Practical - 1

Implement a function for each of following problems and count the number of steps executed/Time taken by each function on various inputs and write complexity of each function. Also draw a comparative chart. In each of the following function N will be passed by user.

1. To calculate sum of 1 to N number using loop.

Code:-

```c
#include <stdio.h>
void add_loop(int);
int count = 0;
int main()
{
    int n;
    printf("Enter number:- ");
    scanf("%d", &n);
    add_loop(n);
    return 0;
}
void add_loop(int n)
{
    int i;
    int sum = 0;
    count++;
    for (i = 1; i <= n; i++)
    {
        count++;
        sum = sum + i;
        count++;
    }
    count++;
    printf("%d", count);
}
```

Output:-

```
PS D:\Documents\DESIGN AND ANALYSIS OF ALGORITHM>
Enter number:- 5
12
```

2. To calculate sum of 1 to N number using equation.

Code:-

```c
#include <stdio.h>
int count = 0;
int main()
{
    int n;
    int sum = 0;
    printf("enter number:- ");
    scanf("%d", &n);
    count++;
    sum = (n * (n + 1)) / 2;
    printf("sum is:- %d", sum);
    printf("\ncount is:- %d", count);
    return 0;
}
```

Output:-

```
PS D:\Documents\DESIGN AND ANALYSIS OF ALGORITHM\PRACTICAL 1>
{ .\EQUATION }
enter number:- 5
sum is:- 15
count is:- 1
```

3. To calculate sum of 1 to N numbers using recursion. [Font Style Times New Roman – 12]

Code:-

```c
#include <stdio.h>
int count = 0;
void main()
{
    int n;
    int sum = 0;
    printf("enter number:- ");
    scanf("%d", &n);
    sum = rec(n);
    printf("sum is:-%d", sum);
    printf("\ncount is:-%d", count);
}
int rec(int n)
{
    count++;
    if (n != 0)
    {
        count++;
```

```
        return n + rec(n - 1);
        count++;
    }
    else
    {
        count++;
        count++;
        return 0;
    }
}
```

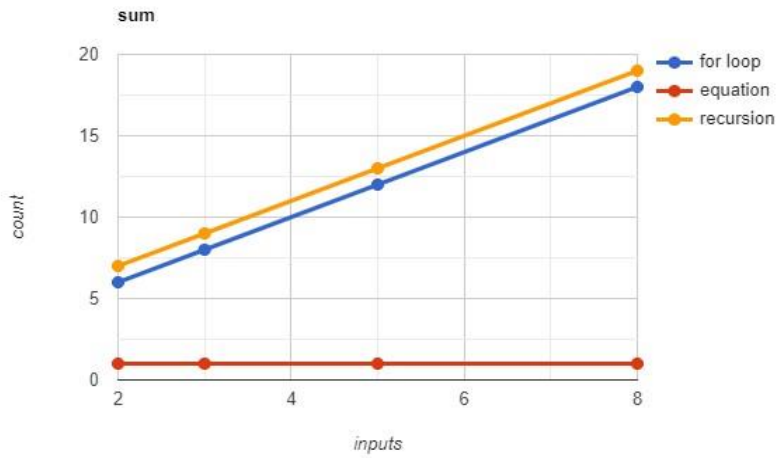
Output:-

```
enter number:- 5
sum is:-15
count is:-13
```

Analysis:-

Method	Step count for 2	Step count for 3	Step count for 5	Step count for 8
For loop	6	8	12	18
Equation	1	1	1	1
Recursion	7	9	13	19

Analysis chart:-



Conclusion:-

- in best case for loop method is a best method.
- in average case equation and recursion are best methods.
- In worst case there are no any methods.

Practical - 2

Implement functions to print nth Fibonacci number using iteration and recursive method. Compare the performance of two methods by counting number of steps executed on various inputs. Also draw a comparative chart. (Fibonacci series 1, 1, 2, 3, 5, 8….. Here 8 is the 6th Fibonacci number) **1) Using iteration method Code:**

```c
#include <stdio.h>
void main()
{
    int n1 = 0, n2 = 1, n3, i, n, count = 0;
    printf("Enter nth Fibonacci number: ");
    scanf("%d", &n);
    printf("The Fibonacci series is: ");
    printf("%d %d ", n1, n2);
    count++;
    for (i = 3; i <= n; i++)
    {
        count++;
        n3 = n1 + n2;
        printf("%d ", n3);
        count++;
        n1 = n2;
        count++;
        n2 = n3;
        count++;
    }
    count++;
    printf("\nCount= %d", count);
}
```

Output:

```
Enter nth Fibonacci number: 5
The Fibonacci series is: 0 1 1 2 3
Count= 14
```

2) Using recursive method Code:

