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

Implementation:

Code:

```
#include<stdio.h>
#include<conio.h>

int main() {
 int a[] ={5,2,1,4,3};
 int n=5,i,min,j;

 for(i=0;i<n-1;i++)
 {
  min=i;
  for(j=i+1;j<n;j++)
  {
   if (a[j]<a[min])
   {
    min=j;
   }
  }
```

```
  if(min!=i)
  {
   int temp=a[i];
   a[i]=a[min];
   a[min]=temp;
  }
 }

 printf("Sorted array:");
 for(i=0;i<n;i++)
 {
  printf("%d ",a[i]);
 }
 getch();
}
```

Output:

Conclusion: In conclusion, the selection sort algorithm effectively sorts an array of integers in ascending order. This sorting method divides the array into two subarrays: sorted and unsorted. Initially, the sorted subarray is empty, and the unsorted subarray contains all the elements of the original array. The algorithm repeatedly selects the smallest element from the unsorted subarray and swaps it with the element at the beginning of the unsorted subarray. This process continues until the entire array is sorted. Selection sort has a time complexity of O(n^2) in the worst case scenario, making it suitable for small to medium-sized arrays. However, it exhibits poor performance on large arrays compared to more efficient sorting algorithms like merge sort or quick sort. Additionally, selection sort is an in-place sorting algorithm, meaning it doesn't require additional memory space for sorting.