CSCI-GA.1170-001/002 Fundamental Algorithms September 14, 2014

Solutions to Problem 1 of Homework 2 (10 points)

Name: Keeyon Ebrahimi *Due: Wednesday, September 17*

(a) (6 points) Suppose you have some procedure FASTMERGE that given two sorted lists of length m each, merges them into one sorted list using m^c steps for some constant $c > 0$. Write a recursive algorithm using FASTMERGE to sort a list of length n and also calculate the run-time of this algorithm as a function of c. For what values of c does the algorithm perform better than $O(n \log n)$.

Solution:
A is the list to be merged, this is the recursive algorithm featuring FASTMERGE to create the sorted list.

```
FASTMERGESORT($A$)
    **If** $A.length < 2$
        **Return** $A$

    $List1 =$ FASTMERGESORT($A[0 : \frac{n}{2}]$)
    $List2 =$ FASTMERGESORT($A[\frac{n}{2} + 1 : n]$)
    **Return** FASTMERGE($List1, List2$)
```

Calculate the run-time of this algorithm as a function of c:

For this, we have a subset size of n/2 and we divide it into 2 subproblems. So we can step through the problem like this.

1. $T(n) = 2T(\frac{n}{2}) + DivideTime + CombineTime$

    – The Divide Time is constant, for we just divide n by 2, so we know.

2. $T(n) = 2T(\frac{n}{2}) + \Theta(1) + CombineTime$

    – The Combine Time is m^c, *with* $c > 0$ and $m = \frac{n}{2}$
    – Combine Time then becomes $(\frac{n}{2})^c$ which reduces to $O(n^c)$

3. $T(n) = 2T(\frac{n}{2}) + \Theta(1) + O(n^c)$

   - Using the master theorem, we can find the running time
   - $n^{\log_b a} = n$, so comparing n with n^c gives us

$$O(n) = \begin{cases} n\ log(n) & \text{if } c = 1 \\ n^c & \text{otherwise} \end{cases}$$

For what values of c does the algorithm perform better than $O(n\ log\ n)$

There are no values of c where this algorithm performs better than $O(n\ log\ n)$ steps.

If $c = 0$, then the complete running time would be $O(n)$, but $c > 0$, meaning that **there are no valid values of c where this algorithm performs better that $O(n\ log\ n)$**

□

(b) (4 points) Let $A[1 \dots n]$ be an array such that the first $n - \sqrt{n}$ elements are already in sorted order. Write an algorithm that will sort A in substantially better than $O(n \log n)$ steps.

Solution: An algorithm that will run substantially better in this situation is running a Merge Sort on the Array A from $n - \sqrt{n} + 1$ to n and then combine the first $n - \sqrt{n}$ with the newly merged items

```
SqrtMerge($A$)
    $listA = MergeSort(A[n - \sqrt{n} + 1 : n])$
    For $i$ in $listA$
        $A[n - \sqrt{n} + 1 + i] = listA[i]$
    Return $A$
```

Running time = $O(log(n))$

We are now running the Merge Sort on a size of $\sqrt{n}$.

On a size of n, Merge Sort gives us the running time of $O(n\ log(n))$. This is because the splitting gives $O(log(n))$ steps, and combining in those steps takes $O(n)$ time, giving $O(n\ log(n))$ running time

On a size of $\sqrt{n}$, like in this problem, the running time is $log(n) * log_4(n)$, which reduces down to $O(log(n))$.

The running time is $log(n) * log_4(n)$ because the splitting of the array creates $log_4(n)$ steps , the combining takes $log(n)$ time in those steps.

$O(log(n) * log_4(n))$ reduces to $O(log(n))$, making this running time substantially shorter

$\square$

CSCI-GA.1170-001/002 Fundamental Algorithms September 14, 2014

Solutions to Problem 2 of Homework 2 (10 points)

Name: Keeyon Ebrahimi *Due: Wednesday, September 17*

Consider the following recursive procedure.

```
BLA(n):
    If $n = 1$ Then Return 1
    Else Return BLA($n/2$) + BLA($n/2$) + BLA($n/2$)
```

(a) (3 points) What function of n does BLA compute (assume it is always called on n which is a power of 2)?