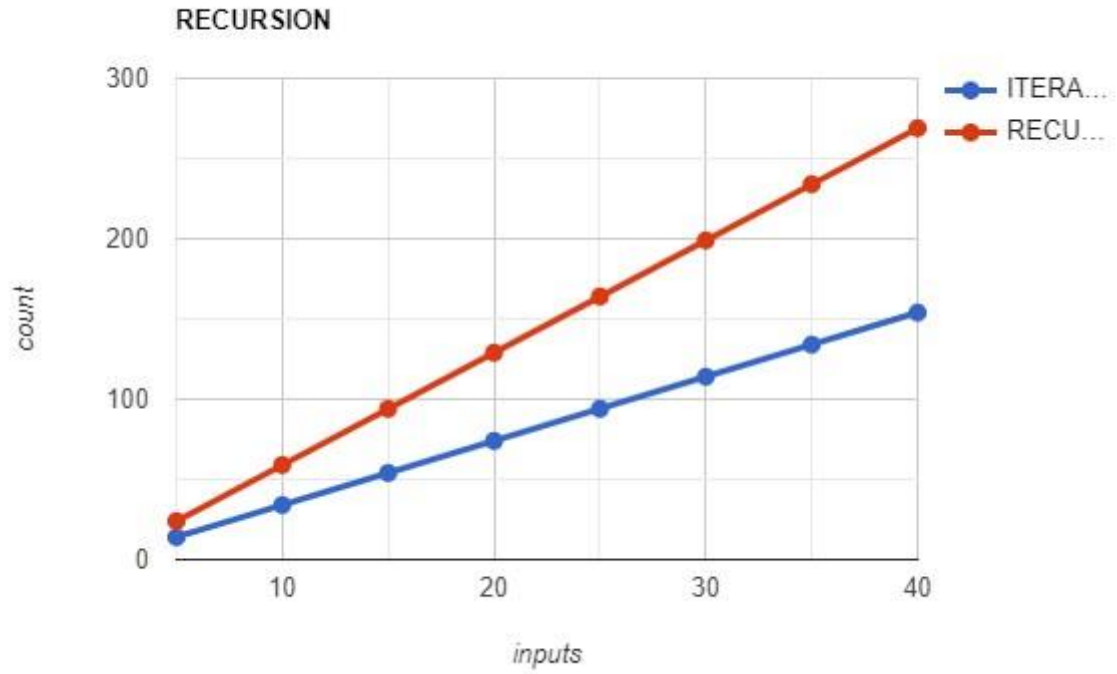
```c
#include <stdio.h>
void Fibonacci(int);
int count = 0;
void main()
{
    int n;
    printf("Enter nth Fibonacci number: ");
    scanf("%d", &n);
    printf("The Fibonacci series is: ");
    printf("%d %d ", 0, 1);
    count++;
    Fibonacci(n - 2);
    printf("\nCount= %d", count);
}
void Fibonacci(int n)
{
    static int n1 = 0, n2 = 1, n3;
    count++;
    if (n > 0)
    {
        count++;
        n3 = n1 + n2;
        count++;
        n1 = n2;
        count++;
        n2 = n3;
        printf("%d ", n3);
        count++;
        Fibonacci(n - 1);
        count++;
    }
    count++;
}
```

Output:

```
Enter nth Fibonacci number: 5
The Fibonacci series is: 0 1 1 2 3
Count= 24
```

Analysis:

INPUT	ITERATIVE	RECURSION
5	**14**	**24**
10	**34**	**59**
15	**54**	**94**
20	**74**	**129**
25	**94**	**164**
30	**114**	**199**
35	**134**	**234**
40	**154**	**269**



Conclusion:

BEST CASE: Iteration method

WORST CASE: Recursive method

Practical - 3

Write user defined functions for the following sorting methods and compare their performance by time measurement with random data and Sorted data.

1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Randomized Quick sort

Using selection sort:

Code:

```c
#include <stdio.h>
#include <time.h>
void selection_sort(int[], int);
void main()
{
    int a[10000], n, i;
    time_t t;
    double time_taken;
    clock_t start, end;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    srand((unsigned)time(&t));
    for (i = 0; i < n; i++)
    {
        a[i] = rand() % 1000;
    }
    printf("Random generated %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    selection_sort(a, n);
    end = clock();
    printf("\nAfter selection sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
```

```
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by selection sort for random unsorted array:
%lf", time_taken);
    for (i = 0; i < n; i++)
    {
        a[i] = i + 1;
    }
    printf("\nSorted %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    selection_sort(a, n);
    end = clock();
    printf("\nAfter selection sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by selection sort for sorted array: %lf",
time_taken);
}
void selection_sort(int a[], int n)
{
    int i, j, min, temp;
    for (i = 0; i < n - 1; i++)
    {
        min = i;
        for (j = i + 1; j < n; j++)
        {
            if (a[min] > a[j])
            {
                min = j;
            }
        }
        if (min != i)
        {
            temp = a[i];
            a[i] = a[min];
            a[min] = temp;
        }
    }
}
```

