Mark Lopes

S.E Comps_A_Batch_C

9913

Rubrics for Lab Experiments

Sr. No	Performance Indicator	Excellent	Good	Average	Below Average
1.	Coding Standards [3M]	The code adheres to all standards. The code is exceptionally well organized and very easy to follow. Comments are complete and useful; variables' purposes are clearly communicated by their names. [3 marks]	There may be some minor failures to adhere to standards, for instance, indentation may be inconsistent, some lines may be too long, or a few variables may have unobvious names or be undocumented. [2 marks]	The code fails to adhere to standards at multiple locations indentation is inconsistent throughout the program, Many variable names are vague, comments are missing. [1 marks]	There are major problems with the program's design or coding style that would interfere with its comprehension, reuse, or maintenance. The code may be poorly formatted. [0 mark]
2	Output validation [2M]	Output is obtained for different test cases of input.[2M]	Output is obtained only for some subsets of input.[1M]	Output is obtained only for some subsets of input, incorrect output for few test cases[0.5M]	no output is obtained. [0 mark]
3	Post Lab Questions [3M]	Answers to all questions are correct and explained in depth. [3 marks]	Answers to most of the questions are correct but not explained in depth. [2 marks]	Few answers are incorrect [1 M]	Answers to most of the questions are incorrect. [0 mark]
4	Promptness / Preparedness [2M]	The laboratory report is submitted on time, all questions are answered. [2 mark]	The laboratory report is submitted next day, some questions answered. [1 marks]	-	The laboratory report is submitted after due date and no preparation. [0 marks]

1] Bubble sort

```c
#include <stdio.h>

// Function to perform Bubble Sort
void bubbleSort(int arr[], int n)
{
    for (int pass = 0; pass < n - 1; pass++) // iterate through all the passes
    {
        for (int i = 0; i < n - pass - 1; i++) // iterate through all
comparisons
        {

            if (arr[i] > arr[i + 1]) // if element is greater then swap
            {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
```

```
        }
    }
}

// Function to print an array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int getArraySize(int arr[])
{
    int size = 0;

    // Iterate through the array until the end marker is found
    while (arr[size] != '\0')
    {
        size++;
    }

    return size;
}


int main()
{
    int arr[] = {34,677,87,2,45};
    int n = getArraySize(arr);

    printf("Original array: ");
    printArray(arr, n);

    bubbleSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

```
=Microsoft-MIEngine-Pid-5ajfpou0.c1m  --dbgExe=C:\msys64\
Original array: 34 677 87 2 45
Sorted array: 2 34 45 87 677
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA>
```

2]Modified bubble sort

```c
#include <stdio.h>
#include <stdbool.h>

void ModifiedBubbleSort(int arr[], int n)
{
    int pass, j;
    bool exchange; // flag to check if any exchange or swaps are made between
passes

    for (pass = 0; pass < n - 1; pass++)
    {
        exchange = false; // No swaps made yet in this pass

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

            if (arr[j] > arr[j + 1])  // if element is greater then swap
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                exchange = true; // Set the flag to indicate a swap
            }
        }

        // If no swaps were made, the array is already sorted
        if (!exchange)
        {
            break;
        }
    }
}

// Function to print an array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
```

```
    printf("\n");
}

int getArraySize(int arr[])
{
    int size = 0;

    // Iterate through the array until the end marker is found
    while (arr[size] != '\0')
    {
        size++;
    }

    return size;
}


int main()
{
    int arr[] = {2,56,34,90,67};
    int n = getArraySize(arr);

    printf("Original array: ");
    printArray(arr, n);

    ModifiedBubbleSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

```
=Microsoft-MIEngine-Pid-vht5orkp.54f' '--dbgExe=C:\m
Original array: 2 56 34 90 67
Sorted array: 2 34 56 67 90
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA>
```

3] Selection sort

```
#include <stdio.h>

void selectionSort(int arr[], int n)
{
    int pass, j, minIndex;

    for (pass = 0; pass < n - 1; pass++) //iterate through every pass
    {
```

```
        minIndex = pass;

        // Find the index of the minimum element in the unsorted part of the
array
        for (j = pass + 1; j < n; j++)
        {
            if (arr[j] < arr[minIndex])
            {
                minIndex = j;
            }
        }

        // Swap the found minimum element with the first element and the
minimum element is now in sorted part of array
        int temp = arr[pass];
        arr[pass] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

// Function to print an array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main()
{
    int arr[] = {34,76,50,1,48};
    int n = 5;

    printf("Original array: ");
    printArray(arr, n);

    selectionSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

```
=Microsoft-MIEngine-Pid-wb0gglwm.yus' '--dbgExe=C:\m
Original array: 34 76 50 1 48
Sorted array: 1 34 48 50 76
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA>
```

4] Insertion sort

