EXPERIMENT 1

20CP209P – Design and Analysis of Algorithm Lab

Aim:

Implement Insertion Sort and Selection Sort and give complexity analysis

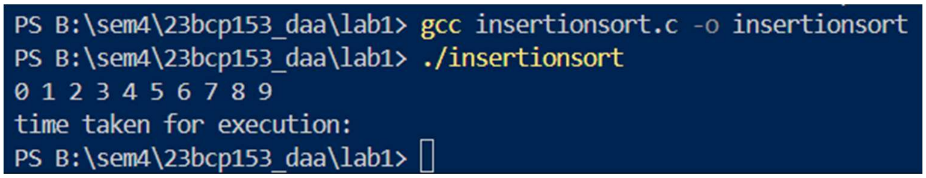
Code:

Insertion Sort:

```
#include <stdio.h>
#include <time.h>

void insertion_sort(int arr[], int len);

int main(void)
{
   clock_t start, end;
   int arr[] = {7,4,8,9,0,1,2,5,3,6};

   int len = sizeof(arr) / sizeof(int);

   start = clock();
   insertion_sort(arr, len);
   end = clock();

   for (int i = 0; i < len; i++)
   {
      printf("%d ", i);
   }
   printf("\n");

   printf("time taken for execution: ", (double) (end - start));

   return 0;
}

void insertion_sort(int arr[], int len)
{
   for (int i = 1; i < n; i++)
   {
      int key = arr[i];
      int j = i - 1;

      while (j >= 0 && arr[j] > key)
      {
         arr[j + 1] = arr[j];
```

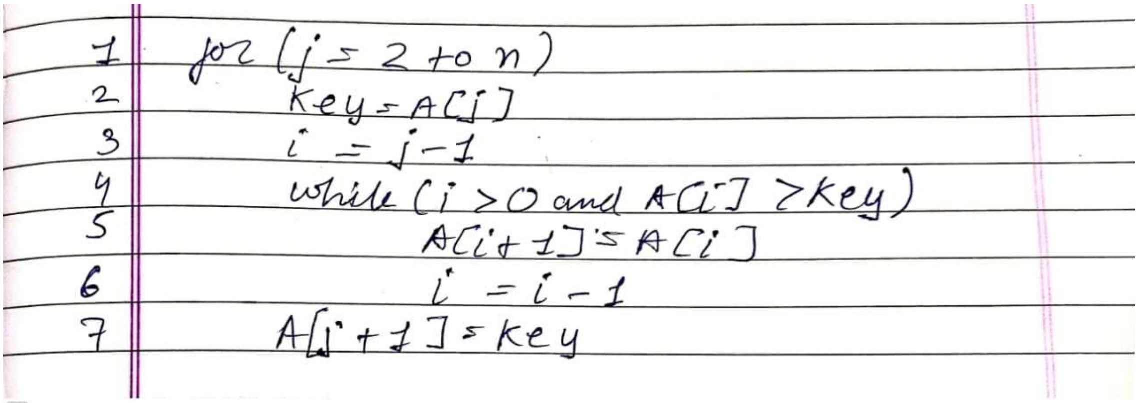
```
            j--;
        }
        arr[j + 1] = key;
    }

    return;
}
```

Output:



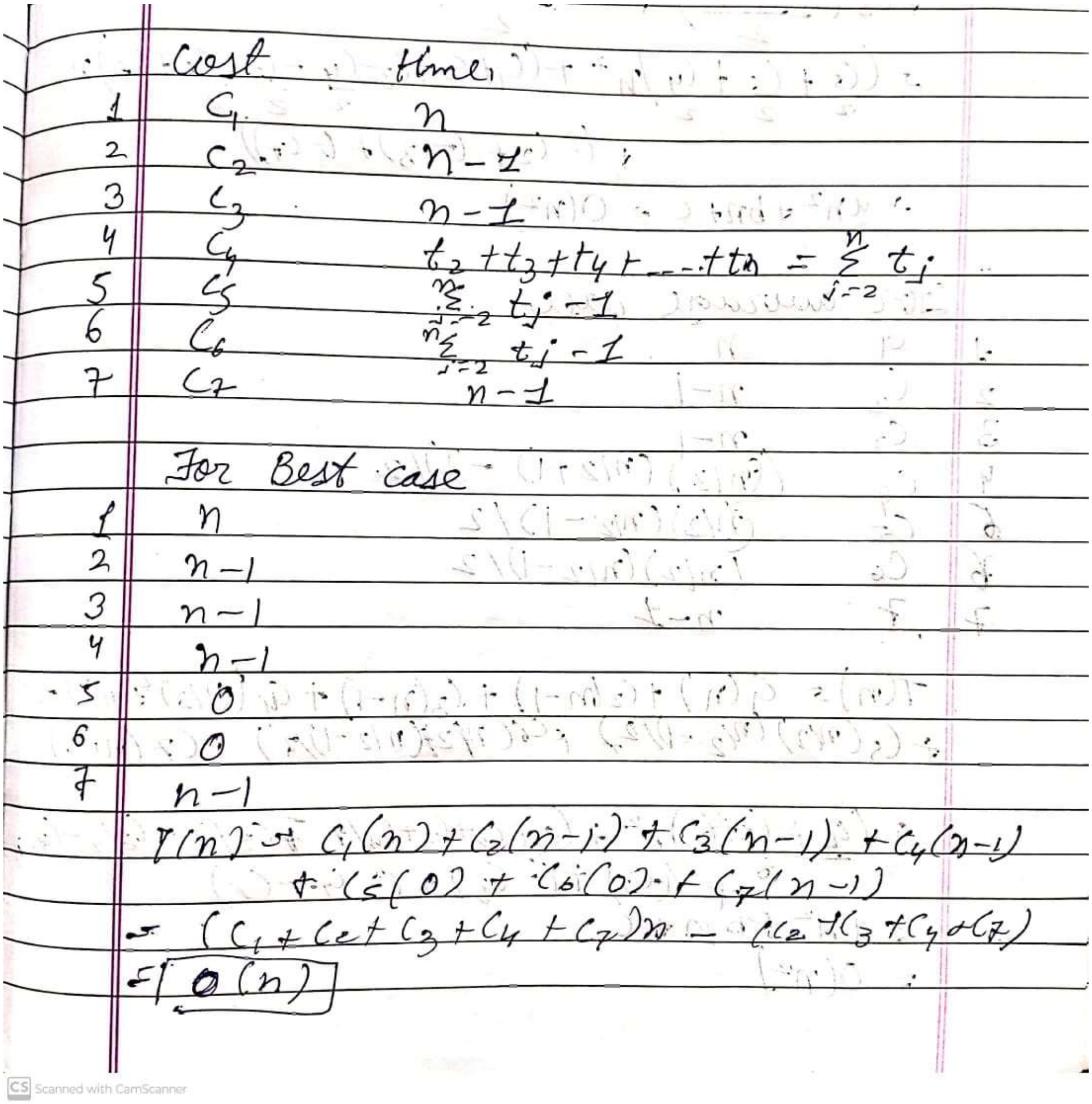
```
PS B:\sem4\23bcp153_daa\lab1> gcc insertionsort.c -o insertionsort
PS B:\sem4\23bcp153_daa\lab1> ./insertionsort
0 1 2 3 4 5 6 7 8 9
time taken for execution:
PS B:\sem4\23bcp153_daa\lab1> 
```