Output:

```
Enter number of elements: 5
Random generated 5 elements are: 416 296 59 551 204
After selection sort, elements are: 59 204 296 416 551
Time taken by selection sort for random unsorted array: 0.000003
Sorted 5 elements are: 1 2 3 4 5
After selection sort, elements are: 1 2 3 4 5
Time taken by selection sort for sorted array: 0.000000
```

Using bubble sort:

Code:

```c
#include <stdio.h>
#include <time.h>
void bubble_sort(int[], int);
void main()
{
    int a[10000], n, i;
    time_t t;
    double time_taken;
    clock_t start, end;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    srand((unsigned)time(&t));
    for (i = 0; i < n; i++)
    {
        a[i] = rand() % 1000;
    }
    printf("Random generated %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    bubble_sort(a, n);
    end = clock();
    printf("\nAfter bubble sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) /
                 CLOCKS_PER_SEC;
 printf("\nTime taken by bubble sort for random
unsorted array: %lf", time_taken);
```

```
 for (i = 0; i < n; i++)
 {
        a[i] = i + 1;
 }
 printf("\nSorted %d elements are: ", n);
 for (i = 0; i < n; i++)
 {
        printf("%d ", a[i]);
 }
 start = clock();
 bubble_sort(a, n);
 end = clock();
 printf("\nAfter bubble sort, elements are: ");
 for (i = 0; i < n; i++)
 {
        printf("%d ", a[i]);
 }
 time_taken = (double)(end - start) /
CLOCKS_PER_SEC;
 printf("\nTime taken by bubble sort for sorted
array: %lf", time_taken);
}
void bubble_sort(int a[], int n)
{
 int i, j, temp, flag;
 for (i = 0; i < n - 1; i++)
 {
     flag = 0;
     for (j = 0; j < n - 1 - i; j++)
     {
         if (a[j] > a[j + 1])
         {
             temp = a[j];
             a[j] = a[j + 1];
             a[j + 1] = temp;
             flag = 1;
         }
     }
     if (flag == 0)
     {
         break;
     }
 }
}
```

Output:

```
Enter number of elements: 5
Random generated 5 elements are: 297 19 267 333 776
After bubble sort, elements are: 19 267 297 333 776
Time taken by bubble sort for random unsorted array: 0.000002
Sorted 5 elements are: 1 2 3 4 5
After bubble sort, elements are: 1 2 3 4 5
Time taken by bubble sort for sorted array: 0.000001
```

Using insertion sort:

Code:

```c
#include <stdio.h>
#include <time.h>
void insertion_sort(int[], int);
void main()
{
    int a[10000], n, i;
    time_t t;
    double time_taken;
    clock_t start, end;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    srand((unsigned)time(&t));
    for (i = 0; i < n; i++)
    {
        a[i] = rand() % 1000;
    }
    printf("Random generated %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    insertion_sort(a, n);
    end = clock();
    printf("\nAfter insertion sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by insertion sort for random unsorted array:
%lf", time_taken);
```

```
    for (i = 0; i < n; i++)
    {
        a[i] = i + 1;
    }
    printf("\nSorted %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    insertion_sort(a, n);
    end = clock();
    printf("\nAfter insertion sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by insertion sort for sorted array: %lf",
time_taken);
}
void insertion_sort(int a[], int n)
{
    int i, j, temp;
    for (i = 1; i < n; i++)
    {
        temp = a[i];
        j = i - 1;
        while (j >= 0 && a[j] > temp)
        {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = temp;
    }
}
```

Output:

```
Enter number of elements: 5
Random generated 5 elements are: 494 914 655 673 98
After insertion sort, elements are: 98 494 655 673 914
Time taken by insertion sort for random unsorted array: 0.000003
Sorted 5 elements are: 1 2 3 4 5
After insertion sort, elements are: 1 2 3 4 5
Time taken by insertion sort for sorted array: 0.000000
```

Using merge sort:

Code:

```c
#include <stdio.h>
#include <time.h>
int a[10000], b[10000];
void merge(int a[], int lb, int mid, int ub)
{
    int i, j, k;
    i = lb;
    j = mid + 1;
    k = lb;
    while (i <= mid && j <= ub)
    {
        if (a[i] <= a[j])
        {
            b[k] = a[i];
            i++;
            k++;
        }
        else
        {
            b[k] = a[j];
            j++;
            k++;
        }
    }
    if (i > mid)
    {
        while (j <= ub)
        {
            b[k] = a[j];
            j++;
            k++;
        }
    }
    else
    {
        while (i <= mid)
        {
            b[k] = a[i];
            i++;
            k++;
        }
    }
    for (i = lb; i <= ub; i++)
    {
        a[i] = b[i];
```

```
    }
}
void merge_sort(int a[], int lb, int ub)
{
    int mid;
    if (lb < ub)
    {
        mid = (lb + ub) / 2;
        merge_sort(a, lb, mid);
        merge_sort(a, mid + 1, ub);
        merge(a, lb, mid, ub);
    }
}
void main()
{
    int n, i, j;
    time_t t;
    double time_taken;
    clock_t start, end;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    srand((unsigned)time(&t));
    for (i = 0; i < n; i++)
    {
        a[i] = rand() % 1000;
    }
    printf("Random generated %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    merge_sort(a, 0, n - 1);
    end = clock();
    printf("\nAfter merge sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by merge sort for random unsorted array: %lf",
time_taken);
    for (i = 0; i < n; i++)
    {
        a[i] = i + 1;
    }
    printf("\nSorted %d elements are: ", n);
    for (i = 0; i < n; i++)
```

```
    {
        printf("%d ", a[i]);
    }
    start = clock();
    merge_sort(a, 0, n - 1);
    end = clock();
    printf("\nAfter merge sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by merge sort for sorted array: %lf",
time_taken);
}
```

