Complexity: Space Complexity is the total memory space required bythe program for its execution.

Both are calculated as the function of input size(n).

1)Selection sort –

Time complexity –

- **Best Case Complexity** occurs when there is no need for sorting, i.e., thearray has already been sorted. The time complexity of selection sort in the best-case scenario is $O(n^2)$.
- **Average Case Complexity** occurs when the array elements are arrangedin a jumbled order that is neither ascending nor descending correctly. The selection sort has an average case time complexity of $O(n^2)$.
- **Worst-case complexity** - Worst case occurs when array elements must be sorted in reverse order. Assume you need to sort the array elements in ascending order, but they are in

descending order. Selection sort has aworst-case time complexity of $O(n^2)$.

Space complexity –

- It performs all computations in the original array and does not use anyother arrays.
- As a result, the space complexity is O (1).

2) Bubble sort –

Time complexity –

- **Worst Case Complexity**: If the array elements are in descending order and we want to make it in ascending order, it is the worst case. The time complexity for the worst case is $O(n^2)$.
- **Best Case Complexity**: The best case is when all the elements are already sorted, and no swapping is required. The time complexity in this scenario is O(n).
- **Average Case Complexity**: This is the case when the elements are jumbled. The time complexity for the average case in bubble sort is $O(n^2)$.

Space complexity –

Space complexity for the standard bubble sort algorithm is O(1) as there is one additional variable required to hold the swapped elements temporarily.