CSCD320 Homework2

Ethan Tuning

Problem 1. *Solve the following recurrences using the Master Theorem.*

1. $T(n) = 4T(n/2) + 3n^2 - 9n$
2. $T(n) = 4T(n/2) + 2n^3 - 100n^2$
3. $T(n) = 4T(n/2) + n + 5\log n$
4. $T(n) = 8T(n/2) + 3n^2 + n\log n$
5. $T(n) = 8T(n/2) + 4n^3 - 5n^2$
6. $T(n) = 8T(n/2) + 2^-10n^4 - 6n^3$

Answer 1. *Answers are in order asked respectively. To start these problems lets define the master theorem as follows. Let $T(n)$ be the time cost of a given algorithm. This will take the form of: $T(n) = aT(n/b) + f(n)$ and $T(1) = c$, where $a \geq 1, b \geq 2, andc > 0$. If $f(n)\epsilon\Theta(n^d)$ where $d \geq 0$, then*

$$T(n) = \left\{ \begin{array}{cc} \Theta(n^d) & \textit{case 1: if } a < b^d \\ \Theta(n^d \log n) & \textit{case 2: if } a = b^d \\ \Theta(n^{\log_b a}) & \textit{case 3: if } a > b^d \end{array} \right\}$$

1. *So for this one we have $a = 4$, $b = 2$ and $d = 2$. Now using the formula from above we can see that $4 = 2^2$, so this satisfies case number 2. So the time cost will be $\Theta(n^2 \log n)$.*
2. *Now we will just continue doing the same thing for the rest of the answers as we did in the first one. So, $a = 4$, $b = 2$ and $d = 3$. Now we can see that $4 < 8$, so this satisfies case 1. Time cost will be $\Theta(n^3)$.*
3. *For this one we have $a = 4$, $b = 2$ and $d = 1$. Now we can see that $4 > 2^1$. This is an example of case number 3. So the time cost will be $\Theta(n^2)$.*
4. *Now $a = 8$, $b = 2$ and $d = 2$. We can see that $8 > 2^2$, so the time cost is $\Theta(n^3)$.*
5. *Now $a = 8$, $b = 2$ and $d = 3$. We can see that $8 = 2^3$, so the time cost is $\Theta(n^3 \log n)$.*
6. *Now $a = 8$, $b = 2$ and $d = 4$. We can see that $8 < 2^4$, so the time cost for this one will be $\Theta(n^4)$.*

Problem 2. *Propose two example recurrences that match the structure of a recurrence that can be solved by the master theorem, but ultimately cannot be solved with the master theorem.*

Answer 2. *For one we can say that $T(n) = 2T(n/2) + 2^n$ cannot be solved by the master theorem, because 2^n is not a polynomial. Now for a second one, we can say that $2T(n/2) + 2^{1/2}$ cannot work, again because $f(n)$ is not a polynomial.*

Problem 3. *Consider a binary tree, where each node has a field "value" which contains a real number. Each tree node also has "left child" and "right child" links that point to the node's left and right subtree respectively. We define the weight of a tree to be the total sum of all the numbers within the tree.*

Design a time-efficient algorithm, such that the algorithm can take the root of a binary tree as input, and return the root of the subtree who's weight is the maximum among all the subtrees of the input tree.

1. *Clearly describe your algorithmic idea.*
2. *Give pseudocode.*
3. *Give the time cost.*

Answer 3. *Answers are given in order asked respectively.*

1. *To solve this problem we need to find the weight of the current root and then traverse the entire tree. At each point in the traversal we need to get the new weight and compare it to the current one. Whichever is larger we keep.*
2.
```
int getWeight(root){
  if root != null
    return root.value + getWeight(root.leftChild) +
        getWeight(root.rightChild);
  else
    return 0;
}

node findMaxWeight(root){
  int maxWeight = -1;

  if root{
    if maxWeight < getWeight(root)
      maxWeight = getWeight(root);
    findMaxWeight(root.leftChild);
    findMaxWeight(root.rightChild);
  }

  return maxWeight;
}
```
3. *The time cost will in fact be $O(n \log n)$. We can say this because the helper will be linear time, and then to total method has 2 recursive calls, add the helpers time and there you go.*

Problem 4. *We know that both BST's and hash tables can be used for data indexing for fast search. Each has pros and cons in the time and space efficiency. Do 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 own.*

Answer 4. *Both these data structures are very great, but depending on what you are doing you will want to se one over the other. For one, hash tables use lots of memory. If you are limited to a certain amount of memory go with the BST for sure. Your only ever going to store the exact amount of data you need with a BST. Not so much with a hash table. Now with regards to sorting you would want to use a BST. This operation is super trivial with a BST adn can be accomplished very elegantly. It is not as natural with a hash table. Now if you are inserting, deleting or searching go with a hash table. These operations can be done in constant time, this is really what a hash table is meant for. Now as far as ease, BST is very easy to implement. If you are just needing to do implementing a data structure quickly and easily use a BST over a hash table. Hash tables have a lot more elements that make it up. All-in-all they are both good, but it just depends on what you are doing.*