Output:

```
Enter number of elements: 5
Random generated 5 elements are: 838 885 713 554 332
After merge sort, elements are: 332 554 713 838 885
Time taken by merge sort for random unsorted array: 0.000009
Sorted 5 elements are: 1 2 3 4 5
After merge sort, elements are: 1 2 3 4 5
Time taken by merge sort for sorted array: 0.000001
```

Using quick sort:

Code:

```
#include <stdio.h>
#include <time.h>
void quick_sort(int a[], int lb, int ub)
{
    int pivot, start, end, temp;
    if (lb < ub)
    {
        pivot = lb;
        start = lb;
        end = ub;
        while (start < end)
        {
            while (a[pivot] >= a[start])
            {
                start++;
            }
```

```
            while (a[pivot] < a[end])
            {
                end--;
            }
            if (start < end)
            {
                temp = a[start];
                a[start] = a[end];
                a[end] = temp;
            }
        }
        temp = a[pivot];
        a[pivot] = a[end];
        a[end] = temp;
        quick_sort(a, lb, end - 1);
        quick_sort(a, end + 1, ub);
    }
}
int main()
{
    int a[100], i, n;
    time_t t;
    double time_taken;
    clock_t start, end; // lb-lower bound ub-upper bound printf("Enter
number of elements: ");
    scanf("%d", &n);
    srand((unsigned)time(&t));
    for (i = 0; i < n; i++)
    {
        a[i] = rand() % 1000;
    }
    printf("Random generated %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    quick_sort(a, 0, n - 1);
    end = clock();
    printf("\nAfter quick sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by quick sort for random unsorted array: %lf",
time_taken);
    for (i = 0; i < n; i++)
```

```
    {
        a[i] = i + 1;
    }
    printf("\nSorted %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    quick_sort(a, 0, n - 1);
    end = clock();
    printf("\nAfter quick sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by quick sort for sorted array: %lf",
time_taken);
    return 0;
}
```

Output:

```
Enter number of elements: 5
Random generated 5 elements are: 600 859 814 697 743
After quick sort, elements are: 600 697 743 814 859
Time taken by quick sort for random unsorted array: 0.000003
Sorted 5 elements are: 1 2 3 4 5
After quick sort, elements are: 1 2 3 4 5
Time taken by quick sort for sorted array: 0.000002
```

Using randomized quick sort:

Code:

```
#include <stdio.h>
#include <time.h>
void quick_sort(int a[], int lb, int ub)
{
    int pivot, start, end, temp;
    if (lb < ub)
    {
```

```
        pivot = rand() % (ub + 1);
        start = lb;
        end = ub;
        while (start < end)
        {
            while (a[pivot] >= a[start])
            {
                start++;
            }
            while (a[pivot] < a[end])
            {
                end--;
            }
            if (start < end)
            {
                temp = a[start];
                a[start] = a[end];
                a[end] = temp;
            }
        }
        temp = a[pivot];
        a[pivot] = a[end];
        a[end] = temp;
        quick_sort(a, lb, end - 1);
        quick_sort(a, end + 1, ub);
    }
}
int main()
{
    int a[10000], i, n;
    time_t t;
    double time_taken;
    clock_t start, end;
    // lb-lower bound ub-upper bound
    printf("***RANDOMIZED QUICK SORT***\n");
    printf("Enter number of elements: ");
    scanf("%d", &n);
    srand((unsigned)time(&t));
    for (i = 0; i < n; i++)
    {
        a[i] = rand() % 1000;
    }
    printf("Random generated %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
```

```
    quick_sort(a, 0, n - 1);
    end = clock();
    printf("\nAfter quick sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by quick sort for random unsorted array: %lf",
time_taken);
    for (i = 0; i < n; i++)
    {
        a[i] = i + 1;
    }
    printf("\nSorted %d elements are: ", n);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    start = clock();
    quick_sort(a, 0, n - 1);
    end = clock();
    printf("\nAfter quick sort, elements are: ");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\nTime taken by quick sort for sorted array: %lf",
time_taken);
    return 0;
}
```