```c
#include <stdio.h>

// Function to perform Insertion Sort
void insertionSort(int arr[], int n)
{
    int i, key, j;

    // Iterate through the array starting from the second element (index 1)
    for (i = 1; i < n; i++)
    {
        key = arr[i]; // Current element to be inserted in the sorted array
        j = i - 1;    // Index of the previous element

        // compare the value of key(the element to be sorted) and the element
in the sorted array
        while (j >= 0 && key < arr[j])
        {
            arr[j + 1] = arr[j]; // when key(element to be inserted) is
smaller than element in sorted array, replace key by element in sorted array
            j = j - 1;           // decrement the value of j to compare to the
previous element(in sorted array) in next iteration
        }
        arr[j + 1] = key; // Place the key at its correct position
    }
}

int getArraySize(int arr[])
{
    int size = 0;

    // Iterate through the array until the end marker is found
    while (arr[size] != '\0')
    {
        size++;
    }
```

```c
    return size;
}

// Function to print an array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main()
{
    int arr[] = {34, 67, 99, 56, 29};
    int n = getArraySize(arr);

    printf("Original array: ");
    printArray(arr, n);

    insertionSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

```
Original array: 34 67 99 56 29
Sorted array: 29 34 56 67 99
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA>
```

Postlab

1. i] Bubble sort

For pass 1:
no. of comparisons = (N-1)
no. of swaps = (N-1)

For pass 2:
no. of comparisons = (N-2)
no. of swaps = (N-2)

∴ for pass N-1:
no. of comparisons = 1
no. of swaps = 1

∴ Total no. = $(N-1)+(N-2)+(N-3)+\dots+2+1$

$= \frac{(N-1)(N-(+1))}{2}$

$= \frac{(N)(N-1)}{2}$

∴ complexity $= O(N^2)$

ii) Selection Sort:-

For First iteration:

no of comparisons = (n-1)

For second iteration:

no. of comparisons = (n-2)

∴ for n^{th} iteration:

no. of comparisons = 1

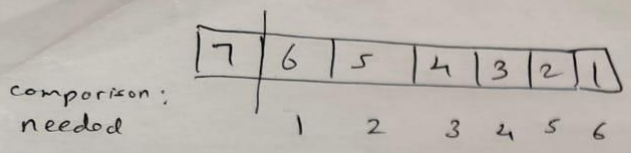
∴ Total = $(n-1)+(n-2)+\ldots+1$

$= \frac{(n-1)(n-1+1)}{2}$

$= \frac{n(n-1)}{2}$

∴ complexity = $O(n^2)$

iii) Insertion Sort



comparison needed: 1 2 3 4 5 6

∴ Total no. of comparisons = 1+2+3+4+5+6

= 1+2+3+4+5+(N-1)

∴ For large quantities of data,

= 1+2+3... + (N-1)

= N(N-1)/2

∴ complexity = O(N²)

Q.2 Array = [2 | 4 | 3]
max

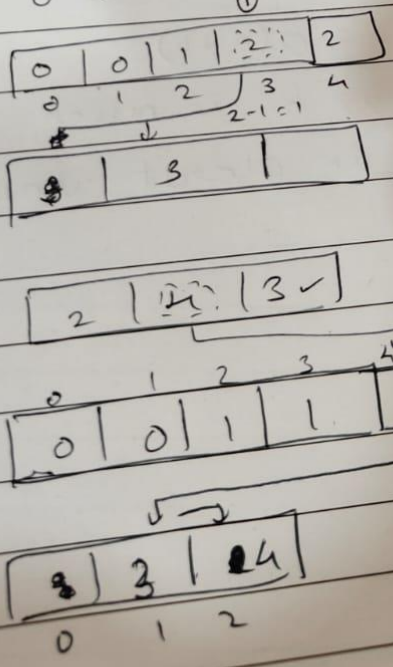
step 1:-
count array = [0 | 0 | 0 | 0 | 0]
0 1 2 3 4

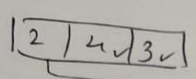
step 2:-
After storing no. of repetitions of values of input array at respective location

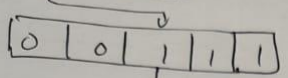
count_array = [0 | 0 | 1 | 1 | 0]
0 1 2 3 4

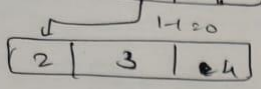
step 3:-
count array = [0 | 0 | 1 | 2 | 2]

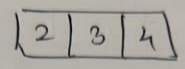
step 4:- a) [2 | 4 | 3]
0 1 2



a) 

b) 

c) 

∴ Output array = 

Complexity

→ for the counting phase, it iterate through array once ∴ time = O(n)

→ To make count array of size k (max value) takes O(k) time

→ To build output array we iterate through array ∴ takes O(n) time

∴ overall complexity = O(n+k) + O(k) + O(n)

= O(n+k)

as k is much larger than in worst case.