Experiment No.1
To implement Selection Sort
Date of Performance: 08/02/2024
Date of Submission: 15/02/2024

Experiment No. 1

Title: To implement selection sort.

Aim: To study, implement and Analyze Selection Sort Algorithm

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 algorithmsin certain situations, particularly where auxiliary memory is limited.

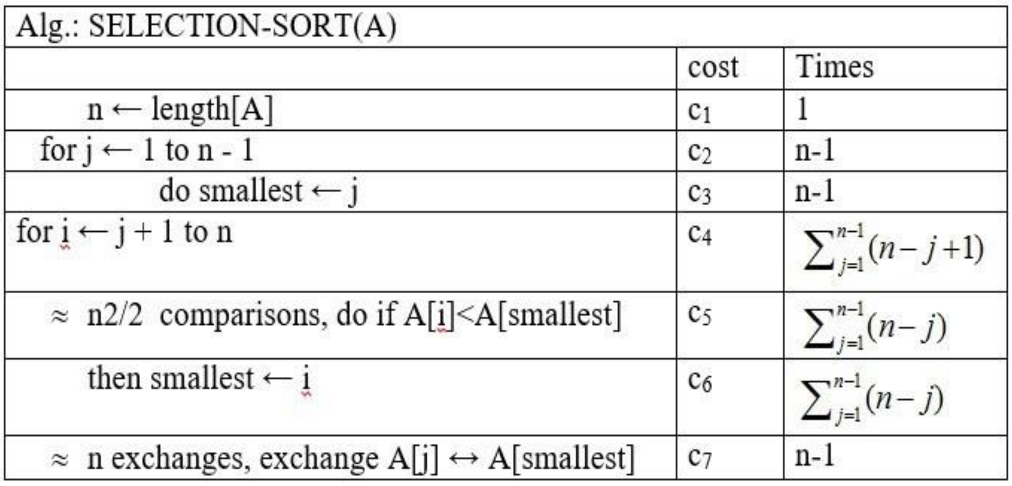
The algorithm divides the input list into two parts: the sub list of items already sorted, which is built up from left to right at the front (left) of the list, and the sub list of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sub list is empty and the unsorted 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 sub-list boundaries one element to the right.

Example:

Sort the given array using selection sort. arr[] = 64 25 12 22 11

11 25 12 22 64	Find the minimum element in arr[0...4] and place it at beginning
11 12 25 22 64	Find the minimum element in arr[1...4] and place it at beginning of arr[1...4]
11 12 **22** 25 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]

Algorithm and Complexity:



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

The recurrence relation for selection sort is:

T(n) = 1 for n=0

= T(n – 1) + n for n>0 ---- 1

From above equation,

T (n – 1) = T(n – 2) + (n – 1)

Use above in equation 1

T(n) = T(n – 2) + (n – 1) + n----2

Let T(n – 2) = T(n – 3) + n – 2

Use above in equation 2

T(n) = T(n – 3) + (n – 2) + (n – 1) + n

After k iterations,

T(n) = T(n – k) + (n – k + 1) + (n – k + 2) + ..… + (n – 1) + n

When k approaches to n,

T(n) = T(0) + 1 + 2 + 3 + … + (n –1) + n

T(0) = 0,

T(n) = 1 + 2 + 3 + … + n

= n(n + 1) / 2

= $(n^2 /2) + (n/2)$

T(n) = O(max($(n^2 /2) + (n/2)$))

= $O(n^2 / 2)$

= $O(n^2)$

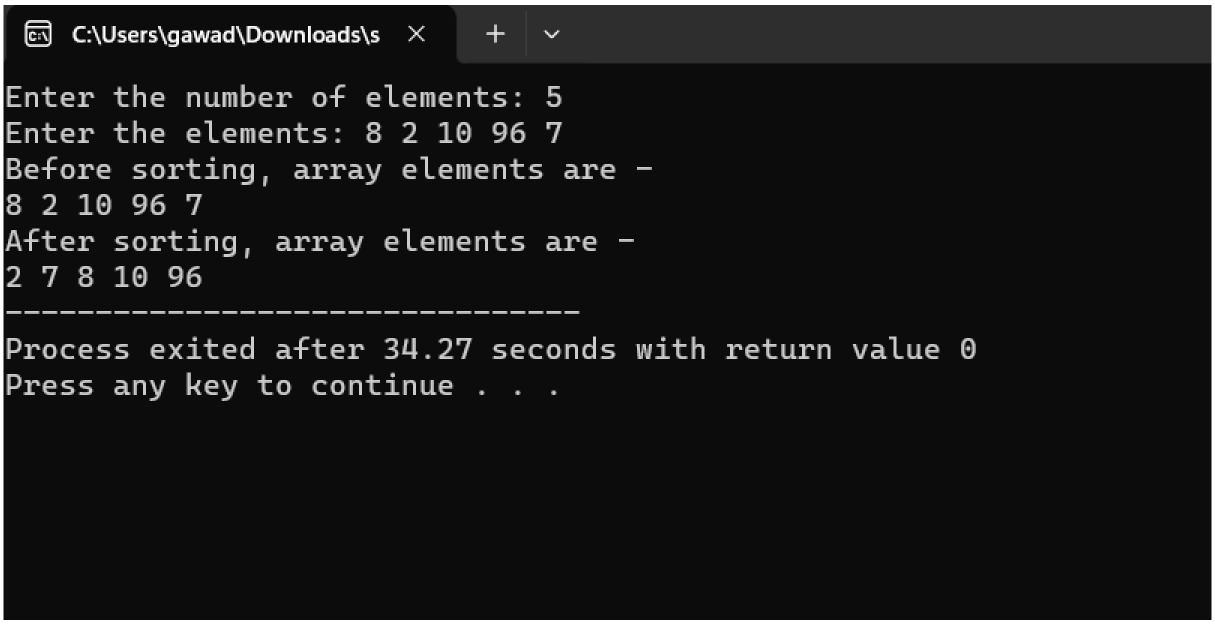
T(n) = $O(n^2)$

Code:

```
#include <stdio.h>

#include <conio.h>

#include <math.h>


// Function for selection sort

void selection_sort(int arr[], int n)

{

    int i, j, small;


    for (i = 0; i < n-1; i++)

    {

        small = i;


        for (j = i+1; j < n; j++)

            if (arr[j] < arr[small])

                small = j;


        int temp = arr[small];

        arr[small] = arr[i];

        arr[i] = temp;

    }

}
```

```
// Function for print array
void print_Array(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}

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

    int a[n];
    printf("Enter the elements: ");
  for (int i=0;i<n;i++){

        scanf("%d", &a[i]);
}
    printf("Before sorting, array elements are - \n");
    // Calling print array function
    print_Array(a, n);

    // Calling selection sort algorithm function
    selection_sort(a, n);


    printf("\nAfter sorting, array elements are - \n");

    print_Array(a, n);


    return 0;

}
```

Output:



```
Enter the number of elements: 5
Enter the elements: 8 2 10 96 7
Before sorting, array elements are -
8 2 10 96 7
After sorting, array elements are -
2 7 8 10 96
--------------------------------
Process exited after 34.27 seconds with return value 0
Press any key to continue . . .
```

Conclusion:

In summary, selection sort is a simple algorithm to understand and implement. However, its time complexity of O(n^2) means it's not the most efficient choice for sorting large datasets. While it's suitable for small lists or educational purposes, for real-world applications, algorithms like merge sort or quicksort are preferable due to their faster performance.