**WORKSHEET 1.3**

**1. Aim:**

Write a program to implement Selection sort along with its complexity analysis.

**2. Problem Description:**

The aim of this experiment is to understand the Selection Sort algorithm, its time and space complexity, and how it compares against other sorting algorithms.

**3. 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

**4. Computational Complexity:-**

* To complete one iteration, we traverse a part of the array (from index i to the end) exactly once (while keeping track of the smallest element encountered so far). Since the longest length we ever traverse in any given iteration is N (in the first iteration when i=1 -> from first to last element), time complexity of completing one iteration is O(N).
* In Selection Sort, we run N iterations, each of which takes O(N) time. Hence overall time complexity becomes O(N\*N).

**5. Pseudo Code :-**

procedure selection sort

list : array of items

n : size of list

for i =1 to n -1

/\* set current element as minimum\*/

min = i

/\* check the element to be minimum \*/

for j = i+1 to n

if list[j]< list[min]then

min = j;

endif

endfor

/\* swap the minimum element with the current element\*/

if indexMin != i then

swap list[min]and list[i]

endif

endfor

end procedure

**6. Source Code:**

#include <iostream>

using namespace std;

int main()

{

int n, i, j, min, temp;

cout<<"Enter the size of array:\n";

cin>>n;

int a[n];

cout<<"Now enter the array elements:\n";

for(i=0;i<n;i++)

{

cin>>a[i];

}

for(i=0;i<n;i++)

{

min=a[i];

for(j=i;j<n;j++)

{

if(a[j]<min)

{

temp=min;

min=a[j];

a[j]=temp;

}

}

a[i]=min;

}

cout<<"Sorted array is:\n";

for(i=0;i<n;i++)

{

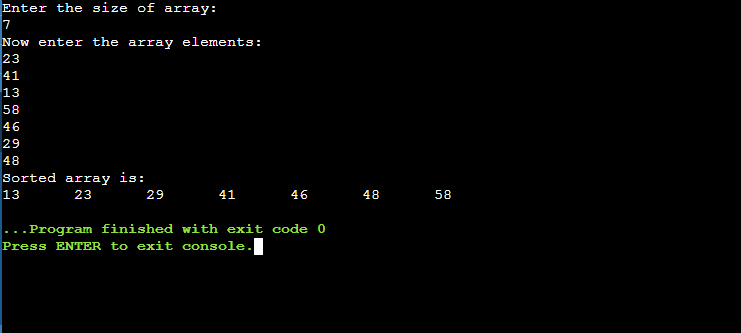
cout<<a[i]<<"\t";

}

return 0;

}

1. **Screenshot of Output:**



1. **Difference b/w Selection Sort and Bubble Sort**

* **Complexity :** The best-case complexity is O(N). It happens when we have an already sorted array, as in that case, we will stop the procedure after a single pass. While The best-case complexity is O(N2) as to find the minimum element at every iteration, we will have to traverse the entire unsorted array.
* **Stability :** Bubble sort is stable, whereas selection sort isn’t.
* **Works well with :** Selection sort works similarly on all kinds of dataset, while Bubble Sort is **adaptive** which means they perform a lesser number of operations if a partially sorted array is provided as input.
* **Applications :** Selection sort can be used to sort a linked list as we can efficiently remove the smallest element and append it in the sorted list. Where as Bubble sortcan detect minor errors in a sorted dataset and sort an almost sorted dataset quickly.
* **Advantage over other :** Selection sort is efficient, where swapping operation is costly as it makes a maximum of N swaps for an array of size N, where as Bubble sort is the simplest stable in-place sorting algorithm and very easy to code.

1. **Learning & Outcomes:**

* Given an unsorted array of numbers, generate a sorted array of numbers by applying Selection Sort
* Demonstrate knowledge of time complexity of Selection Sort by counting the number of operations involved in each iteration.
* Compared Selection Sort with other sorting algorithms and realise Selection Sort as a stable comparison sorting algorithm.
* Also compared Selection Sort to the prior sorting algorithms, (Bubble Sort) to find which sorting algorithm is best and in which case.