Algorithm:



```
for (j = 2 to n)
    Key = A[j]
    i = j-1
    while (i > 0 and A[i] > key)
        A[i+1] = A[i]
        i = i - 1
    A[i + 1] = key
```

Complexity Analysis:



	Cost	time
1	C_1	n
2	C_2	$n-1$
3	C_3	$n-1$
4	C_4	$t_2+t_3+t_4+ \ldots +t_n = \sum_{j=2}^{n} t_j$
5	C_5	$\sum_{j=2}^{n} t_j - 1$
6	C_6	$\sum_{j=2}^{n} t_j - 1$
7	C_7	$n-1$

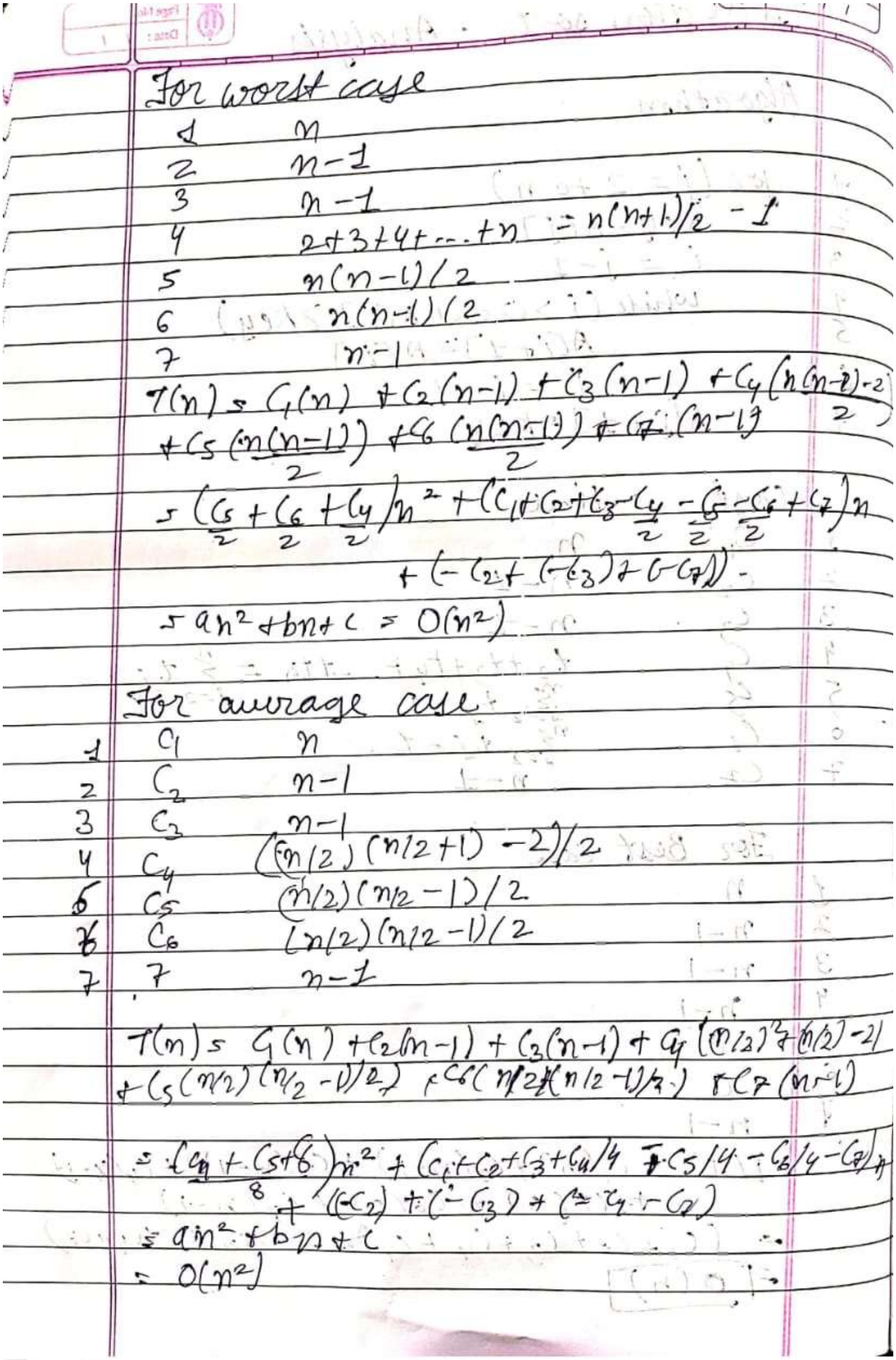
For Best case

1	n
2	$n-1$
3	$n-1$
4	$n-1$
5	0
6	0
7	$n-1$

$$T(n) = C_1(n) + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_5(0) + C_6(0) + C_7(n-1)$$

$$= (C_1 + C_2 + C_3 + C_4 + C_7)n - (C_2 + C_3 + C_4 + C_7)$$

$$= \boxed{O(n)}$$



For worst case

1 n

2 $n-1$

3 $n-1$

4 $2+3+4+\dots+n = n(n+1)/2 - 1$

5 $n(n-1)/2$

6 $n(n-1)/2$

7 $n-1$

$$T(n) = C_1(n) + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n-1)}{2}-2\right) + C_5\left(\frac{n(n-1)}{2}\right) + C_6\left(\frac{n(n-1)}{2}\right) + C_7(n-1)$$

$$= \left(\frac{C_5}{2}+\frac{C_6}{2}+\frac{C_4}{2}\right)n^2 + \left(C_1+C_2+C_3-\frac{C_4}{2}-\frac{C_5}{2}-\frac{C_6}{2}+C_7\right)n + (-C_2+(-C_3)+(-C_7)).$$

$$= an^2+bn+c = O(n^2)$$

For average case

1	C_1	n
2	C_2	$n-1$
3	C_3	$n-1$
4	C_4	$((n/2)(n/2+1)-2)/2$
5	C_5	$(n/2)(n/2-1)/2$
