Name: Umang Bhalla **Date: 05 Aug 2021** **SchNo: 20U03031**

Experiment no 1

Selection Sort

(i) Explain the following sorting algorithm

Consider the following elements are to be sorted in ascending order using selection sort-

6, 2, 11, 7, 5

Selection sort works as-

- It finds the first smallest element (2).
- It swaps it with the first element of the unordered list.
- It finds the second smallest element (5).
- It swaps it with the second element of the unordered list.
- Similarly, it continues to sort the given elements.

(ii) Write down algorithm

Consider the following elements are to be sorted in ascending order-

6, 2, 11, 7, 5

The selection sort algorithm works as illustrated below-

Step-01: For i = 0

[6, 2, 11, 7, 5]

we find minimum element of array and swap it with 1st element of array

[2, 6, 11, 7, 5]

Step-02: For i = 1

[2, 6, 11, 7, 5]

we find minimum element of sub array and swap it with 2nd element of array

[2, 5, 11, 7, 6]

Step-03: For i = 2

[2, 5, 11, 7, 6]

we find minimum element of sub array and swap it with 3nd element of array

[2, 5, 6, 7, 11]

Step-04: For i = 3

[2, 5, 6, 7, 11]

we find minimum element of sub array , but there is no need of swap

Step-05: For i = 4

Loop gets terminated as ‘i’ becomes 4.

The state of array after the loops are finished is

[2, 5, 6, 7, 11]

With each loop cycle,

- The minimum element in unsorted sub-array is selected.
- It is then placed at the correct location in the sorted sub-array until array A is completely sorted.

(iii) Evaluate Time and Space complexity

Time Complexity Analysis

Selection sort algorithm consists of two nested loops.
Owing to the two nested loops, it has O(n2) time complexity.

	Time Complexity
Best Case	n^2
Average Case	n^2
Worst Case	n^2

Space Complexity Analysis

- Selection sort is an in-place algorithm.
- It performs all computation in the original array and no other array is used.
- Hence, the space complexity works out to be O(1).

Name: Umang Bhalla **Date: 05 Aug 2021** **SchNo: 20U03031**

Experiment no 1

(iv) Write down code and attach screenshot of output

Let A be an array with n elements. Then, selection sort algorithm used for sorting is as follows-