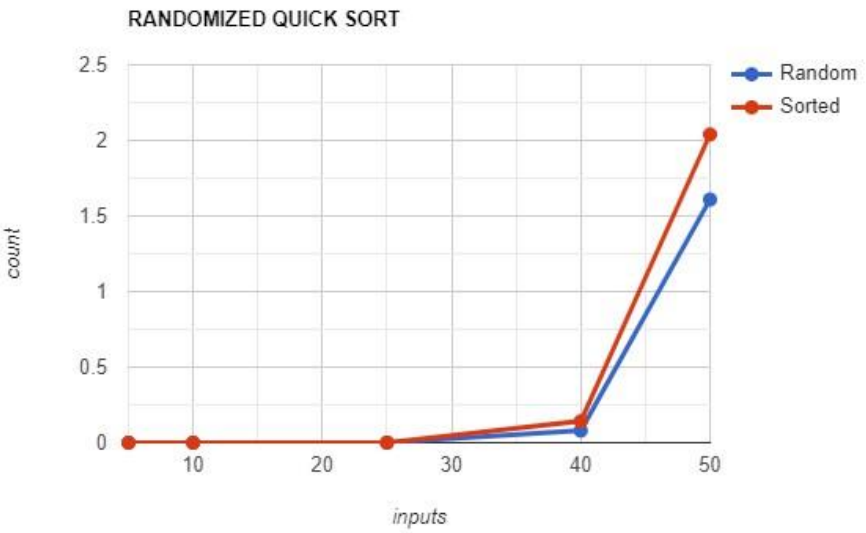
Output:

```
***RANDOMIZED QUICK SORT***
Enter number of elements: 5
Random generated 5 elements are: 328 452 687 926 452
After quick sort, elements are: 328 452 452 687 926
Time taken by quick sort for random unsorted array: 0.000003
Sorted 5 elements are: 1 2 3 4 5
After quick sort, elements are: 1 2 3 4 5
Time taken by quick sort for sorted array: 0.000002
```

Analysis:

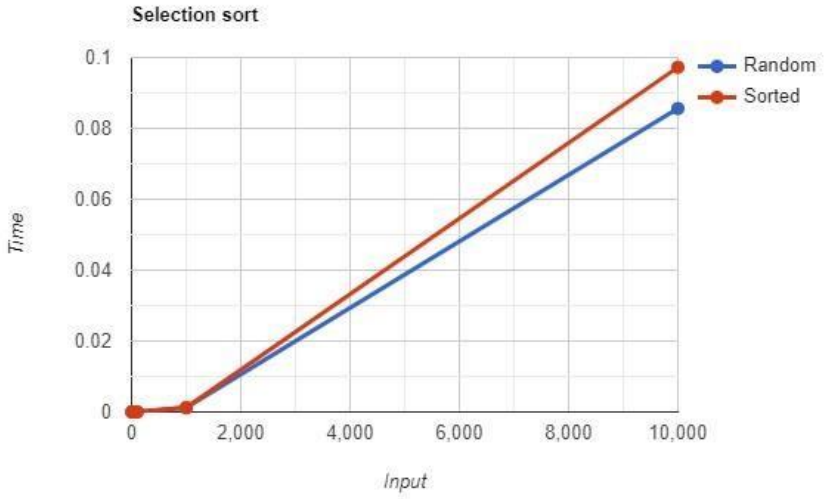
For Randomized Qsort:

Input	Random	Sorted
5	0.000003	0.000002
10	0.000004	0.000004
25	0.000048	0.000661
40	0.079794	0.142141
50	1.608957	2.039727



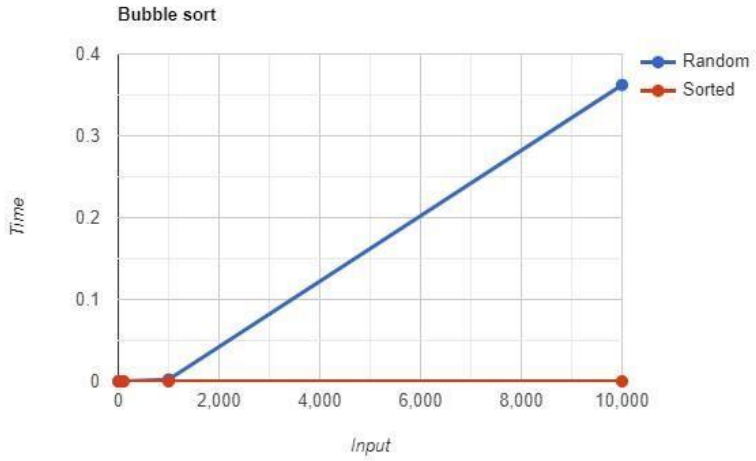
For Selection sort:

Input	Random	Sorted
5	0.000003	0.000001
25	0.000003	0.000006
100	0.000015	0.000013
1000	0.001187	0.001212
10000	0.085607	0.097245