6	C_6	$(n/2)(n/2-1)/2$
7	7	$n-1$

$$T(n) = C_1(n) + C_2(n-1) + C_3(n-1) + C_4((n/2)^2+(n/2)-2) + C_5(n/2)(n/2-1)/2) + C_6(n/2)(n/2-1)/2) + C_7(n-1)$$

$$= \frac{(C_4+C_5+C_6)}{8}n^2 + (C_1+C_2+C_3+C_4/4 - C_5/4 - C_6/4 - C_7)n + ((-C_2)+(-C_3)+(-C_4-C_7))$$

$$= an^2+bn+c$$

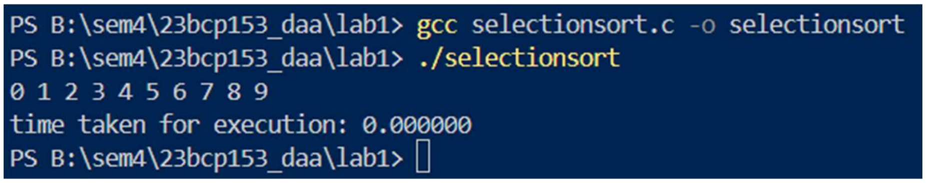
$$= O(n^2)$$

Code:

Selection Sort:

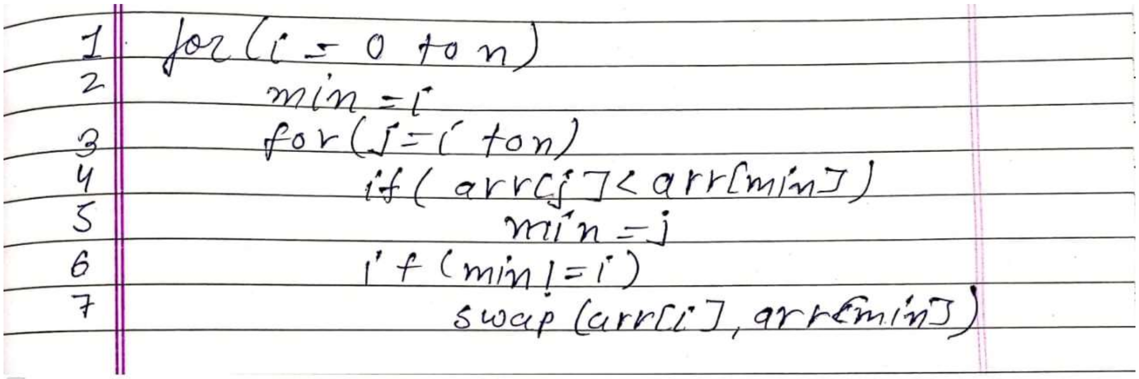
```
#include <stdio.h>
#include <time.h>

void selection_sort(int arr[], int len);

int main(void)
{
    clock_t start, end;
    int arr[] = {7,4,8,9,0,1,2,5,3,6};

    int len = sizeof(arr) / sizeof(int);

    start = clock();
    selection_sort(arr, len);
    end = clock();

    for (int i = 0; i < len; i++)
    {
        printf("%d ", i);
    }
    printf("\n");

    printf("time taken for execution: %f", (double) (end - start));

    return 0;
}

void selection_sort(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int min = i;
        for (int j = i; j < n; j++)
        {
            if (arr[j] < arr[min])
            {
                min = j;
            }
        }
        if (min != i)
        {
            int temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
        }
    }
    return;
}
```