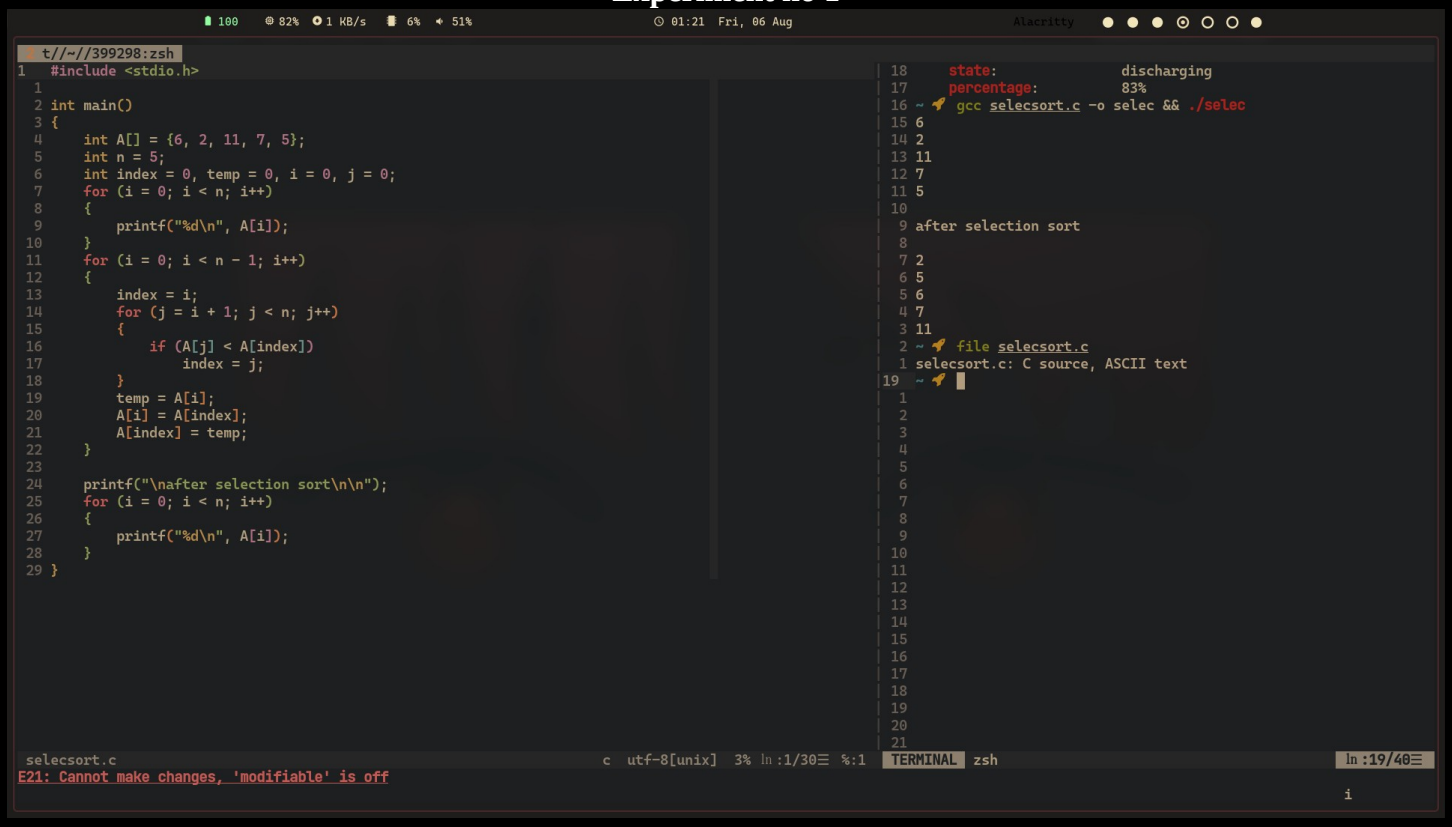
```
#include <stdio.h>

int main()
{
    int A[] = {6, 2, 11, 7, 5};
    int n = 5;
    int index = 0, temp = 0, i = 0, j = 0;
    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }
    for (i = 0; i < n - 1; i++)
    {
        index = i;
        for (j = i + 1; j < n; j++)
        {
            if (A[j] < A[index])
                index = j;
        }
        temp = A[i];
        A[i] = A[index];
        A[index] = temp;
    }

    printf("\nafter selection sort\n\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }
}
```

Here,

- i = variable to traverse the array A
- index = variable to store the index of minimum element
- j = variable to traverse the unsorted sub-array
- temp = temporary variable used for swapping

Name: Umang Bhalla **Date: 05 Aug 2021** **SchNo: 20U03031**

Experiment no 1



Name: Umang Bhalla **Date: 05 Aug 2021** **SchNo: 20U03031**
Experiment no 1

Bubble Sort

(i) Explain the following sorting algorithm

Bubble sort uses multiple passes (scans) through an array.

- In each pass, bubble sort compares the adjacent elements of the array.
- It then swaps the two elements if they are in the wrong order.
- In each pass, bubble sort places the next largest element to its proper position.
- In short, it bubbles down the largest element to its correct position.

(ii) Write down algorithm

Consider the array A-

6, 2, 11, 7, 5

Step-01:

- We have pass=1 and i=0.
- We perform the comparison A[0] > A[1] and swaps if the 0^{th} element is greater than the 1^{th} element.
- Since 6 > 2, so we swap the two elements.

  [6, 2, 11, 7, 5] ----> [2, 6, 11, 7, 5]

Step-02:

- We have pass=1 and i=1.
- We perform the comparison A[1] > A[2] and swaps if the 1^{th} element is greater than the 2^{th} element.
- Since 6 < 11, so no swapping is required.