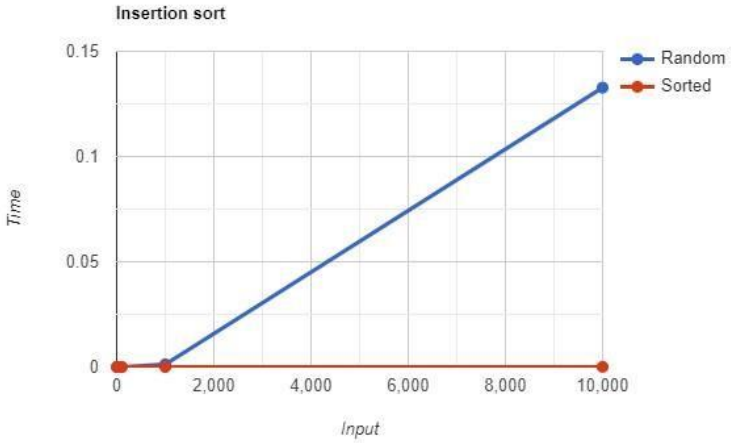
For Bubble sort:

Input	Random	Sorted
5	0.000003	0.000002
25	0.000005	0.000002
100	0.000049	0.000002
1000	0.002263	0.000004
10000	0.3622	0.000035



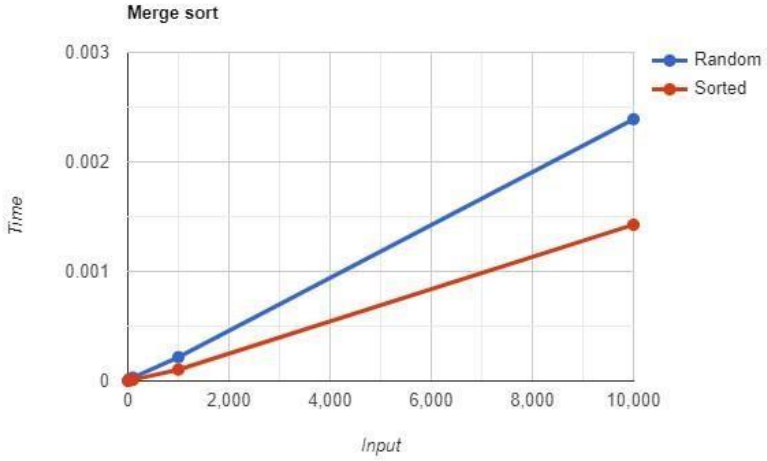
For Insertion sort:

Input	Random	Sorted
5	0.000003	0.000001
25	0.000003	0.000002
100	0.000015	0.000002
1000	0.001259	0.000009
10000	0.132727	0.000076



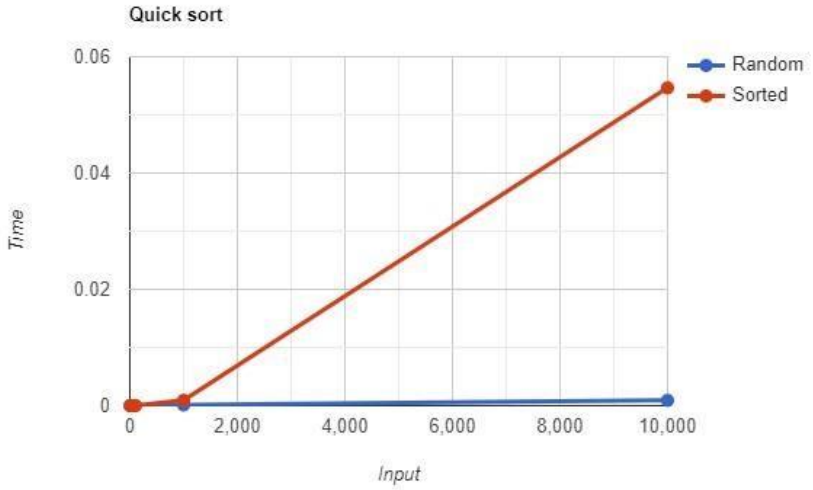
For Merge sort:

Input	Random	Sorted
5	0.000003	0.000002
25	0.000016	0.000004
100	0.000031	0.000011
1000	0.000217	0.000104
10000	0.00239	0.001426