Output:



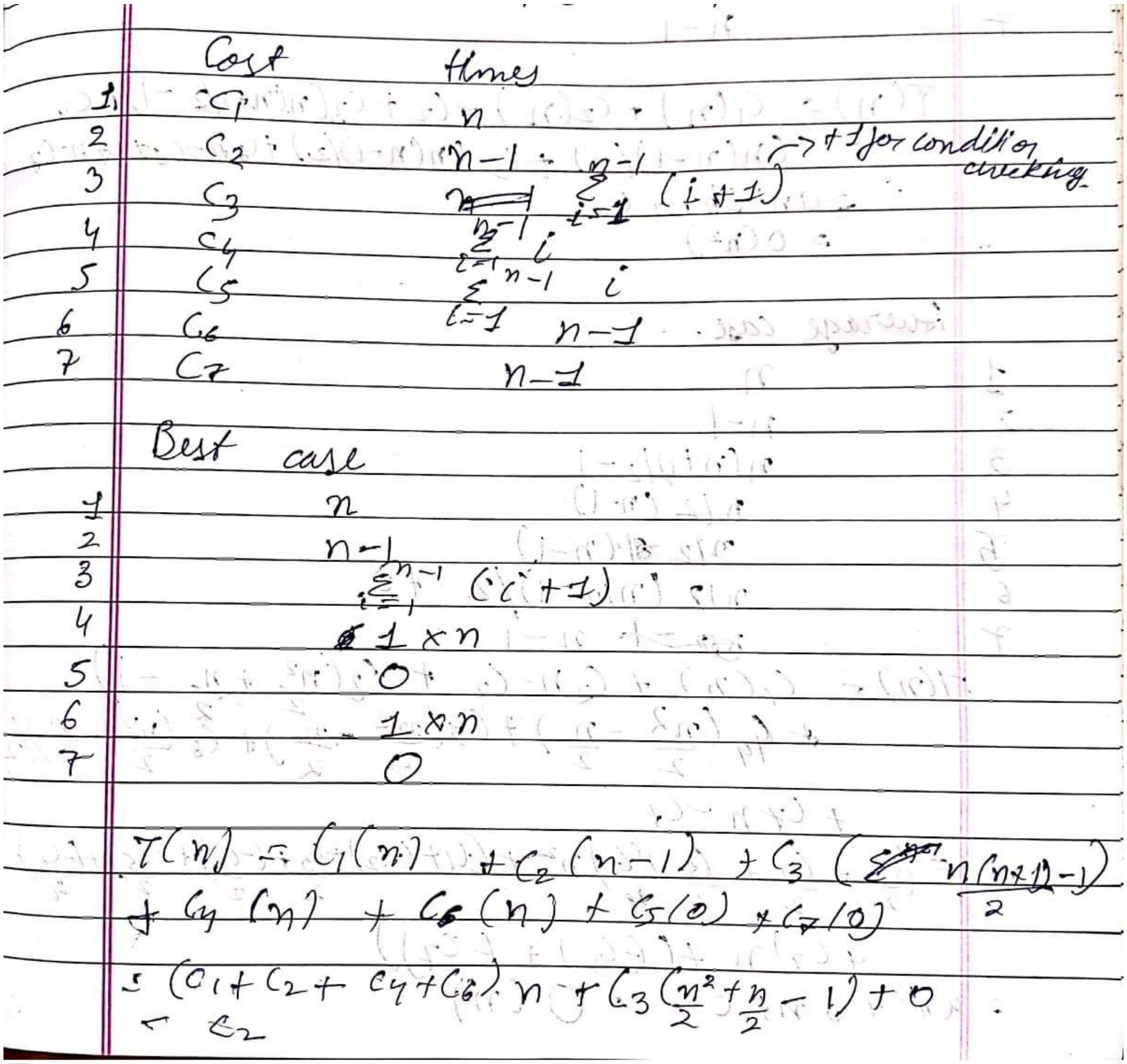
```
PS B:\sem4\23bcp153_daa\lab1> gcc selectionsort.c -o selectionsort
PS B:\sem4\23bcp153_daa\lab1> ./selectionsort
0 1 2 3 4 5 6 7 8 9
time taken for execution: 0.000000
PS B:\sem4\23bcp153_daa\lab1> 
```

Algorithm:



```
for(i = 0 to n)
    min = i
    for(j=i to n)
        if( arr[j]< arr[min])
            min = j
    if (min != i)
        swap (arr[i], arr[min])
```

Complexity Analysis:



	Cost	times
1	C_1	n
2	C_2	$n-1$
3	C_3	~~$n-1$~~ $\sum_{i=1}^{n-1} (i+1)$ → +1 for condition checking
4	C_4	$\sum_{i=1}^{n-1} i$
5	C_5	$\sum_{i=1}^{n-1} i$
6	C_6	$n-1$
7	C_7	$n-1$

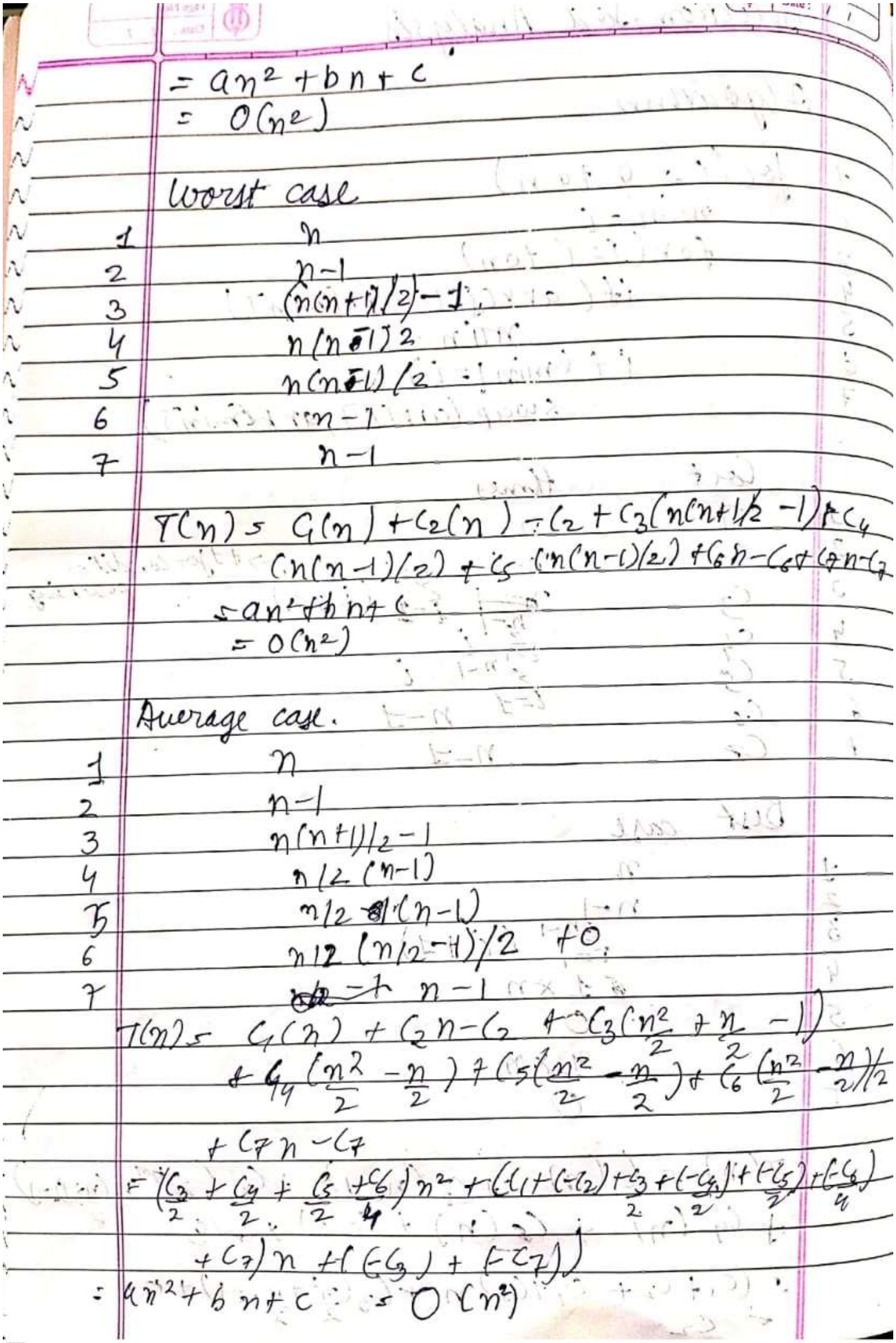
Best case

1	n
2	$n-1$
3	$\sum_{i=1}^{n-1} (i+1)$
4	$1 \times n$
5	0
6	$1 \times n$
7	0

$$T(n) = C_1(n) + C_2(n-1) + C_3\left(\frac{n(n+1)}{2} - 1\right) + C_4(n) + C_6(n) + C_5(0) + C_7(0)$$

$$= (C_1 + C_2 + C_4 + C_6)\,n + C_3\left(\frac{n^2}{2} + \frac{n}{2} - 1\right) + 0$$

$- C_2$



$= an^2 + bn + c$

$= O(n^2)$

Worst case

1	n
2	$n-1$
3	$(n(n+1)/2)-1$
4	$n(n-1)2$
5	$n(n-1)/2$
6	$n-1$
7	$n-1$

$$T(n) = C_1(n) + C_2(n) - C_2 + C_3(n(n+1/2 - 1) + C_4 (n(n-1)/(2) + C_5 (n(n-1)/2) + C_6 n - C_6 + C_7 n - C_7$$

$= an^2 + bn + c$

$= O(n^2)$

Average case.