Solution: $\Theta(1)$

To find the function of n, we add $O(divideTime) + O(combineTime)$
In this example, the divide time is constant because we just divide n by 2
The combine time is also constant because we are doing 2 additions, which also takes constant time
$O(1) + O(1) = \Theta(1)$

□

(b) (3 points) What is the running time $T(n)$ of BLA?

Solution: $O(n^{\log_2(3)})$

With each step of recursion, we divide the problem into 3 sub problems of size $\frac{n}{2}$

1. In $T(n)$ notation, we know the running time is $T(n) = aT(\frac{n}{b}) + f(n)$
2. This means that for our example, $T(n) = 3T(\frac{n}{2}) + \Theta(1)$
3. Using the master theorem, we know that the running time is $n^{\log_2(3)}$

□

(c) (4 points) How do the answers to (a) and (b) change if we replace the last line by "**Else Return** $3 \cdot$ BLA($n/2$)"?

Solution: The answer of (a) does not change, but (b) does. Now both (a) and (b) = $O(1)$

Part A $= \Theta(1)$

When we change the last line to the $3 \cdot \textsc{Bla}(n/2)$, the change in regards of $f(n)$, we are only changing 2 addition operations into a multiplication by 3, which internally is the exact process, so for part a

$f(n) = \Theta(1)$

Part B $= \Theta(1)$

Now, instead of dividing the problem into 3 sub problem of size $\frac{n}{2}$, we are now just making this divide into 1 sub problem of $\frac{n}{2}$

This means that in our $T(n) = aT(\frac{n}{b}) + f(n)$ equation, with $a = 1$ and $b = 2$, we get $T(n) = 1T(\frac{n}{2}) + f(n)$. We then solv

□

CSCI-GA.1170-001/002 Fundamental Algorithms September 14, 2014

Solutions to Problem 3 of Homework 2 (14 points)

Name: Keeyon Ebrahimi *Due: Wednesday, September 17*

Consider the recurrence $T(n) = 8T(n/4) + n$ with initial condition $T(1) = 1$.

(a) (2 points) Solve it asymptotically using the "master theorem".

Solution: $O(n\ log(n))$

In this recurrence, we have 8 subdivisions of size $\frac{n}{4}$, this means that we have to compare $n^{log_4(8)}$ with $f(n)$, which is n. The power of $n^{log_4(8)}$ is greater than the power of n, and using the master theorem, we know that our running time is $n^{log_4(8)}$, which reduces down to $n^{1.5}$ which equals $n\ log(n)$

□

(b) (4 points) Solve it by the "guess-then-verify method". Namely, guess a function $g(n)$ — presumably solving part (a) will give you a good guess — and argue by induction that for all values of n we have $T(n) \leq g(n)$. What is the "smallest" $g(n)$ for which your inductive proof works?

Solution:

When we use part A as our guess, we see that $T(1) = 1\ log(1) = 0$, meaning our Part A guess doesn't pass the base case.
That being the case, we change our guess to $O(n\ log(n) + 1)$, which does pass the base case test.

- $T(n) = 8T(n/4) + n$
- $g(n) = cn\ log(n) + 1$ where $c > 0$

We want to prove that $T(n) \leq g(n)$

To solve, we need to substitute $g(n)$ into $T(n)$ and solve in order to inductively proof.

1. $T(n) = 8(c * \frac{n}{4} *\ log(\frac{n}{4}) + 1) + n$
2. $T(n) = 2nc *\ log(\frac{n}{4}) + 8 + n$
3. $T(n) = (2nc *\ log(n)) - (2nc *\ log(4)) + 8 + n$
4. $T(n) = cn\ log(n) - cn + n$
5. $T(n) \leq cn\ log(n)$
6. $T(n) \leq g(n)$

What is the "smallest" $g(n)$ for which your inductive proof works?

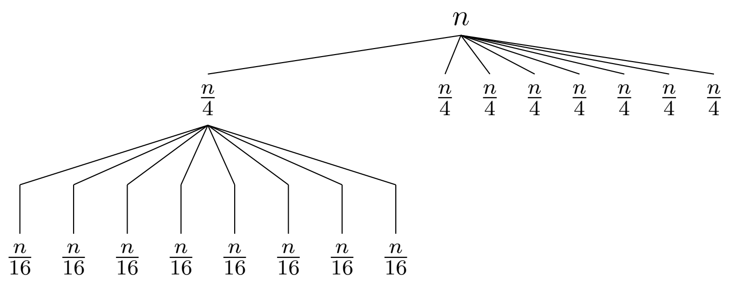
The smallest $g(n)$ would be when $T(n) = g(n)$, which only happens when $c = 1$ and $n = 1$

$\square$

(c) (4 points) Solve it by the "recursive tree method". Namely, draw the full recursive tree for this recurrence, and sum up all the value to get the final time estimate. Again, try to be as precise as you can (i.e., asymptotic answer is OK, but would be nice if you preserve a "leading constant" as well).

