# **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--;

        }

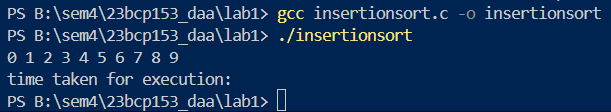
        arr[j + 1] = key;

    }

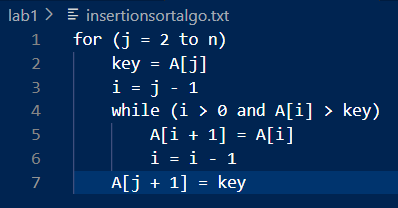
    return;

}

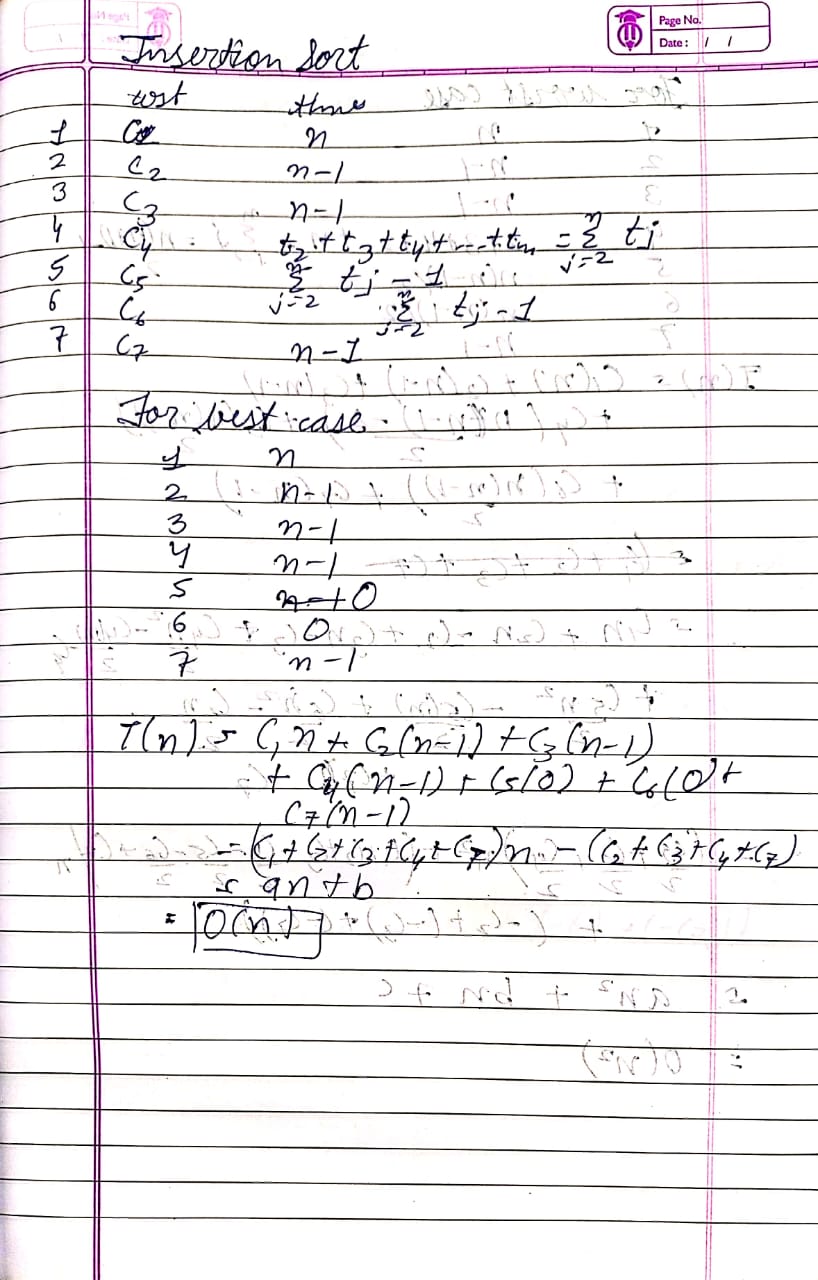
