Experiment No.2
Selection Sort
Date of Performance:
Date of Submission:

Experiment No. 2

Title: Selection Sort

Aim: To implement Selection Comparative analysis for large values of ‘n’

Objective: To introduce the methods of designing and analyzing algorithms

Theory:

Selection sort is a sorting algorithm, specifically an in-place comparison sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

The algorithm divides the input list into two parts: the sub list of items already sorted, which is built up from left to right at the front (left) of the list, and the sub list of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sub list is empty and the unsorted sub list is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sub list, exchanging it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

Example:

arr[] = 64 25 12 22 11

// Find 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]

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 beginning of arr[3...4]

11 12 22 **25** 64

Algorithm and Complexity:

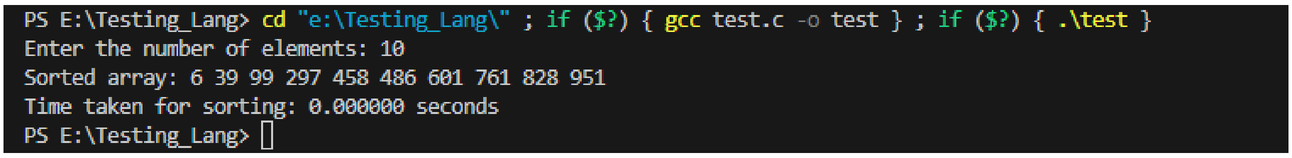
Alg.: SELECTION-SORT(A)		
	cost	Times
n ← length[A]	c_1	1
for j ← 1 to n - 1	c_2	n-1
do smallest ← j	c_3	n-1
for i ← j + 1 to n	c_4	$\sum_{j=1}^{n-1}(n-j+1)$
≈ n2/2 comparisons, do if A[i]<A[smallest]	c_5	$\sum_{j=1}^{n-1}(n-j)$
then smallest ← i	c_6	$\sum_{j=1}^{n-1}(n-j)$
≈ n exchanges, exchange A[j] ↔ A[smallest]	c_7	n-1

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to perform selection sort
void selectionSort(int arr[], int n) {
   int i, j, min_idx;
   for (i = 0; i < n-1; i++) {
      min_idx = i;
      for (j = i+1; j < n; j++) {
         if (arr[j] < arr[min_idx]) {
            min_idx = j;
         }
      }
      // Swap the found minimum element with the first element
      int temp = arr[min_idx];
      arr[min_idx] = arr[i];
      arr[i] = temp;
   }
}

// Function to generate random array of given size
```

```
void generateRandomArray(int arr[], int n) {
   srand(time(NULL));
   for (int i = 0; i < n; i++) {
      arr[i] = rand() % 1000; // Generating random numbers between 0 and 999
   }
}

int main() {
   int n, i;

   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int arr[n];

   // Generating random array
   generateRandomArray(arr, n);

   // Sorting the array using Selection Sort
   clock_t start = clock();
   selectionSort(arr, n);
   clock_t end = clock();
   double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

   // Displaying sorted array
   printf("Sorted array: ");
   for (i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");

   // Displaying time taken for sorting
   printf("Time taken for sorting: %f seconds\n", time_taken);

   return 0;
}
```

Output :



```
PS E:\Testing_Lang> cd "e:\Testing_Lang\" ; if ($?) { gcc test.c -o test } ; if ($?) { .\test }
Enter the number of elements: 10
Sorted array: 6 39 99 297 458 486 601 761 828 951
Time taken for sorting: 0.000000 seconds
PS E:\Testing_Lang> 
```

Conclusion:

This program prompts the user to enter the number of elements 'n'. It then generates a random array of 'n' elements, sorts the array using the Selection Sort algorithm, and displays the sorted array along with the time taken for sorting.

To perform a comparative analysis for large values of 'n', you can modify the program to execute multiple times with increasing values of 'n' and measure the time taken for each execution. Additionally, you can compare the performance of Selection Sort with other sorting algorithms to analyze their efficiency for different input sizes.