CSCD320 Homework2, Spring 2014, Eastern Washington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 20, 2014 (Sunday)

Please follow these rules strictly:

1. Write your name and EWUID on **EVERY** page of your submission.

2. Verbal discussions with classmates are encouraged, but each student must independently write his/her own solutions, without referring to anybody else's solution.

3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Blackboard system). Send in whatever you have by the deadline.

4. Submission must be computer typeset in the **PDF** format and sent to the Blackboard system. I encourage you all to use the LaTeX system for the typesetting, as what I am doing for this homework as well as the class slides. LaTeX is a free software used by publishers for professional typesetting and are also used by nearly all the computer science and math professionals for paper writing.

5. Your submission PDF file must be named as:
firstname_lastname_EWUID_cscd320_hw2.pdf
(1) We use the underline ' ' not the dash '-'.
(2) All letters are in the lower case including your name and the filename's extend.
(3) If you have middle name(s), you don't have to put them into the submission's filename.
6. Sharing any content of this homework and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

Problem 1 (16 points)**.** *By using the Master theorem, it's easy to get that* $T(n) = 2T\left(\frac{n}{4}\right) + n = \Theta(n) = O(n)$*. Prove this result using the inductive proof. That is, prove: there exist some positive constant* n_0 *and* c*, such that* $T(n) \leq cn$ *for all* $n \geq n0$*.* (Hint: any positive values for n_0 and c are good as long as they can make the proof work.)

n0 = 4, c = 2, T(1) = 1
Base: $T(4) = 2T(1) + 4 \leq 2 * 4 \rightarrow T(4) = 6 \leq 8$
Assume: $T(n) \leq cn$, when n = (4, 5, 6,...,k)
Hypothesis: $T(k+1) \leq 2(k+1)$

$$T(k+1) = 2T\left(\frac{k+1}{4}\right) + (k+1) \rightarrow T(k+1) \leq 2 * 2\left(\frac{k+1}{4}\right) + (k+1)$$

$$\rightarrow T(k+1) \leq (k+1) + (k+1) \rightarrow T(k+1) \leq 2(k+1)$$

Therefore: T(n) = O(n)

CSCD320 Homework2, Spring 2014, Eastern Washington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 20, 2014 (Sunday)

Problem 2 (24 points)**.** *Solve the following recurrences using the* Master Theorem*. (You can directly give the results.)*

1. $T(n) = 8T\left(\frac{n}{2}\right) + 3n^2 - 9n$

By Case 1 of the Master Theorem:

$T(n) = 8T\left(\frac{n}{2}\right) + 3n^2 - 9n = \Theta(n^3)$

2. $T(n) = 8T\left(\frac{n}{2}\right) + 2n^3 - 100n^2$

By Case 2 of the Master Theorem:

$T(n) = 8T\left(\frac{n}{2}\right) + 2n^3 - 100n^2 = \Theta(n^3 \log n)$

3. $T(n) = 4T\left(\frac{n}{2}\right) + n^2 + 5\log n$

By Case 2 of the Master Theorem:

$T(n) = 4T\left(\frac{n}{2}\right) + n^2 + 5\log n = \Theta(n^2 \log n)$

4. $T(n) = 8T\left(\frac{n}{2}\right) + n^3 + n\log n$

By Case 2 of the Master Theorem:

$T(n) = 8T\left(\frac{n}{2}\right) + n^3 + n\log n = \Theta(n^3 \log n)$

5. $T(n) = 8T\left(\frac{n}{2}\right) + 4n$

By Case 1 of the Master Theorem:

$T(n) = 8T\left(\frac{n}{2}\right) + 4n = \Theta(n^3)$

6. $T(n) = 4T\left(\frac{n}{2}\right) + 2^{-10}n^4 - 6n^3$

By Case 3 of the Master Theorem

$T(n) = 4T\left(\frac{n}{2}\right) + 2^{-10}n^4 - 6n^3 = \Theta(n^4)$

CSCD320 Homework2, Spring 2014, Eastern Washington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 20, 2014 (Sunday)

Problem3 (10 points)**.** *Propose TWO example recurrences that CANNOT be solved by the* Master Theorem*. Note that your examples must follow the shape that* T(n) = aT(n/b)+f(n)*, where* n *are natural numbers,* a ≥ 1, b > 1*, and* f(n) *is an increasing and non-negative function. Explain why your recurrences cannot be solved by the master theorem.*

One example is $\mathrm{T}(\mathrm{n}) = 4 * \mathrm{T}\left(\frac{\mathrm{n}}{2}\right) + \frac{\mathrm{n}^2}{\mathrm{logn}}$ .The Master Theorem cannot solve this recurrence because, f(n) is not a polynomial and $f(n) \notin \Theta(n^2 \log^k n)$ for some k ≥ 0. The shape follows the pattern T(n) = aT(n/b)+f(n)*,* but does not satisfy any of the conditions needed to apply the Master Theorem.

Another example is $T(n) = T\left(\frac{n}{2}\right) + n(2 - cosn)$. This example fits the pattern, and satisfies case three of the Master Theorem, but f(n) violates the required regularity condition needed to apply the Master Theorem, due to the nature of cos(n).

Problem 4 (30 points)**.** *Given the root of an existing binary search tree (BST) which is quite balanced and contains* n *distinct keys, you are asked to design a* Θ(n log n)*-time algorithm that will create a copy of the given BST. By "copy" we mean (1) the two trees have the exactly same structure; (2) for each pair of two nodes from the two trees, they contain the same key if they are at the same position.*

1. Clearly and concisely describe your algorithmic idea. Note: You do not have to write every detail of the code in a real programming language, but must be precise and clear about the main algorithmic idea, and reason why your algorithm works.

My algorithm would use recursive preorder traversal, and start by copying the data from the root node into a new Binary search tree node. It would then call itself recursively, and pass the left child node of the root, while initializing a left child node for the new tree. It would then proceed to copy data from that left child node of the root, into the left child node of the new node. It would continue in this manner until it could go left no further. It would then return until it could proceed right, again initializing a right child node in the new tree as it traversed the original. After moving right, it would copy the data of that right child into its counter part of the new tree. After this it would again move left as far as possible, before returning far enough to go right again. The algorithm would continue in this manner until it could no longer go right or left, at which point it would return all the way to the top and be finished. This should copy the original tree exactly with a time complexity of Θ(n log n)*.*

2. Give the pseudocode of your algorithm.

```
public Node copyBST(Node root){

        Node copyT = null;
        if(root != null){
                copyT = new Node();
                copyT.data = root.data
                copyT.leftChild = copyBST(root.leftChild);
                copyT.rightChild = copyBST(root.rightChild);
        }
        return copyT;
}
```

CSCD320 Homework2, Spring 2014, Eastern Washington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 20, 2014 (Sunday)

Problem 5 (20 points)**.** *We know both binary search tree (BST) and hash table can be used for data indexing for fast search. Each has pros and cons in the time and space efficiency. Do an extensive research to learn and compare these two data structures. Explain in your own language the pros and cons of each of them, and in what scenarios which works better. For each idea/opinion, provide the source of them if they are not your owns. For ex., the url of the webpages, the title and page of a book, the title/author/year of an article, etc.*

According to this article: http://brackece.wordpress.com/2012/09/18/hash-table-vs-binary-search-tree/, a drawback of hash tables is, as they grow larger the possibility of conflictions increases. If the hash table is large enough that many conflictions need to be handled, then the time it takes to retrieve data will be much slower, due to the need to search lists built out of the conflicting data. A Binary Search Tree will never have this problem, because there can be no conflictions/duplicates. Also, a hash table does not store sorted data like a Binary Search Tree does. So in the event that it is necessary or more efficient to have sorted data, a hash table will not be appropriate. Another important thing to note, is that a Binary Search Tree only takes as much space as is exactly needed, and it is easy to make it bigger. Whereas, a hash table is usually initialized to consume much more space then it strictly must, in order to leave room for new data; due to difficulties and resource cost of making already implemented hash tables bigger. However, according to: http://stackoverflow.com/questions/4128546/advantages-of-binary-search-trees-over-hash-tables, a hash table in the right circumstance can have a massive speed advantage over a Binary Search Tree. If there is no need for sorted data, and conflictions are minimal, a hash table that is set up correctly can retrieve a specified piece of data almost instantaneously with $\Theta(1)$ time. So in summation, Binary Search Trees are memory efficient, fast, and good for sorted data. Whereas, a hash table is not memory efficient, but in the right circumstance can be extremely fast.

According to http://stackoverflow.com/questions/5010854/concrete-examples-of-using-binary-search-trees, an example of a good application for a Binary Search Tree, is the management of directory listings in a program that involves file systems. An example of a good use of hash tables, is in network router programming. Hash Tables are used as an efficient lookup structure, having as key the IP address and as value the routing path for that address. This is stated at: http://programmers.stackexchange.com/questions/121621/what-are-some-common-examples-of-a-hash-table