  [2, 6, 11, 7, 5] ----> [2, 6, 11, 7, 5]

Step-03:

- We have pass=1 and i=2.
- We perform the comparison A[2] > A[3] and swaps if the 2^{nd} element is greater than the 3^{rd} element.
- Since 11 > 7, so we swap the two elements.

[2, 6, 11, 7, 5] ----> [2, 6, 7, 11, 5]

Step-04:

- We have pass=1 and i=3.
- We perform the comparison A[3] > A[4] and swaps if the 3rd element is greater than the 4th element.
- Since 11 > 5, so we swap the two elements.

[2, 6, 11, 7, 5] ----> [2, 6, 7, 5, 11]

Finally after the first pass, we see that the largest element 11 reaches its correct position.

Step-05:

- Similarly after pass=2, element 7 reaches its correct position.
- The modified array after pass=2 is shown below-

[2, 6, 11, 7, 5] ----> [2, 6, 5, 7, 11]

Step-06:

- Similarly after pass=3, element 6 reaches its correct position.
- The modified array after pass=3 is shown below-

[2, 6, 11, 7, 5] ----> [2, 5, 6, 7, 11]

Step-07:

- No further improvement is done in pass=4.
- This is because at this point, elements 2 and 5 are already present at their correct positions.
- The loop terminates after pass=4.
- Finally, the array after pass=4 is shown below-

[2, 5, 6, 7, 11]

Name: Umang Bhalla Date: 05 Aug 2021 SchNo: 20U03031
Experiment no 1

(iii) Evaluate Time and Space complexity

Time Complexity Analysis

- Bubble sort uses two loops- inner loop and outer loop.
- The inner loop deterministically performs O(n) comparisons.

Worst Case-

- In worst case, the outer loop runs O(n) times.
- Hence, the worst case time complexity of bubble sort is O(n x n) = $O(n^2)$.

Best Case-

- In best case, the array is already sorted but still to check, bubble sort performs O(n) comparisons.
- Hence, the best case time complexity of bubble sort is O(n).

Average Case-

- In average case, bubble sort may require (n/2) passes and O(n) comparisons for each pass.
- Hence, the average case time complexity of bubble sort is O(n/2 x n) = $\Theta(n^2)$.

The following table summarizes the time complexities of bubble sort in each case-

	Time Complexity
Best Case	O(n)
Average Case	$\Theta(n^2)$
Worst Case	$O(n^2)$

From here, it is clear that bubble sort is not at all efficient in terms of time complexity of its algorithm.

Name: Umang Bhalla **Date: 05 Aug 2021** **SchNo: 20U03031**

Experiment no 1

Space Complexity Analysis

- Bubble sort uses only a constant amount of extra space for variables like flag, i, n.
- Total comparisons in Bubble sort is: n (n - 1) / 2 ≈ n^2 - n
- Hence, the space complexity of bubble sort is O(1).
- It is an in-place sorting algorithm i.e. it modifies elements of the original array to sort the given array.

Name: Umang Bhalla **Date: 05 Aug 2021** **SchNo: 20U03031**

Experiment no 1

(iv) Write down code and attach screenshot of output

```
#include <stdio.h>

void swap(int x, int y, int A[])
{
    int temp = A[x];
    A[x] = A[y];
    A[y] = temp;
}
int main()
{
    int A[] = {6, 2, 11, 7, 5};
    int n = 5;
    int i = 0;

    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }
    for(int pass=1 ; pass<=n-1 ; ++pass) // Making passes through array
    {
        for(int i=0 ; i<=n-2 ; ++i)
        {
            if(A[i] > A[i+1]) // If adjacent elements are in wrong order
                swap(i,i+1,A); // Swap them
        }
    }
    //swap function : Exchange elements from array A at position x,y

    printf("\nafter selection sort\n\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", A[i]);
    }


}
```

Here,

- pass : Variable to count the number of passes that are done till now
- n : Size of the array
- i : Variable to traverse the array A
- swap() : Function to swap two numbers from the array
- x,y : Indices of the array that needs to be swapped

Name: Umang Bhalla **Date: 05 Aug 2021** **SchNo: 20U03031**

Experiment no 1