## **Output:**

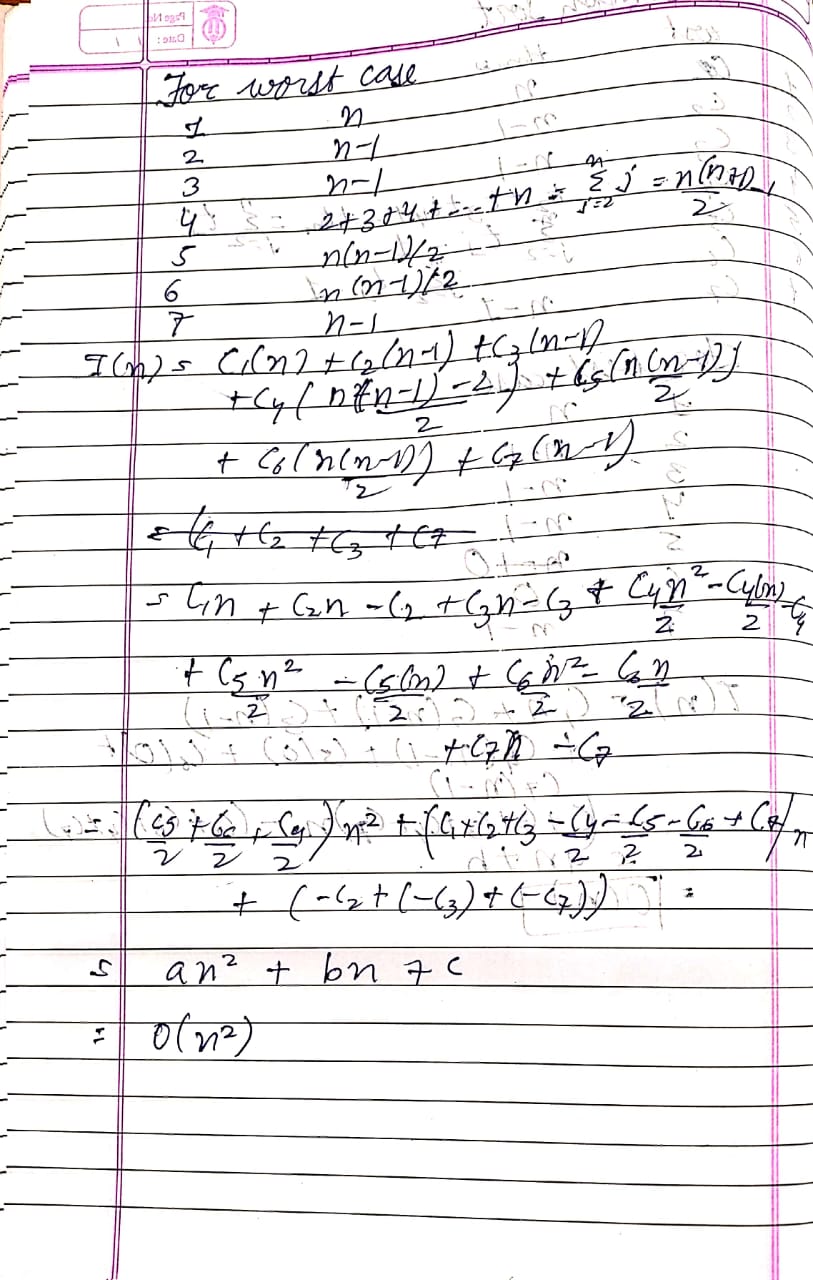


## **Algorithm:**



## **Complexity Analysis:**





## **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;

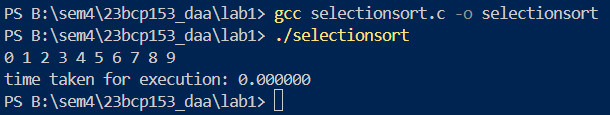
        }

    }

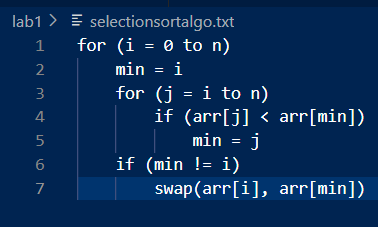
    return;

