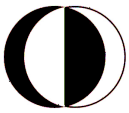

MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

CNG 315 - Algorithms - Fall 2017
Take Home Programming Assignment #1

Due on November 13, 2017

Below you will find two programming exercises. For the exercises below you need to submit a **well-commented C source code and PDF report**. Here are some suggestions and requirements:

- You need to use **ANSI C** for your implementation.
- Use Linux system and gcc if you can (I am using gcc!).
- Arrays and loops would be enough. **Do not use** pointers, **"`malloc`"**, or recursion!
- Customize your code according to the sample runs given in the exercises for the input-output format (try them out!).
- Sample data file (**"`values.txt`"**) and instructions (**"`README.txt`"**) will be given for the Problem 1.
- Comment your code and put your name, last name, SID at the beginning of your code.
- Analyze the time and space complexity of your algorithms and try to be as efficient as possible. For example if you find n^2 algorithm try to make it $nlog(n)$
- Let's name the source code for the first problem **"`a1.c`"** and for the second **"`a2.c`"**.
- In your report (must be PDF, like "report.pdf"), for each problem:
  1. Explain your algorithm with words.
  2. Explain your algorithm with a pseudo code.
  3. Give analysis on the space and time complexity of your algorithm.
- The best way to submit your homework:
  1. Create a directory with your first name.
  2. Put "`a1.c`" and "`a2.c`" into that directory.
  3. Put your "report.pdf" into that directory.
  4. Zip the directory.
  5. Submit the zip to "odtuclass".

Grading, Total is 100 Pts		
Problem No	1	2
Code	35 Pts	35 Pts
Report	15 Pts	15 Pts
Total	50 Pts	50 Pts

Problem 1

(a) Given the 3D matrix with integers having values in the 0-255 range (call it "*values*"), design an algorithm to find the sum of the integers in the "bounding box" specified by the coordinates of the opposite corners $((x_{min}, y_{min}, z_{min}), (x_{max}, y_{max}, z_{max}))$, where $(x_{min}, y_{min}, z_{min})$ is the closest point to origin $(0, 0, 0)$, and $(x_{max}, y_{max}, z_{max})$ is the point farthest from the origin. For simplicity assume that $(x_{max} \geq x_{min}, y_{max} \geq y_{min}, z_{max} \geq z_{min})$. Also further assume that the max size of the 3D matrix is $k \times l \times n$ and dimensions need not be equal (like cube) where:

- k, l, n being dimensions, $3 \leq k, l, n \leq 100$
- k, l, n can be all equal or all different
- $0 \leq values[x][y][z] \leq 255$, where $0 \leq x \leq k-1$, $0 \leq y \leq l-1$, $0 \leq z \leq n-1$.

(b) After part (a), utilize your algorithm to generate another matrix, call it "*boxsum*", having the same size with "*values*", $(k \times l \times n)$. The $boxsum[x][y][z]$ will contain the sum of integers in the bounding box specified by $((0, 0, 0), (x, y, z))$. That is from the origin (if you think like 3D coordinates) to every point. The figure 1 below shows you the 3D bounding box given by coordinates as $((0, 0, 0), (1, 1, 1))$. The green balls are representing the integer numbers! The $boxsum[1][1][1]$ will be the sum of the integers in the figure. That is (143+191+70+241+62+252+102+253) = 1314. Similarly:

- $boxsum[0][0][0] = 143$ (Just a point, 1D sum!)
- $boxsum[0][0][1] = 143 + 241 = 384$ (Two points, rather a thin "box", 2D sum!)
- $boxsum[1][1][0] = 143 + 191 + 62 + 252 = 648$ (4 points on the xy plane, the "box" is thin, just 1 unit wide in the z direction. Now we have "real" 3D sum!)

Then by using "*boxsum*" matrix, can you improve your algorithm in part **(a)**?

Please give the code and analysis for both **(a)** and **(b)**!

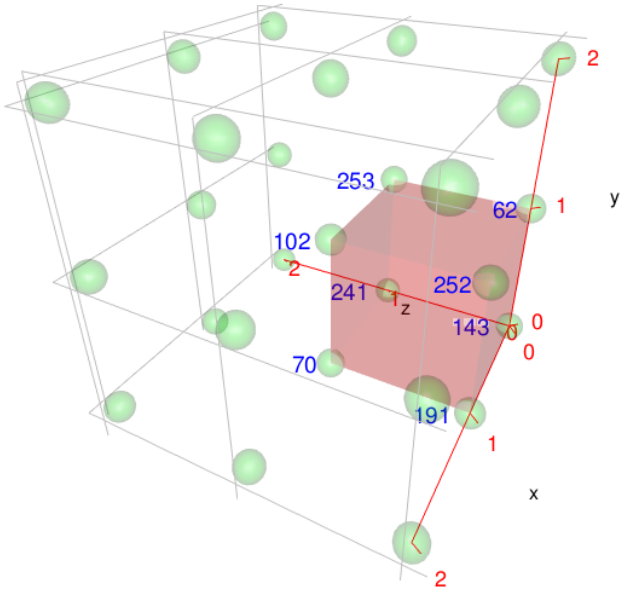

Figure 1: The 3D bounding box

Problem 2

In this exercise, firstly your program will read a sorted (increasing order, that is first element is min and the last is max) array of positive integers (they are big numbers huh!!!) from the user. The array will have N (where $10 \leq N \leq 2000$) elements. Then your program will distort the array a bit for the next phase in which you should design a cool sorting algorithm for this "distorted array"! The distortion goes like this:

1. Pick a random position t, where $3 \leq t \leq N-3$ for the "pivot element" which will be the t^{th} element of the array.
2. Then place the pivot element as a first element into new but "distorted" array.
3. After that start placing the other elements to the "new distorted" array by taking one from the left and one from the right of the pivot element till either left or right elements finished. The order is important: Left, right, left, right,... Like the marching soldiers!
4. Finally, when either left or right wing elements finished, copy the remaining elements to the "new distorted" array
5. Now you are ready to find a cool sorting algorithm for this "new distorted" array!
6. **Note that your sorting algorithm does not know where was the pivot element!!!**

Here is a sample run:

```
How many elements [10..2000]: 8
Enter the array elements: 2 4 5 6 7 8 9 10
Pivot is chosen as the element with index 2 which is 5
Distorted array: 5 4 6 2 7 8 9 10
Sorted array: 2 4 5 6 7 8 9 10
How many elements [10..2000]: 9
Enter the array elements: 2 4 5 6 7 8 9 10 11
Pivot is chosen as the element with index 6 which is 9
Distorted array: 9 8 10 7 11 2 4 5 6
Sorted array: 2 4 5 6 7 8 9 10 11
```

(**Warning:** Remember in C language the index value of the first array element is 0!)

Good luck!!!