1	n
2	$n-1$
3	$n(n+1)/2 - 1$
4	$n/2\,(n-1)$
5	$n/2\ \cancel{1}(n-1)$
6	$n/2\,(n/2 - 1)/2 + 0$
7	$\cancel{n} - 1 \quad n-1$

$$T(n) = C_1(n) + C_2 n - C_2 + C_3\left(\frac{n^2}{2} + \frac{n}{2} - 1\right) + C_4\left(\frac{n^2}{2} - \frac{n}{2}\right) + C_5\left(\frac{n^2}{2} - \frac{n}{2}\right) + C_6\left(\frac{n^2}{2} - \frac{n}{2}\right)/2 + C_7 n - C_7$$

$$= \left(\frac{C_3}{2} + \frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{4}\right)n^2 + \left(C_1 + (-C_2) + \frac{C_3}{2} + \left(-\frac{C_4}{2}\right) + \left(-\frac{C_5}{2}\right) + \left(-\frac{C_6}{4}\right) + C_7\right)n + ((-C_3) + (-C_7))$$

$= an^2 + bn + c = O(n^2)$

EXPERIMENT 2

20CP209P – Design and Analysis of Algorithm Lab

Aim:

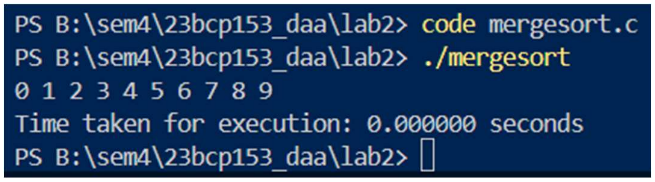
Implement Merge Sort and Quick Sort and give complexity analysis

Code:

Merge Sort:

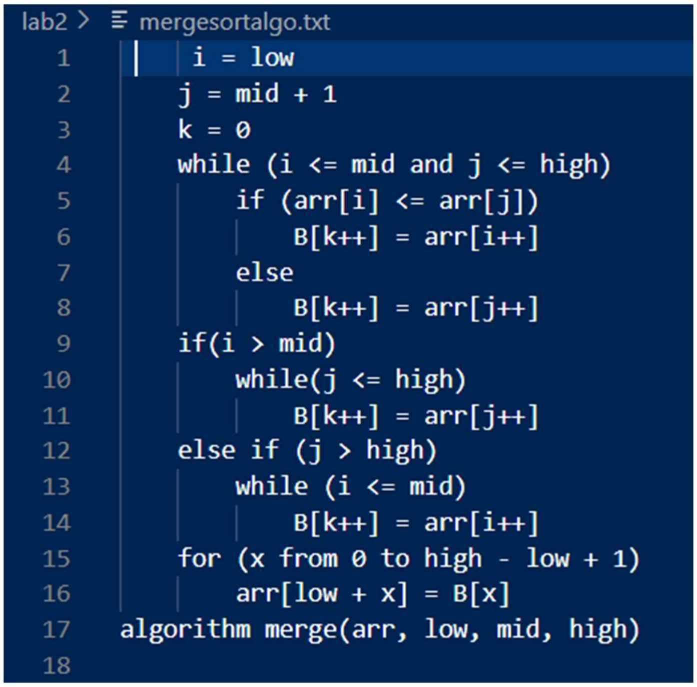
```
#include <stdio.h>
#include <time.h>

void merge_sort(int arr[], int low, int high);
void merge(int arr[], int low, int mid, int high);

int main(void)
{
   clock_t start, end;
   int arr[] = {7, 4, 8, 9, 0, 1, 2, 5, 3, 6};

   int len = sizeof(arr) / sizeof(int);
   int low = 0;
   int high = len - 1;
   start = clock();
   merge_sort(arr, low, high);
   end = clock();

   for (int i = 0; i < len; i++)
   {
      printf("%d ", arr[i]);
   }
   printf("\n");

   printf("Time taken for execution: %f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

   return 0;
}

void merge_sort(int arr[], int low, int high)
{
   if (low < high)
   {
      int mid = (low + high) / 2;
      merge_sort(arr, low, mid);
      merge_sort(arr, mid + 1, high);
      merge(arr, low, mid, high);
   }
}

void merge(int arr[], int low, int mid, int high)
{
    int i = low;
    int j = mid + 1;
    int k = 0;

    int arrB[high - low + 1];

    while (i <= mid && j <= high)
    {
        if (arr[i] <= arr[j])
        {
            arrB[k] = arr[i];
            i++; k++;
        }
        else
        {
            arrB[k] = arr[j];
            j++; k++;
        }
    }

    if (i > mid)
    {
        while (j <= high)
        {
            arrB[k] = arr[j];
            j++; k++;
        }
    }

    else if (j > high)
    {
        while (i <= mid)
        {
            arrB[k] = arr[i];
            i++; k++;
        }
    }

    for (int x = 0; x < (high - low + 1); x++)
        arr[low + x] = arrB[x];

    return;
}
```

Output:



```
PS B:\sem4\23bcp153_daa\lab2> code mergesort.c
PS B:\sem4\23bcp153_daa\lab2> ./mergesort
0 1 2 3 4 5 6 7 8 9
Time taken for execution: 0.000000 seconds
PS B:\sem4\23bcp153_daa\lab2> 
```

