Q.1) Implement two different sorting algorithms bubble sort and quick sort measure and compare the execution time for different input size and calculate and compare the time complexity of these trends.

#include<stdio.h>

#include <iostream.h>

#include <conio.h>

#include <stdlib.h>

#include <dos.h>

#include <time.h>

#include <graphics.h>

void swap(int &a, int &b) {

    int temp = a;

    a = b;

    b = temp;

}

class SortingAlgorithms {

public:

    void bubbleSort(int arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

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

                if (arr[j] > arr[j + 1]) {

                    swap(arr[j], arr[j + 1]);

                }

            }

        }

    }

    int partition(int arr[], int low, int high) {

        int pivot = arr[high];

        int i = low - 1;

    for (int j = low; j < high; j++) {

            if (arr[j] < pivot) {

                i++;

                swap(arr[i], arr[j]);

            }

        }

        swap(arr[i + 1], arr[high]);

        return i + 1;

    }

    void quickSort(int arr[], int low, int high) {

        if (low < high) {

            int pi = partition(arr, low, high);

            quickSort(arr, low, pi - 1);

            quickSort(arr, pi + 1, high);

        }

    }

};

void generateRandomArray(int arr[], int size) {

    for (int i = 0; i < size; i++) {

        arr[i] = rand() % 10000;

    }

}

void drawGraph(double bubbleTimes[], double quickTimes[], int sizes[], int count) {

    int gd = DETECT, gm;

    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

    setcolor(WHITE);

    outtextxy(250, 20, "Shiv Arora");

    outtextxy(150, 40, "Sorting Algorithm Time Complexity Comparison");

    outtextxy(50, 410, "Input Size");

    outtextxy(20, 200, "Time (scaled)");

    int scale = 2000; *// Moderate scaling factor for visibility*

    int barWidth = 40, spacing = 120, x = 100;

    for (int i = 0; i < count; i++) {

        int bubbleHeight = bubbleTimes[i] \* scale;

        int quickHeight = quickTimes[i] \* scale;

*// Bubble Sort bar*

        setfillstyle(SOLID\_FILL, RED);

        bar(x, 400 - bubbleHeight, x + barWidth, 400);

        outtextxy(x, 405, "Bubble");

*// Quick Sort bar*

        setfillstyle(SOLID\_FILL, BLUE);

        bar(x + barWidth + 10, 400 - quickHeight, x + 2 \* barWidth + 10, 400);

        outtextxy(x + barWidth + 10, 405, "Quick");

*// Input size label*

        char sizeLabel[10];

        sprintf(sizeLabel, "%d", sizes[i]);

        outtextxy(x + barWidth / 2, 430, sizeLabel);

        x += spacing;

    }

    getch();

    closegraph();

}

void main() {

    clrscr();

    cout<< "Shiv Arora"<< endl;

    SortingAlgorithms sorter;

    int sizes[] = {100, 1000, 5000, 10000};

    int repetitions[] = {1000, 100, 10, 1}; *// corresponding repetitions*

    clock\_t start, stop;

    double bubbleTimes[4], quickTimes[4];

    for (int i = 0; i < 4; i++) {

        int size = sizes[i];

        int repeat = repetitions[i];

        double totalBubble = 0, totalQuick = 0;

        for (int r = 0; r < repeat; r++) {

            int\* arr = new int[size];

            int\* arrCopy = new int[size];

            generateRandomArray(arr, size);

            for (int j = 0; j < size; j++) {

                arrCopy[j] = arr[j];

            }

            start = clock();

            sorter.bubbleSort(arr, size);

            stop = clock();

            totalBubble += (double)(stop - start) / CLK\_TCK;

            start = clock();

            sorter.quickSort(arrCopy, 0, size - 1);

            stop = clock();

            totalQuick += (double)(stop - start) / CLK\_TCK;

            delete[] arr;

            delete[] arrCopy;

        }

        bubbleTimes[i] = totalBubble / repeat;

        quickTimes[i] = totalQuick / repeat;

        cout << "Average Bubble Sort time for n = " << size << " : " << bubbleTimes[i] << " sec\n";

        cout << "Average Quick Sort time  for n = " << size << " : " << quickTimes[i] << " sec\n\n";

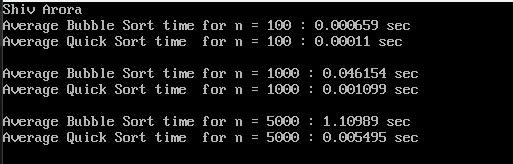
    }

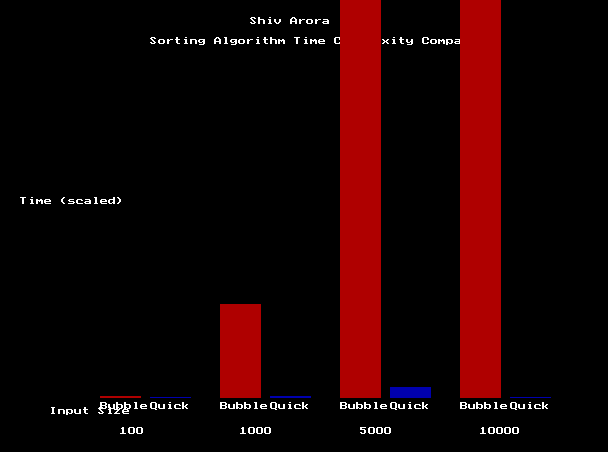
    drawGraph(bubbleTimes, quickTimes, sizes, 4);

    getch();

}

OUTPUT:





Q.2) Write a simple programme that compares three functions print their computed values for increasing values of and classify them under **O, Ω, Θ** notations