}

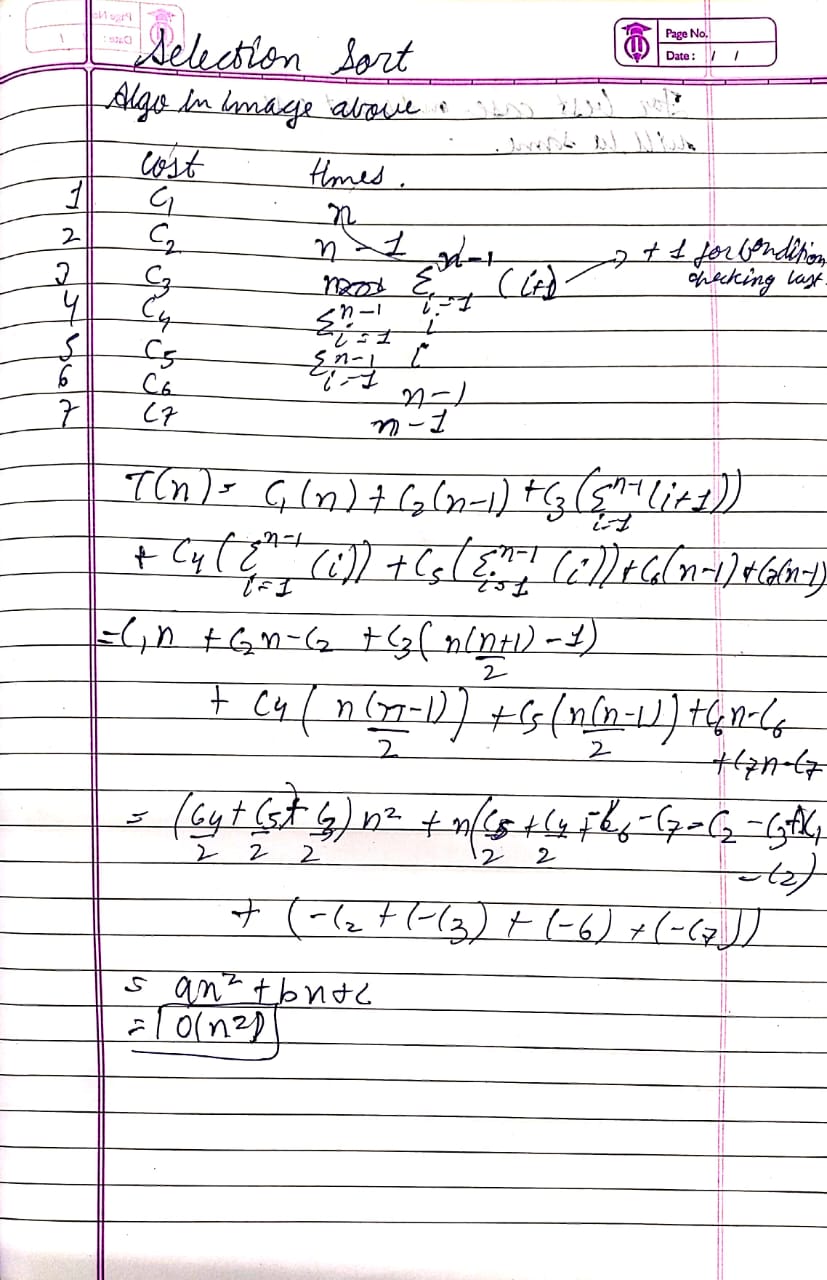
## **Output:**

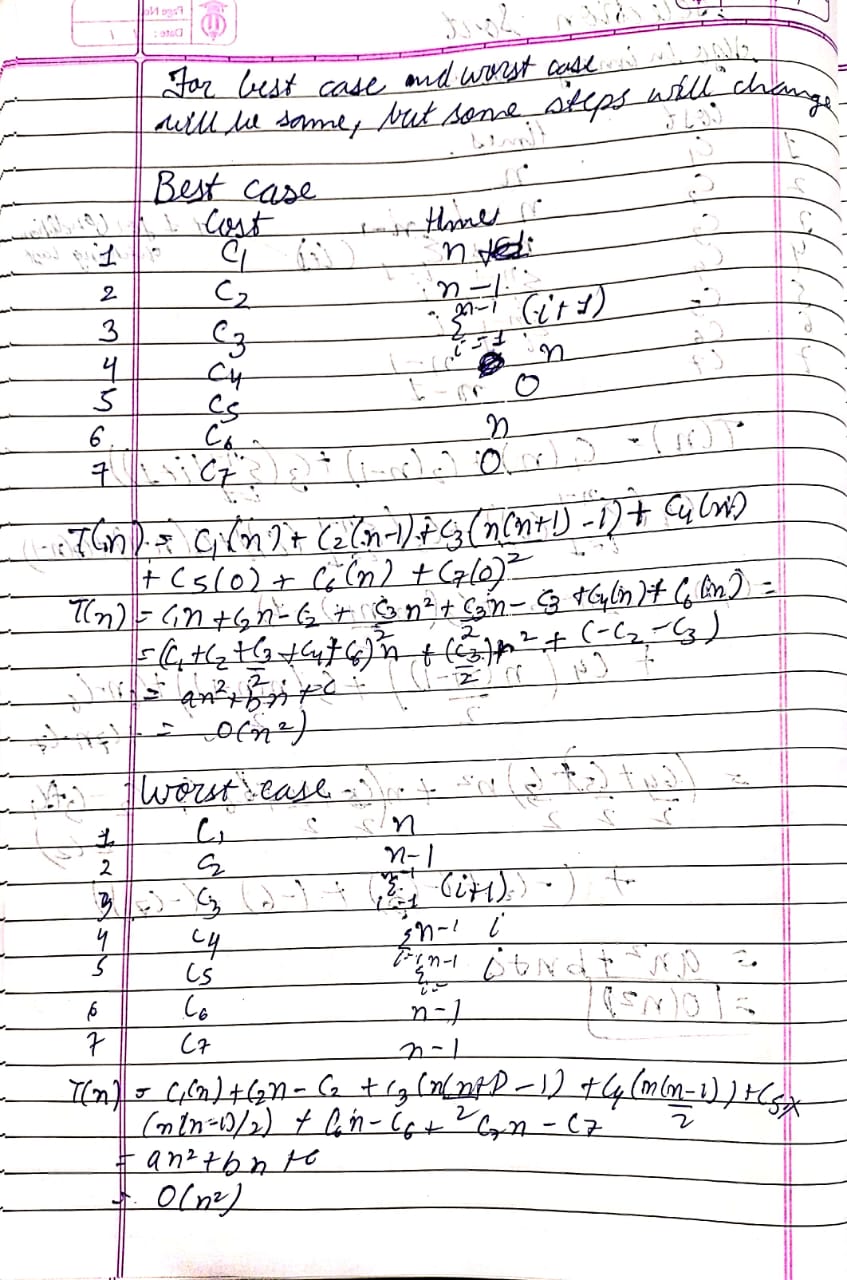


## **Algorithm:**



## **Complexity Analysis:**





# **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++;

        }

    }

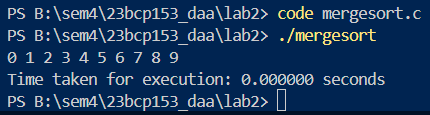
    for (int x = 0; x < (high - low + 1); x++)

        arr[low + x] = arrB[x];

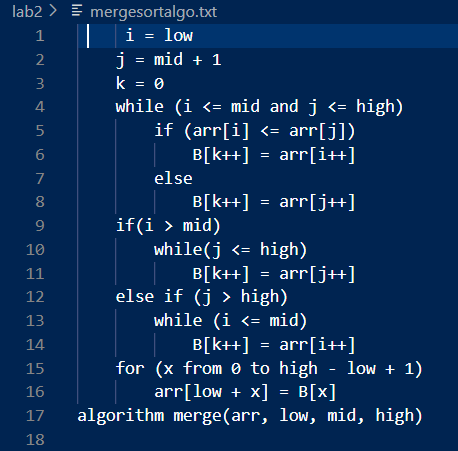
    return;

}

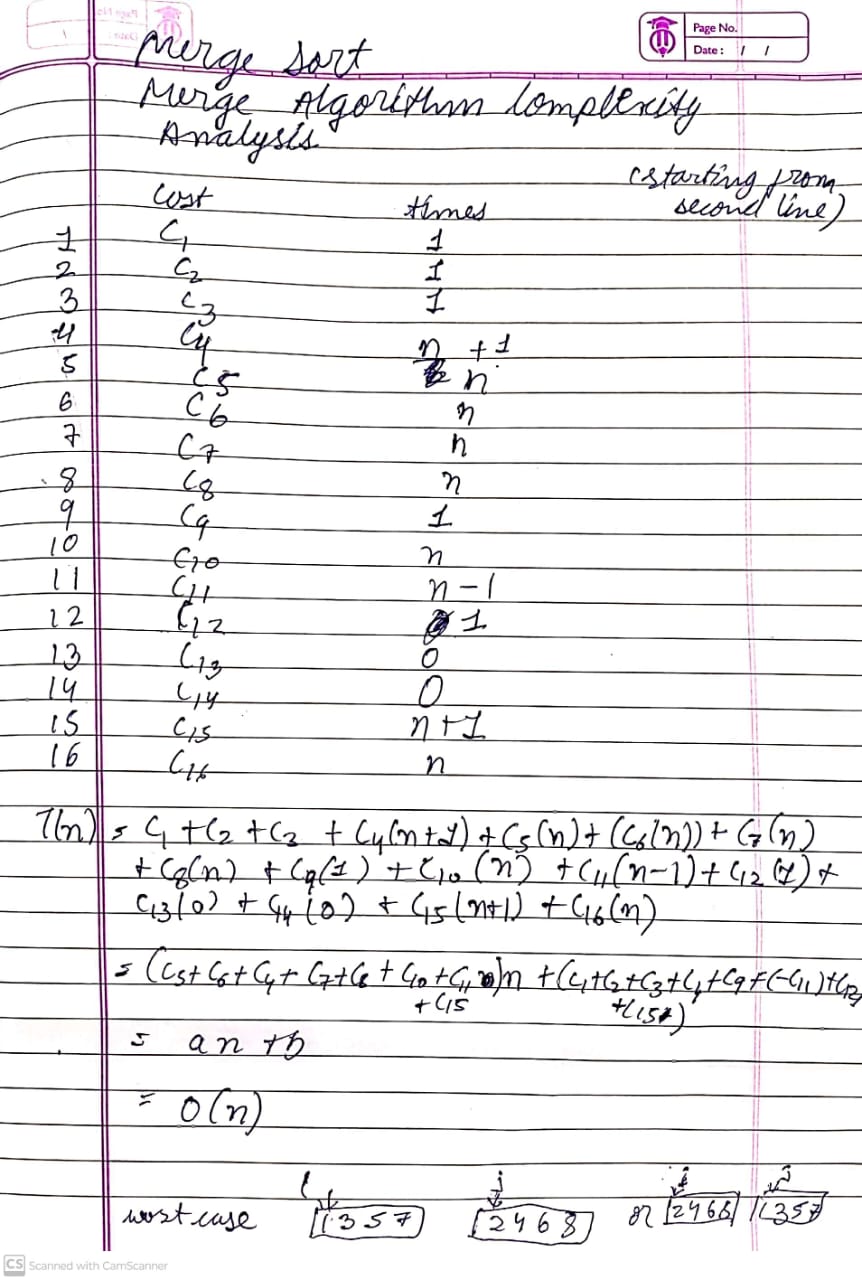
## **Output:**



## **Algorithm:**



## **Complexity Analysis:**



## **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;

        }

    }

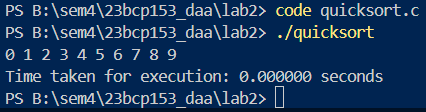
    arr[low] = arr[j];

    arr[j] = pivot;

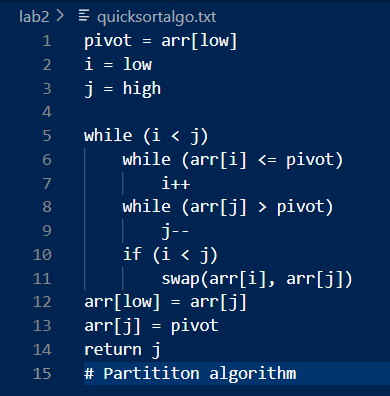
    return j;

}

## **Output:**



## **Algorithm:**



## **Complexity Analysis:**