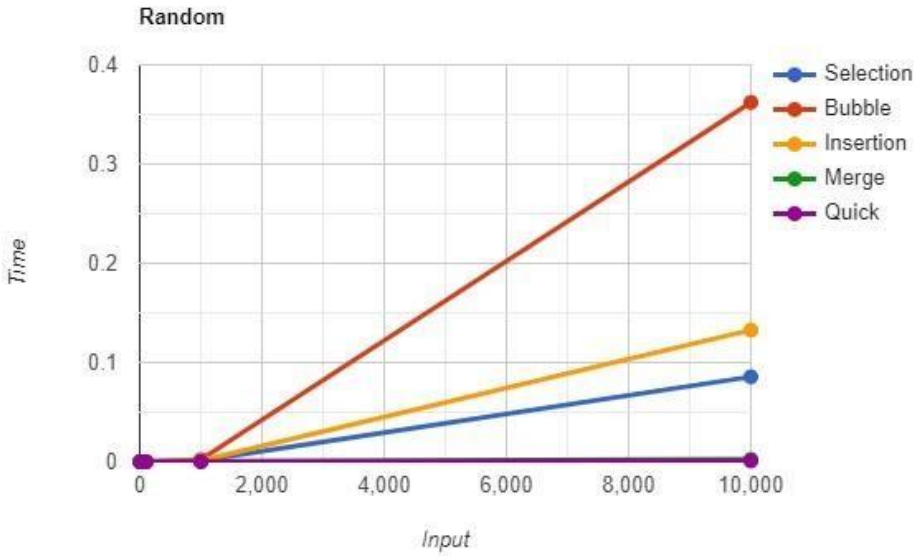
For Quick sort:

Input	Random	Sorted
5	0.000003	0.000002
25	0.000002	0.000001
100	0.000019	0.00001
1000	0.000111	0.000939
10000	0.000915	0.054668



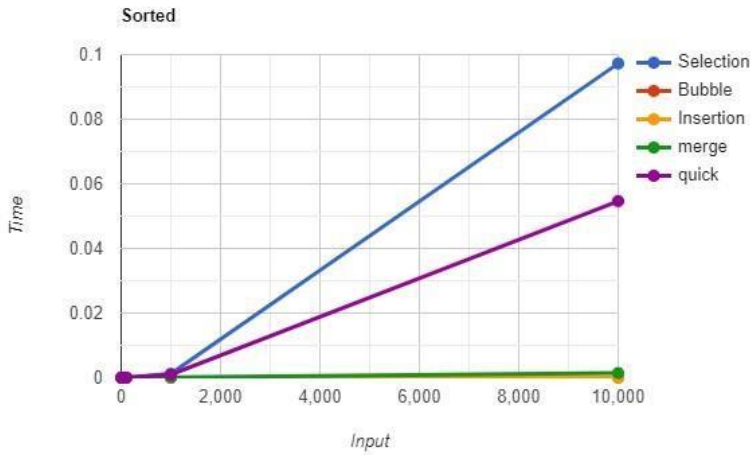
For Random unsorted array:

Input	Selection	Bubble	Insertion	Merge	Quick
5	0.000003	0.000003	0.000003	0.000003	0.000003
25	0.000003	0.000005	0.000003	0.000016	0.000002
100	0.000015	0.000049	0.000015	0.000031	0.000019
1000	0.001187	0.002263	0.001259	0.000217	0.000111
10000	0.085607	0.3622	0.132727	0.00239	0.000915



For sorted array:

Input	Selection	Bubble	Insertion	Merge	Quick
5	0.000001	0.000002	0.000001	0.000002	0.000002
25	0.000006	0.000002	0.000002	0.000004	0.000001
100	0.000013	0.000002	0.000002	0.000011	0.00001
1000	0.001212	0.000004	0.000009	0.000104	0.000939
10000	0.097245	0.000035	0.000076	0.001426	0.054668



Conclusion:

BEST CASE:

For random: selection sort, quick sort.

For sorted: bubble sort, insertion sort, merge sort.

WORST CASE:

For random: bubble sort, insertion sort, merge sort.

For sorted: selection sort, quick sort.

Practical - 4

Implement a function of Sequential Search & Binary Search and count the steps executed by function on various inputs for best case and worst case. Also write complexity in each case and draw a comparative chart.

Sequential Search Code:

```c
#include <stdio.h>
void SequentialSearch(int[], int);
int count = 0;
void main()
{
    int a[100], n, num, i;
    printf("Enter a number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
    count++;
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    count++;
    count++;
    SequentialSearch(a, n);
    printf("\nCount= %d", count);
}
void SequentialSearch(int a[], int n)
{
    int i, num;
    printf("Enter a number to search: ");
    scanf("%d", &num);
    count++;
    for (i = 0; i < n; i++)
    {
        count++;
        if (a[i] == num)
        {
            printf("%d is present at index %d", num, i);
            count++;
            break;
        }
    }
    count++;
    count++;
    if (i == n)
    {
        printf("%d is not present in the array", num);
```

```
    }
    count++;
}
```

Output:

Best

```
Enter a number of elements: 5
Enter 5 elements: 6 7 9 3 4
Enter a number to search: 6
6 is present at index 0
Count= 9
```

