|  |
| --- |
| **Experiment No.1** |
| **To implement Selection Sort** |
| Date of Performance: 8/02/2024 |
| 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 algorithms in certain situations, particularly where auxiliary memory is limited.

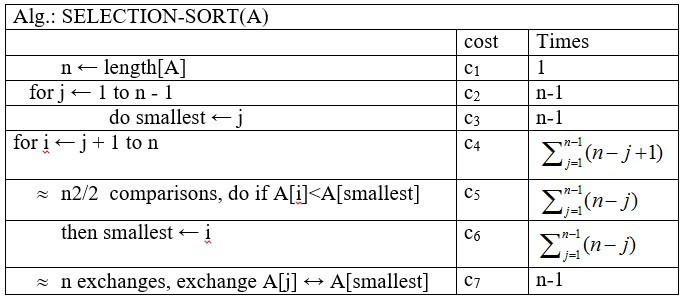
The algorithm divides the input list into two parts: the sub list of items already sorted, which is built up from left to right at the front (left) of the list, and the sub list of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sub list is empty and the unsorted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



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

= (n2 /2) + (n/2)

T(n) = O(max( (n2 /2) + (n/2) ))

= O(n2 / 2)

= O(n2)

T(n) = O(n2)

**Code:**

#include <stdio.h>

void selectionSort(int arr[], int n) {

int i, j, minIndex, temp;

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

minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

int main() {

int arr[100], n, i;

printf("Enter the number of elements: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

scanf("%d", &arr[i]);

}

selectionSort(arr, n);

printf("Sorted array: ");

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

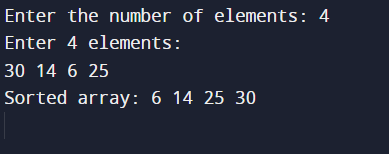
printf("%d ", arr[i]);

}

return 0;

}

**Output:**

****

**Conclusion:** The provided C implementation showcases Selection Sort's simplicity and effectiveness in sorting arrays. Through user input, it demonstrates the algorithm's accuracy in sorting elements in non-decreasing order. However, Selection Sort's time complexity of O(n^2) makes it less efficient for large datasets, emphasizing the need for more efficient algorithms in such cases. Overall, the experiment provides a practical understanding of Selection Sort's mechanics and its performance characteristics**.**