Solution:

This is what the tree will look like. Each of the $\frac{n}{4}$ subtrees should have the tree of $\frac{n}{16}$ under it, but due to space, I had to draw it this way.



n
$\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$
$\frac{n}{16}$ $\frac{n}{16}$ $\frac{n}{16}$ $\frac{n}{16}$ $\frac{n}{16}$ $\frac{n}{16}$ $\frac{n}{16}$ $\frac{n}{16}$

The first layer sums to n, the remaining layers excluding the last layer each sum up to $2n$, and the last layer sums up to $n^{log_4(8)}$ which is $n\ log(n)$

We also know that we have a total of $log_b(n)$ levels as well.

When we sum those all together, we get $2n\ log(n) + n\ log(n)$ which equals $3n\ log(n)$

$\square$

(d) (4 points) Solve it *precisely* using the "domain-range substitution" technique. Namely, make several changes of variables until you get a basic recurrence of the form $S(k) = S(k-1) + f(k)$ for some f, and then compute the answer from there. Make sure you carefully maintain the correct initial condition.

Solution: ***************** INSERT PROBLEM 3d SOLUTION HERE ***************

$\square$

(e) This part will not be graded. However, briefly describe your personal comparison of the above 4 methods. Which one was the fastest? The easiest? The most precise?

Solution: For me, the fastest and easiest method was the Master Theorem. Although you do not get the same precision with the Master Theorem as you would with the Recursive Tree Method. I enjoy the all $\square$

CSCI-GA.1170-001/002 Fundamental Algorithms September 14, 2014

Solutions to Problem 4 of Homework 2 (12 points)

Name: Keeyon Ebrahimi *Due: Wednesday, September 17*

Solve the following recurrences using any method you like. If you use "master theorem", use the version from the book and justify why it applies. Assume $T(1) = 2$, and be sure you explain every important step.

(a) (3 points) $T(n) = T(2n/3) + \sqrt{n}$.

Solution: $O(sqrt(n))$

Using the master theorem, we compare the $n^{log_b(a)}$ with $f(n)$. $f(n) = \sqrt{n}$
For $n^{log_b(a)}$, we can see that $a = 1$ and $b = 3/2$. This means $n^{log_b(a)} = n^{log_{\frac{3}{2}}(1)} = n^0 = \Theta(1)$

When comparing $\Theta(1)$ with $\sqrt{n}$, we see that the $f(n)$ power is greater than $n^{log_b(a)}$, meaning that we fall into Case 3 from the book, giving us a total running time of $O(\sqrt{n})$ $\square$

(b) (4 points) $T(n) = T(n/2) + \log n$.

Solution: $O(log(n))$

Using the master theorem, we compare the $n^{log_b(a)}$ with $f(n)$. $f(n) = log(n)$.
For $n^{log_b(a)}$, we can see that $a = 1$ and $b = 2$. This means $n^{log_b(a)} = n^{log_2(1)} = n^0 = \Theta(1)$

When comparing $\Theta(1)$ with $log(n)$, we see that the $f(n)$ power is greater than $n^{log_b(a)}$, meaning that we fall into Case 3 from the book, giving us a total running time of $O(log(n))$ $\square$

(c) (5 points) $T(n) = T(\sqrt{n}) + 1$. (**Hint**: Substitute ... until you are done!)

Solution: $O(log(log(n)))$
We look at our base case to find what our ending value is after all the recursion, and it is 2. With this , we know that our total steps of recursions are 2^{2^r} where r is the amount of steps.

We know that $2^{2^r} = n$. So

1. $2^{2^r} = n$
2. $2^r = log_2(n)$
3. $r = log_2(log_2(n))$

This shows us that the amount of layers in our recursion tree is $log_2(log_2(n))$. Our $f(n) = \Theta(1)$, so when we multiple $f(n)$ by the amount of recursion steps we take, we get the total running time of $O(log(log(n)))$ $\square$