Average

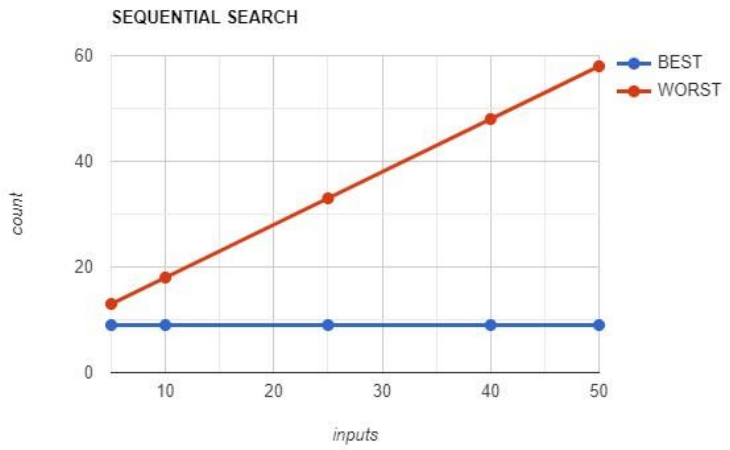
```
Enter a number of elements: 5
Enter 5 elements: 6 7 9 3 4
Enter a number to search: 9
9 is present at index 2
Count= 11
```

Worst

```
Enter a number of elements: 5
Enter 5 elements: 6 7 9 3 4
Enter a number to search: 4
4 is present at index 4
Count= 13
```

Analysis:

Input	Best	Worst
5	9	13
10	9	18
25	9	33
40	9	48
50	9	58



Conclusion:

BEST CASE:

Time Complexity: O(1)

Space Complexity: O(1)

WORST CASE:

Time Complexity: O(n)

Space Complexity: O(1)

Binary Search Code:

```c
#include <stdio.h>
void BinarySearch(int[], int);
void bubble_sort(int[], int);
int count = 0;
void main()
{
    int n, a[100], i;
    printf("Enter a number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    BinarySearch(a, n);
    printf("\nCount= %d", count);
}
void BinarySearch(int a[], int n)
{
    int low, high, mid, i, j, temp, num;
    count++;
```

```
    bubble_sort(a, n);
    printf("Sorted array: ");
    count++;
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    count++;
    printf("\nEnter a number to search: ");
    scanf("%d", &num);
    low = 0, high = n - 1;
    count++;
    while (low <= high)
    {
        count++;
        mid = (low + high) / 2;
        count++;
        if (a[mid] == num)
        {
            printf("%d is present at index %d", num, mid);
            break;
        }
        else if (a[mid] < num)
        {
            count++;
            count++;
            low = mid + 1;
            count++;
        }
        else
        {
            count++;
            count++;
            high = mid - 1;
        }
        count++;
    }
    count++;
    count++;
    if (low > high)
    {
        printf("%d is not present in the array", num);
    }
    count++;
}
void bubble_sort(int a[], int n)
{
    int i, j, temp, flag;
    for (i = 0; i < n - 1; i++)
    {
        flag = 0;
        for (j = 0; j < n - 1 - i; j++)
        {
            if (a[j] > a[j + 1])
            {
                temp = a[j];
```

```
                a[j] = a[j + 1];
                a[j + 1] = temp;
                flag = 1;
            }
        }
        if (flag == 0)
        {
            break;
        }
    }
}
```

Output:

Best

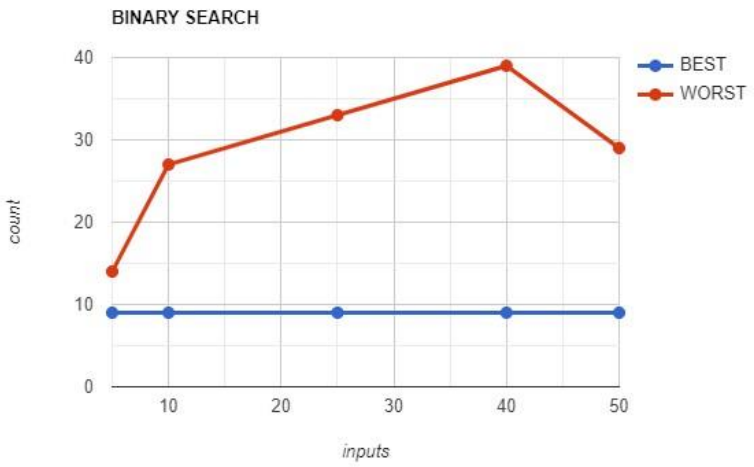
```
Enter a number of elements: 5
Enter 5 elements: 3 9 1 4 6
Sorted array: 1 3 4 6 9
Enter a number to search: 4
4 is present at index 2
Count= 9
```

Worst

```
Enter a number of elements: 5
Enter 5 elements: 3 9 1 4 6
Sorted array: 1 3 4 6 9
Enter a number to search: 1
1 is present at index 0
Count= 14
```

Analysis:

Input	Best	Worst
5	9	14
10	9	27
25	9	33
40	9	39
50	9	29



Conclusion:

BEST CASE:

Time Complexity: O(1)

Space Complexity: O(1)

WORST CASE:

Time Complexity: O(log n)

Space Complexity: O(1)