DATE:01/03/2023 Roll Number: A422

EXPERIMENT NO: 1

**Aim:**To study and implement selection sort.

**Theory:**

SELECTION SORT-

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted. One variation of selection sort is called “Bidirectional selection sort” that goes through the list of elements by alternating between the smallest and largest element, this way the algorithm can be faster in some cases.

The algorithm maintains two subarrays in a given array.

- The subarray which already sorted.

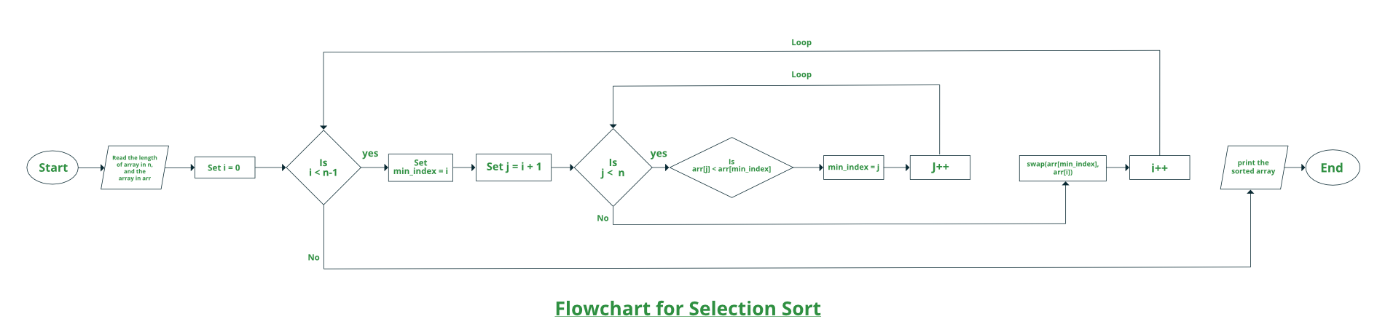
- The remaining subarray was unsorted.

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the beginning of unsorted subarray.

After every iteration sorted subarray size increase by one and unsorted subarray size decrease by one.

After N (size of array) iteration we will get sorted array.

Flowchart of the Selection Sort:



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

**How selection sort works?**

Lets consider the following array as an example: arr[] = {64, 25, 12, 22, 11}

First pass:

For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 64 | 25 | 12 | 22 | 11 |

Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 25 | 12 | 22 | 64 |

Second Pass:

For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 25 | 12 | 22 | 64 |

After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | 25 | 22 | 64 |

Third Pass:

Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | 25 | 22 | 64 |

While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | 22 | 25 | 64 |

Fourth pass:

Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array

As 25 is the 4th lowest value hence, it will place at the fourth position.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | 22 | 25 | 64 |

Fifth Pass:

At last the largest value present in the array automatically get placed at the last position in the array

The resulted array is the sorted array.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | 22 | 25 | 64 |

Follow the below steps to solve the problem:

* Initialize minimum value(min\_idx) to location 0.
* Traverse the array to find the minimum element in the array.
* While traversing if any element smaller than min\_idx is found then swap both the values.
* Then, increment min\_idx to point to the next element.
* Repeat until the array is sorted.

**Source code:**

// C++ program for implementation of

// selection sort

#include <bits/stdc++.h>

using namespace std;

//Swap function

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of

// unsorted subarray

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

{

// Find the minimum element in

// unsorted array

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

if (min\_idx!=i)

swap(&arr[min\_idx], &arr[i]);

}

}

//Function to print an array

void printArray(int arr[], int size)

{

int i;

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

{

cout << arr[i] << " ";

cout << endl;

}

}

// Driver program to test above functions

int main()

{

int arr[] = {64, 25, 12, 22, 11};

int n = sizeof(arr)/sizeof(arr[0]);

selectionSort(arr, n);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

Output

Sorted array:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 64 | 25 | 12 | 22 | 11 |
| 11 | 25 | 12 | 22 | 64 |
| 11 | 12 | 25 | 22 | 64 |
| 11 | 12 | 22 | 25 | 64 |
| 11 | 12 | 22 | 25 | 64 |

**Complexity Analysis of Selection Sort:**

**Time Complexity:** The time complexity of Selection Sort is O(N2) as there are two nested loops:

One loop to select an element of Array one by one = O(N)

Another loop to compare that element with every other Array element = O(N)

Therefore overall complexity = O(N) \* O(N) = O(N\*N) = O(N2)

Auxiliary Space: O(1) as the only extra memory used is for temporary variables while swapping two values in Array. The selection sort never makes more than O(N) swaps and can be useful when memory write is a costly operation.

**Conclusion:**

In this experiment we learn about Selection sort its application and analyse the time and space complexities for the same.