Algorithm:



```
    i = low
    j = mid + 1
    k = 0
    while (i <= mid and j <= high)
        if (arr[i] <= arr[j])
            B[k++] = arr[i++]
        else
            B[k++] = arr[j++]
    if(i > mid)
        while(j <= high)
            B[k++] = arr[j++]
    else if (j > high)
        while (i <= mid)
            B[k++] = arr[i++]
    for (x from 0 to high - low + 1)
        arr[low + x] = B[x]
algorithm merge(arr, low, mid, high)
```

Complexity Analysis:

Page No.
Date: / /

Merge Sort

Merge Algorithm Complexity Analysis

	cost	times	(starting from second line)
1	C_1	1	
2	C_2	1	
3	C_3	1	
4	C_4	$n + 1$	
5	C_5	~~2~~ n	
6	C_6	n	
7	C_7	n	
8	C_8	n	
9	C_9	1	
10	C_{10}	n	
11	C_{11}	$n-1$	
12	C_{12}	~~0~~ 1	
13	C_{13}	0	
14	C_{14}	0	
15	C_{15}	$n+1$	
16	C_{16}	n	

$$T(n) = C_1 + C_2 + C_3 + C_4(n+1) + C_5(n) + (C_6(n)) + C_7(n) + C_8(n) + C_9(1) + C_{10}(n) + C_{11}(n-1) + C_{12}(1) + C_{13}(0) + C_{14}(0) + C_{15}(n+1) + C_{16}(n)$$

$$= (C_5 + C_6 + C_4 + C_7 + C_8 + C_{10} + C_{11} + C_{15})n + (C_1 + C_2 + C_3 + C_4 + C_9 + (-C_{11}) + C_{12} + C_{15})$$

$$= an + b$$

$$= O(n)$$

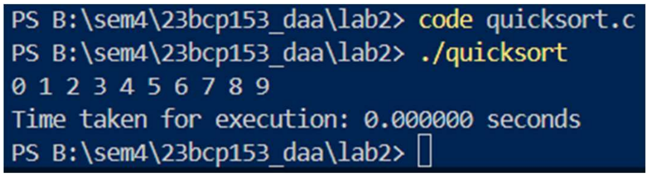
worst case $\overset{i}{\boxed{1\ 3\ 5\ 7}}$ $\overset{j}{\boxed{2\ 4\ 6\ 8}}$ or $\overset{i}{\boxed{2\ 4\ 6\ 8}}$ $\overset{j}{\boxed{1\ 3\ 5\ 7}}$

Code:

Quick Sort:

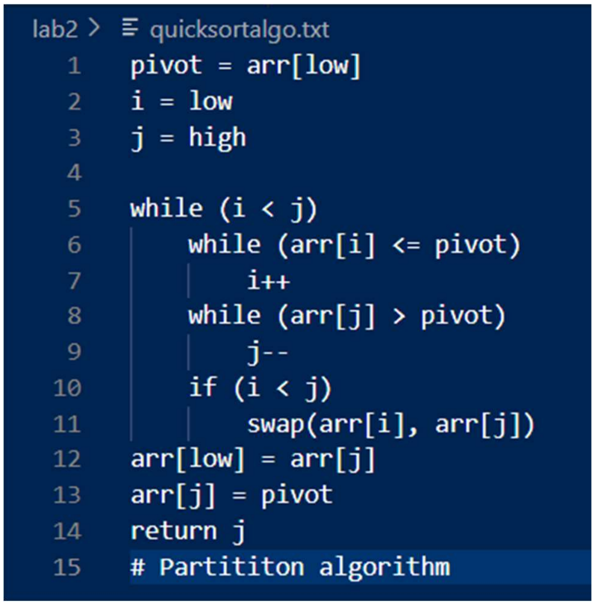
```
#include <stdio.h>
#include <time.h>

void quick_sort(int arr[], int low, int high);
int partition(int arr[], int low, int high);

int main(void)
{
    clock_t start, end;
    int arr[] = {7, 4, 8, 9, 0, 1, 2, 5, 3, 6};

    int len = sizeof(arr) / sizeof(int);
    int low = 0;
    int high = len - 1;
    start = clock();
    quick_sort(arr, low, high);
    end = clock();

    for (int i = 0; i < len; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("Time taken for execution: %f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

    return 0;
}

void quick_sort(int arr[], int low, int high)
{
    if (low < high)
    {
        int location = partition(arr, low, high);
        quick_sort(arr, low, location - 1);
        quick_sort(arr, location + 1, high);
    }
}

int partition(int arr[], int low, int high)
{
    int pivot = arr[low];
    int i = low; // i is start in lab algo
    int j = high; // j is end in lab algo

    while (i < j)
    {
        while (arr[i] <= pivot)
            i++;
        while (arr[j] > pivot)
            j--;

        if (i < j)
        {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    arr[low] = arr[j];
    arr[j] = pivot;

    return j;
}
```

Output:



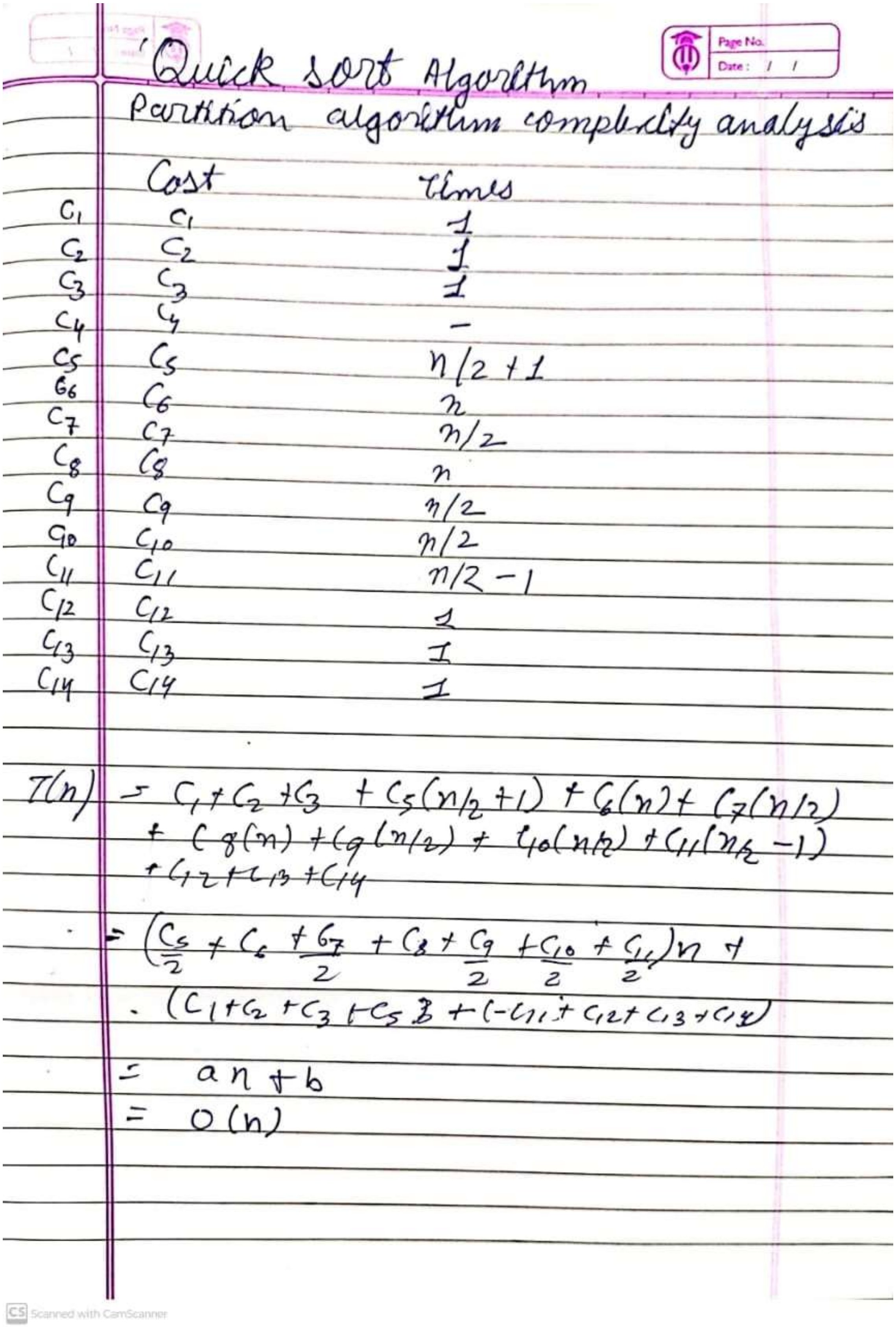
```
PS B:\sem4\23bcp153_daa\lab2> code quicksort.c
PS B:\sem4\23bcp153_daa\lab2> ./quicksort
0 1 2 3 4 5 6 7 8 9
Time taken for execution: 0.000000 seconds
PS B:\sem4\23bcp153_daa\lab2> 
```

Algorithm:



```
pivot = arr[low]
i = low
j = high

while (i < j)
    while (arr[i] <= pivot)
        i++
    while (arr[j] > pivot)
        j--
    if (i < j)
        swap(arr[i], arr[j])
arr[low] = arr[j]
arr[j] = pivot
return j
# Partititon algorithm
```

Complexity Analysis:



Page No.
Date : / /

Quick sort Algorithm

Partition algorithm complexity analysis

	Cost	Times
C_1	C_1	1
C_2	C_2	1
C_3	C_3	1
C_4	C_4	—
C_5	C_5	$n/2+1$
C_6	C_6	n
C_7	C_7	$n/2$
C_8	C_8	n
C_9	C_9	$n/2$
C_{10}	C_{10}	$n/2$
C_{11}	C_{11}	$n/2-1$
C_{12}	C_{12}	1
C_{13}	C_{13}	1
C_{14}	C_{14}	1

$$T(n) = C_1+C_2+C_3+C_5(n/2+1)+C_6(n)+C_7(n/2) + C_8(n)+C_9(n/2)+C_{10}(n/2)+C_{11}(n/2-1) + C_{12}+C_{13}+C_{14}$$

$$= \left(\frac{C_5}{2}+C_6+\frac{C_7}{2}+C_8+\frac{C_9}{2}+\frac{C_{10}}{2}+\frac{C_{11}}{2}\right)n + (C_1+C_2+C_3+C_5) + (-C_{11}+C_{12}+C_{13}+C_{14})$$

$$= an+b$$

$$= O(n)$$

EXPERIMENT 3

20CP209P – Design and Analysis of Algorithm Lab

Aim:

Code:

Addition:

Output:

Algorithm:

Complexity Analysis:

Code:

Subtraction:

Output:

Algorithm:

Complexity Analysis:

Code:

Multiplication:

Output:

Code:

Exponential:

Output: