Priyansh Salian
2003148
C 31

Page :
Date :

Experiment No - 1(A)

Aim :- To implement Selection Sort

Theory :-

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

i. The subarray which is already sorted.
ii. Remaining subarray which is sorted

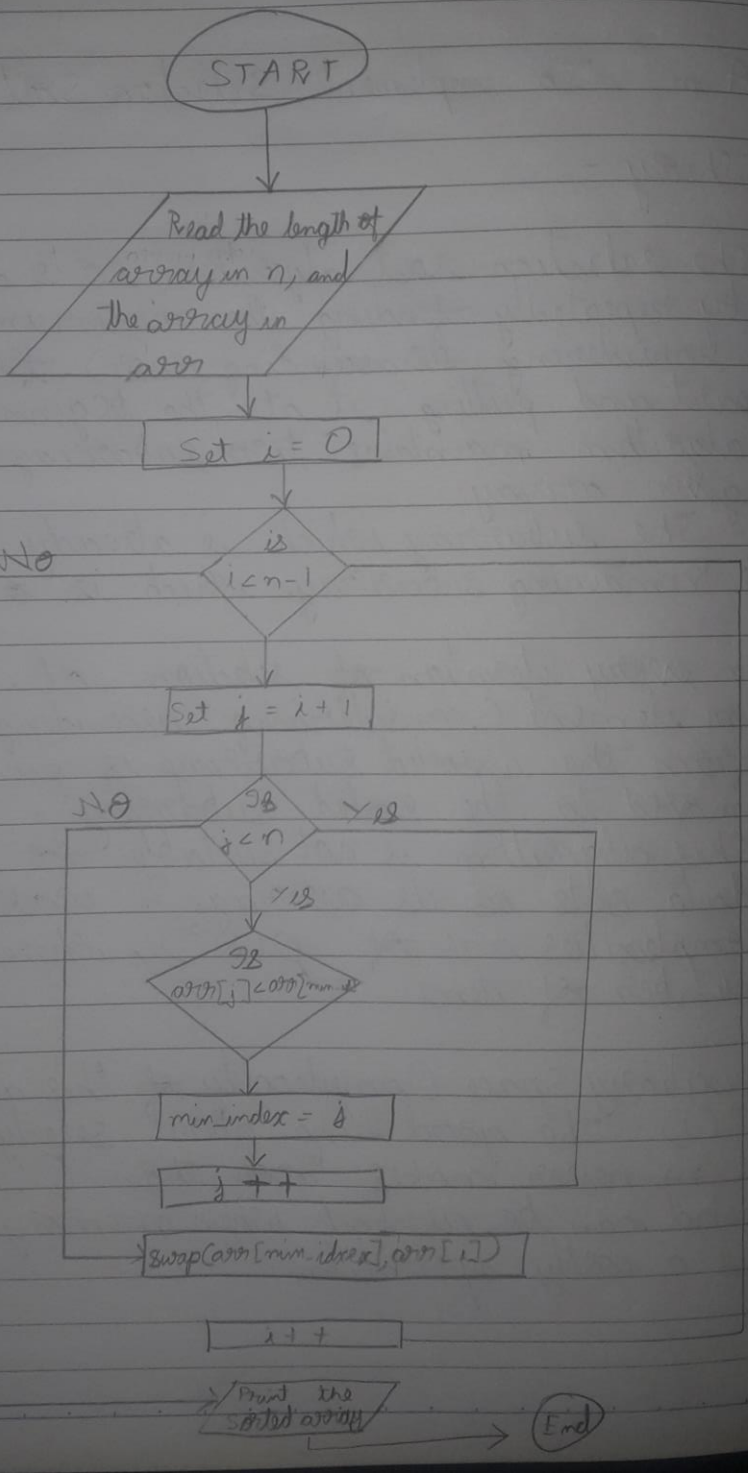
In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.
This algorithm is not suitable for large data sets as its average & worst case complexities are of $O(N^2)$, where N is the number of items.

Auxiliary Space Complexity of this algorithm is $O(1)$. The good thing about selection sort is it never makes more than $O(n)$ swaps and can be useful when memory write is a costly operation.

Priyansh Salian
200148
C 3'

Page :
Date :

Flow Chart



Priyanshi Sahai
2003148
C31

Page :
Date :

Algorithm

Step 1:- Select the first element as minimum

Step 2: Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.

Compare the minimum with the third element. Again, if the third element is smaller then assign the minimum to the third element otherwise do nothing. The process goes on until the last element.

Step 3:- After each iteration, minimum is placed in front of the unsorted list

Step 4:- For each iteration, indexing start from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

Example :-

arr[] = 64 25 12 22 11

// Finding the minimum element in arr[0....4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1....4]
// and place it at beginning of arr[1....4]

Priyansh Salian
200148
C31

11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3....4]
// and place it at the beginning of arr[3...4]

11 12 22 25 64

Analysis

Let's find the time required to execute each line

n ← length[A]	C_1	1
for (i = 1 to A.length - 1)	C_2	n
{		
min = i	C_3	$n-1$
for (j = i+1 to A.length - 1)	C_4	$n+n-1+n-2+\dots = \frac{n(n-1)}{2}$
{		
if (A[j] < A[min])	C_5	$\sum_{j=1}^{n-1}(n-j)$
{		
min = j	C_6	$\sum_{j=1}^{n-1}(n-j)$
}		
temp = A[i]	C_7	$n-1$
A[i] = A[min]		
A[min] = temp		
}		
}		

Total Time $C_1 + C_2 n + C_3(n-1) + C_4\left(\frac{n(n-1)}{2}\right) + C_5 \sum_{j=1}^{n-1}(n-j) + C_6 \sum_{j=2}^{n-1}(n-j) + C_7(n-1)$

Priyansh Sahian
2003148
C31

Page :
Date :

$= an^2 + bn + c$
$= O(n^2)$

Best Case :- Here the array will be already sorted so program will travers the array ~~two~~ in ~~$O(n^2)$~~ n^2 times. So time complexity will be $O(n^2)$

Average Case :- In this the existing elements are in jumbled order, i.e neither in ascending order nor in descending order. So time complexity will be $O(n^2)$ ~~for~~ traversing the array ~~in for~~ n^2 times ~~some time~~ plus some time to swap elements.

Worst case :- ~~Here~~ This will take place when array will be in descending order & we want to arrange it in ascending order. So here the time complexity will be $O(n^2)$ ~~for~~ traversing the array + ~~O~~ plus $O(n)$ ~~for~~ swapping the elements. Overall time complexity is $O(n^2)$

Space Complexity

Space Complexity of selection sort will be $O(1)$ as we ~~don't~~ only require one variable ~~in~~ this whole process ~~and~~ and so this algorithm does not depend on ~~two~~ size of array so it will be executed in constant time.

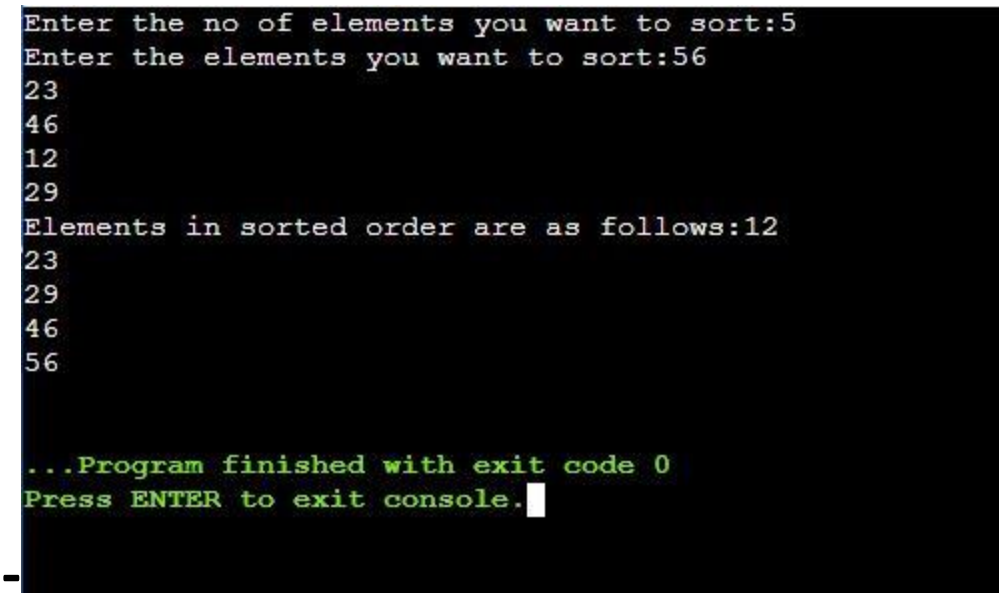
CODE:-

ex_1_a.cpp > main()

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a;
    cout << "Enter the no of elements you want to sort:-";
    cin >> a;
    const int n = a;
    int arr[n];
    cout << "Enter the elements you want to sort:-";

    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[i])
```

```cpp
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[i])
            {
                int temp = arr[j];
                arr[j] = arr[i];
                arr[i] = temp;
            }
        }
    }
    cout << "Elements in sorted order are as follows:-";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i];
        cout << "\n";
    }
}
```

OUTPUT:-


Enter the no of elements you want to sort:5
Enter the elements you want to sort:56
23
46
12
29
Elements in sorted order are as follows:12
23
29
46
56
...Program finished with exit code 0
Press ENTER to exit console.

Priyansh Salian
2003148
C31

Page :
Date :

Application

It is used when
1. A small array is to be sorted
2. Swapping cost does not matter
3. It is compulsory to check all elements

Conclusion

Selection sort is unstable algorithm that is good for sorting small datasets. Time taken by this algorithm is more but it is beneficial to scenarios where we have memory limitations. It does not stop unless the no number of iterations have been achieved even though the list is already sorted. In it we compare Bubble sort with Selection sort we can find that only space complexity is reduced but the time complexity is reduced.