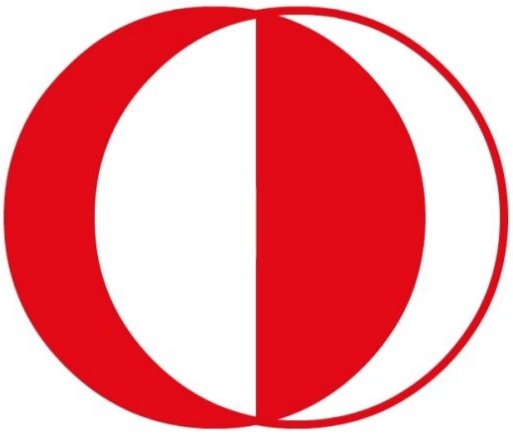

METU
NORTHERN CYPRUS
CAMPUS

CNG 315 - Algorithms
Fall 2017

Take Home Programming Assignment #1
Report

Furkan Tokaç
e2016145

1 Algorithms, Analysis

1.1 Question 1, Part A, Version 1

In the question 1, we have 3 dimensional rectangle data which is stored the values.txt file. In the each point of the rectangle, there is a value stored. Firstly, we read the data from text file and store values of corresponding points in the 3 dimensional array. After that, we get input from the user or a text file which is 2 points on the 3 dimensional rectangle. These 2 points create subrectangle in the main rectangle and we have to calculate all the point covered by that subrectangle.

In the first version of the algorithm, I wrote basic algorithm that comes to mind first. There are 3 nested loops and in the each loop, algorithm visits a point and add its value to the “sum” variable and at the and, function returns the “sum” variable.

Complexity of this algorithm is basically;
N = x*y*z
O(N)

1.2 Question 1, Part A, Version 2

I wanted to upgrade previous algorithm so I made a version 2 for the algorithm. In this algorithm, basically I have a reference point which is x. I have just 1 main loop and inside the main loop, I have 2 small loops. Let’s say, user entered the coordinates (0,0,0) and (1,2,3) and we have to calculate subrectangle that is starts with (0,0,0) and ends with (1,2,3). In my algorithm, my first reference point is (0,0,0) and then I have a loop. In the loop, I visit all the points along the way y which are (0,0,0), (0,1,0), (0,2,0) and add their values to the “sum” variable. Then, visit all the z points according to reference point which are (0,0,1), (0,0,2), (0,0,3). After that, I increase y and z values of the reference point and reference point become (0,1,1). By this move, I move inner part of the 2d rectangle formed by y and z axises. This continues till the reference point becomes (0,2,3) which is limit of the 2d rectangle created by y and z axises. After that, I increase x and moving the next 2d rectangle which is (1,0,0) and continue the same process.

As you can understand, in this algorithm, I divide 3d rectangle to the 2 dimensions by y and z axis and calculating them. Thanks to this algorithm, complexity decreases and becomes;

If y > z : n/z = x*y which is O(N/z)
If z > y : n/y = x*z which is O(N/y)

1.3 Question 1, Part B

In this algorithm, we have to create new array which is sum of the points from (0,0,0) to till the corresponding point. I used the same logic that I used in question 1, part a, version 1. I visited all the points and calculated values till the that points by using algorithm in the question 1, part a, version 2.

Although we use more memory in this algorithm, it can decrease the complexity of part a to O(1). Because of we have all the sums, we can calculate any sum of the subrectangle without loop by just subtraction, adding the values in sumbox array.

1.4 Question 2

In this question, basically we have a set of values and firstly, we have to distort it. Then we have to write an algorithm to sort it in an efficient way. This part tells my algorithm to sort this distorted array.

After thinking about lots of different algorithms, I focused on already sorted values in the list to decrease complexity. Let's say, we have numbers;

9 8 10 7 11 2 4 5 6

We have an array to store new sorted list which's size is 1000 and its name is sortedList. According to my algorithm, first value becomes the middle number and placed to the middle of the array. (SIZE/2)

9

The number that comes after it goes it's right if it is smaller than middle number, goes left if it is bigger than the middle number. In my case, steps will be like;

(8 goes left) 8 **9**
(10 goes right) 8 **9** 10
(7 goes left) 7 8 **9** 10
(11 goes right) 7 8 **9** 10 11
(2 goes left) 2 7 **9** 10 11

Now the problematic part comes. We have to insert an element between 2 and 7 but insertion is a problematic operation so insted of insertion, we move 2 into different place in the array and write the index of the place as negative. Let's say we move 2 to the beginning of the sorted list which is 2 (we ignore 0 and 1) so new list becomes

-2 7 9 10 11 AND 2

And we insert the new value to the right of the 2

-2 7 **9** 10 11 AND **2** 4
(5 goes right of the 2) -2 7 **9** 10 11 AND **2** 4 5
(6 goes right of the 2) -2 7 **9** 10 11 AND **2** 4 5 6

So if we have extra value which is 1, how to insert it ? Actually before adding the, let's say 5, it is compared with 2 and decided if it is less than 2. If it is not, it is added to right of the 2. If it is, the new values will be like;

(1 goes left of the 9) 1 -2 **9** 10 11 AND **2** 4 5 6

Criticism

1. In this example, I assume that there will be no negative numbers.
2. In the worst case, size of the sorted list must be (number of numbers)*3. (if we have 9 numbers, sorted array size must be at least 27).
3. If 3 will be added in the final step, there will be problem. Looking for an efficient way to solve this problem.
4. If after After sorted array created, it will not be directly sorted along the way. Although it can be used as it is without any problem, if programmer wants directly sorted list, there should be another loop with the complexity O(N) to visit all the values and save them in an array.
5. In the current version of the algorithm, it is assumed that there will be no duplicate values in the distorted list. With the improvement, this may be allowed in the next versions on the algorithm. It will not be a big problem I think.
6. Other than the limitations above, there is no limitations (as I can see) as long as the indexes arranged wisely.
7. If linkedlist is used in this algorithm, it can dramatically improve the quality of algorithm.

This algorithm will be efficient in the situations like if anybody thinks that there is already sorted values in the array. Also, in the beginning of the algorithm, instead of choosing first elements as middle value, it can be find to middle value with another algorithm. This will also eliminate the problem in the criticism 3 and dramatically decrease the complexity. For this algorithm, complexity values are;

Worst case complexity : O(N^2-1)
Best case complexity : O(N)

2 Pseudo Codes of Algorithms

2.1 Question 1, Part A, Version 1

```
int partA_v1(const int values[][SIZE][SIZE],
             const Coordinate cord1,
             const Coordinate cord2)
{
    Coordinate cord;
    int sum=0;

    for(cord.x=cord1.x; cord.x<=cord2.x; cord.x+=1)
        for(cord.y=cord1.y; cord.y<=cord2.y; cord.y+=1)
            for(cord.z=cord1.z; cord.z<=cord2.z; cord.z+=1)
                sum += values[cord.x][cord.y][cord.z];

    return sum;
}
```

2.2 Question 1, Part A, Version 2

```
int partA_v2(const int values[SIZE][SIZE][SIZE],
             const Coordinate cord1,
             const Coordinate cord2)
{
    int sum=0;
    Coordinate cordRef;
    Coordinate cord;

    cordRef = cord1;

    while(cordRef.x <= cord2.x)
    {
        for(cord.y=cordRef.y; cord.y<=cord2.y; cord.y+=1)
            sum += values[cordRef.x][cord.y][cordRef.z];
        cord.y = cordRef.y;

        for(cord.z=cordRef.z+1; cord.z<=cord2.z; cord.z+=1)
            sum += values[cordRef.x][cordRef.y][cord.z];
        cord.z = cordRef.z;

        cordRef.y += 1;
        cordRef.z += 1;

        if(cordRef.y>cord2.y || cordRef.z>cord2.z )
            cordRef = (cordRef.x+1, cord1.y, cord1.z);
    }
    return sum;
}
```

2.3 Question 1, Part B

```
void fillBoxsumArray(int boxsum[SIZE][SIZE][SIZE],
                     const int values[SIZE][SIZE][SIZE],
                     const Coordinate dimensions)
{
    Coordinate cord;
    Coordinate cordZero;

    cordZero = (0,0,0);

    for(cord.x=0; cord.x<=dimensions.x; cord.x+=1)
        for(cord.y=0; cord.y<=dimensions.y; cord.y+=1)
            for(cord.z=0; cord.z<=dimensions.z; cord.z+=1)
                boxsum[cord.x][cord.y][cord.z] = getSumOfPoint(cordZero,cord);
}
```

2.4 Question 2 (Not completed yet)

```
void mySortingAlgorithm(const int array[],
                        const int size,
                        int sorted[])
{
    int leftestIndex = 0, rightestIndex = 0;
    const int middleIndex = SIZE/2;

    sorted[ middleIndex ] = array[0];
    leftestIndex = middleIndex;
    rightestIndex = middleIndex;

    for(int i=1; i<size; i++)
    {
        if( array[i]>sorted[middleIndex] )
        {
            if( array[i]>sorted[rightestIndex] )
            {
                rightestIndex++;
                sorted[rightestIndex] = array[i];
            }
        }
        else
        {
            if( array[i]<sorted[leftestIndex] )
            {
                leftestIndex--;
                sorted[leftestIndex] = array[i];
            }
        }
    }

    sorted[leftestIndex-1] = -1;
    sorted[rightestIndex+1] = -1;
}
```