CS 240: Data Structure
Problem Set 3

Due: 04/26/2020

Instructions:
Please follow the instruction given at each problem for submission. The deadline of each assignment is 11:59 pm on the due date, unless otherwise specified. It is important that your code/solution is straight forward, not cumbersome. At the beginning of each cpp file, you must include the following:
1. Description: description of the program (later we will elaborate this more, but for now, just describe what the program does)
2. Author: the person who writes this program
3. Date: the very last date/time the program is modified

First Name:

Last Name:

Score: **/130**

Problem 1 Recursion + Complexity: 10 + (10+10) pts

In chapter 10 we discussed about the top down method for the Fibonacci function and we concluded that its recursion implementation is not a good approach.
(a) Please explain why the implementation is inefficient and give it an estimate on its complexity.

(b) Can you design and implement another approach such that your implementation has a better complexity? Please implement your own Fibonacci function and please justify its efficiency by providing the complexity.

Problem 2 Recursion: 20pts

(1) Recall in your MAT 115 class you learned about GCD (greatest common divisor). Please write a recursive function GCD(int a, int b) that computes the gcd of a and b.

(2) Please write a recursive function VectReverse to reverse a vector of type T. [Hint: vector has size function, and template helps with a generic type]

Problem 3 Sparse matrix : 10+10+20+10 pts

We explained the cumbersome implementation of a sparse matrix structure in doubly linkedlist. One of the major problem is random memory access due to the nature constraint of linkedlist (too much freedom and hard to calculate the exactly location of a particular item with one shot). We provided the bruteforce vector version of the implementation as an alternative. However, it is not ideal because in that approach it still loads all those zero entries into the matrix A. Matrix linear algebra is quite important in advanced computer science topics. To attack this problem, you have to do the following:

(1) Write a function $GenMatrix(int \quad n)$ that takes the input size n to generate a random double matrix A of size n by n while the number of nonzero entries in each row is $\lfloor \log n \rfloor$.

(2) Come up with a vector based container VectA that can efficiently represent your matrix A. (we want this VectA to be as small as possible)

(3) Generate a random vector x where $x = (x_1, ..., x_n)$ is of size n by 1 (it cannot be an all zero vector). and rewrite your operate* function that does $A * x$. let us call the resultant vector $B = (B_1, ..., B_n)$. Clearly B is also an n by 1 vector. Clearly, in the old operate* operation, $A * x$ involves n big operations as for entry B_i we know

$$B_i = \sum_{j=1}^{n} A_{ij} \times x_j$$

while each big operation takes n pairs of mutliplication plus $n - 1$. Hence, the total number of opreations is approximately $n * (2n - 1) \simeq 2n^2$. Please explain the complexity of your own operate* function when the input size of A is n by n.

(4) Finally, let assume each operation takes 1 second, and $n = 10^5$. How long will it take for the old operate* to finish the computation? And how long will it take for your operate*?

Problem 4 Chapter 12: BST Modification: 15 + 15 pts

Two parts for this problem. One is the coding that is 35 pts and one is the explanation that is worth 15 points.

Part I:
Please modify the BST code (with insertion, deletion) uploaded in blackboard to avoid un-necessary recursion calls in the deletion function. It is where they swap the found smallest element of the right subtree element and the matched element. Please figure how this step could have been eliminated. Also, at the bottom of the deletion function, it branches out to search left and right subtrees, which can also be avoided if we know the relation between the value to be removed and the value of current root.

Part II:
Please explain how and why (like giving concrete example situation) where your code will definitely outperform the given original code.