#include <iostream.h>

#include <conio.h>

#include <math.h>

#include <graphics.h>

class AsymptoticComparison {

public:

    int function1(int n) { return n \* n; }

    float function2(int n) { return n \* log(n); }

    int function3(int n) { return n; }

    void plotGraph(int start, int end) {

        int gd = DETECT, gm;

        initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

        int x = 50, y\_base = 400, prev\_x = -1;

        int prev\_y1 = -1, prev\_y2 = -1, prev\_y3 = -1;

        for (int i = start; i <= end; i \*= 2) {

            int y1 = y\_base - function1(i) / 100; *// scale O(n^2)*

            int y2 = y\_base - function2(i) / 10; *// scale O(n log n)*

            int y3 = y\_base - function3(i); *// scale O(n)*

            putpixel(x, y1, BLUE); *// O(n^2)*

            putpixel(x, y2, GREEN); *// O(n log n)*

            putpixel(x, y3, RED); *// O(n)*

            if (prev\_x != -1) {

                setcolor(BLUE);

                line(prev\_x, prev\_y1, x, y1); *// O(n^2)*

                setcolor(GREEN);

                line(prev\_x, prev\_y2, x, y2); *// O(n log n)*

                setcolor(RED);

                line(prev\_x, prev\_y3, x, y3); *// O(n)*

            }

            prev\_x = x;

            prev\_y1 = y1;

            prev\_y2 = y2;

            prev\_y3 = y3;

        x += 10;

        }

        setcolor(WHITE);

        outtextxy(50, 420, "Red: O(n), Green: O(n log n), Blue: O(n^2)");

        outtextxy(50, 450, "Shiv Arora");

        getch();

        closegraph();

    }

};

int main() {

    clrscr();

    AsymptoticComparison comparer;

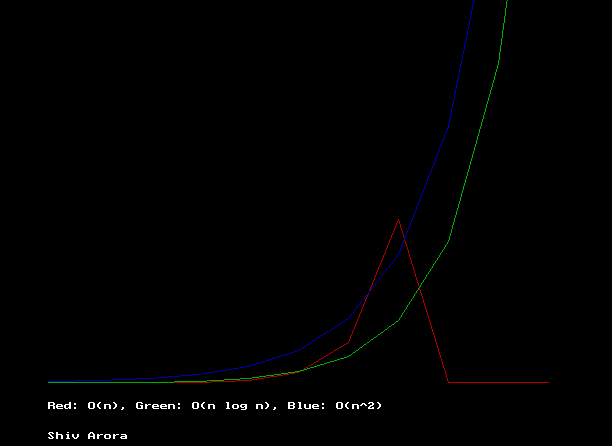
    comparer.plotGraph(1, 1100);

    getch();

    return 0;

}

OUTPUT:



Q.3) Solve the recurrence relations using master method and substitution method.

#include <iostream.h>

#include <conio.h>

#include <stdlib.h>

void main(){

clrscr();

cout<< "Shiv Arora" << endl;

int choice;

cout<<"Choose one of the following options: " << endl;

cout<< "1) T(n) = 2T(n/2) + n" << endl;

cout<< "2) T(n) = T(n/2) + 1" << endl;

cin >> choice;

int method;

cout<<"Choose one of the following methods: " << endl;

cout<< "1) Substitution Method" << endl;

cout<< "2) Master Theorem" << endl;

cin>> method;

if(choice == 1 && method == 1) {

cout << "T(n) = 2T(n/2) + n" << endl;

cout << "Assume T(n) = nlogn" << endl;

cout << "T(n) = 2T(n/2) + n" << endl;

cout << "T(n) = 2(n/2)log(n/2) + n" << endl;

cout << "T(n) = nlog(n/2) + n" << endl;

cout << "T(n) = n(logn - log2) + n" << endl;

cout << "T(n) = nlogn - nlog2 + n" << endl;

cout << "T(n) = nlogn - n + n" << endl;

cout << "T(n) = Theta(nlogn)" << endl;

}

else if(choice == 1 && method == 2){

cout<< "Master Theorem: T(n) = aT(n/b) + f(n)"<<endl;

cout<< "a = 2, b = 2, f(n) = n"<<endl;

cout<< "n^log\_b(a) = n^log\_2(2) = n^1"<<endl;

cout<< "as f(n) = n^log\_b(a)"<<endl;

cout<<"Case 3: f(n) = n^log\_b(a) => T(n) = Theta(f(n) \* logn)"<<endl;

cout << "T(n) = Theta(nlogn)" << endl;

}

else if(choice == 2 && method == 1){

cout << "T(n) = T(n/2) + 1" << endl;

cout << "Assume T(n) = 1" << endl;

cout << "T(n) = T(n/2) + 1" << endl;

cout << "T(n) = T(n/2) + 1" << endl;

cout << "T(n) = T(n/4) + 1 + 1" << endl;

cout << "T(n) = T(n/8) + 1 + 1 + 1" << endl;

cout << "T(n) = T(n/2^k) + k" << endl;

cout << "T(n) = T(1) + log(n)" << endl;

cout << "T(n) = 1 + log(n)" << endl;

cout << "T(n) = Theta(logn)" << endl;

}

else if(choice == 2 && method == 2){

cout<< "Master Theorem: T(n) = aT(n/b) + f(n)"<<endl;

cout<< "a = 1, b = 2, f(n) = 1"<<endl;

cout<< "n^log\_b(a) = n^log\_2(1) = n^0 = 1"<<endl;

cout<< "as f(n) = n^log\_b(a)"<<endl;

cout<<"Case 3: f(n) = n^log\_b(a) => T(n) = Theta(f(n) \* logn)"<<endl;

cout << "T(n) = Theta(logn)" << endl;

}

else{

cout << "Invalid choice" << endl;

}

cout << "End of program" << endl;

getch();

}

OUTPUT:

