CSCI-GA.1170-001/002 Fundamental Algorithms September 16, 2014

Solutions to Problem 1 of Homework 2 (10 points)

Name: Sahil Goel *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:

MergeSort(A,low,high) {
if($low >= high$)
return;
$pivot = \frac{low+high}{2}$;
MergeSort(A,low,pivot);
MergeSort(A,pivot+1,high);
FastMerge(A,low,pivot+1,high);
}

Now calculating the complexity of solution

$$T(n) = 2 * T(n/2) + (n/2)^c;$$

Using master's theorem, $f(n) = n^{\log_b a}$
Therefore $f(n) = n^{\log_2(2)}$
f(n)=n
g(n)=$(n/2)^c$; Now for the runtime algorithm to be perform better than $O(n \log n)$, $f(n) > g(n)$
Therefore $n > (n/2)^c$
Taking log on both sides

$$\log n > c \log(n/2)$$

$$c < \frac{\log n}{\log n/2}$$

$$c < \frac{\log n}{\log n - \log 2}$$

For very large n

$$c < 1$$

if $c < 1$, then complexity of function will be O(n) □

(b) (4 points) Let $A[1 \ldots n]$ 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: Using the MergeSort from last question and Merge function to merge 2 sorted array in one complete sorted array
Step 1: Call MergeSort for last $\sqrt{n}$ elements
Step 2: Call Merge for 2 sorted arrays A1[1:n-$\sqrt{n}$] and A2[n-$\sqrt{n}$+1:n]

```
SortLast(A,n)
{
MergeSort(A,n − √n + 1,n); //Calling Merge sort only for last √n elements
Merge(A,0,n − √n + 1,n); //Merging 2 sorted arrays
}

MergeSort(A,low,high) {
if(low >= high)
return;
pivot = (low+high)/2;
MergeSort(A,low,pivot);
MergeSort(A,pivot+1,high);
FastMerge(A,low,pivot+1,high);
}
```

In merge function, l is the first index for first sorted array and m is the first index for second sorted array and h is the last element in second array

```
Merge(A,l,m,h)
{
int b1[m-l],b2[h-m+1],i=0,j=0,lower=l;
//storing two sorted arrays in temporary arrays b1 and b2
while(i < m − l)
{
b1[i]=a[l+i];
i++;
}
while(j < h − m + 1)
{
b2[j]=a[m+j];
j++;
}
i=0;
j=0;
//Now merging two arrays and comparing only b1[i] and b2[j]
while(i < m − lower&&j < h − m + 1)
{
if(b1[i] < b2[j])
```

a[l++]=b1[i++];
else
a[l++]=b2[j++];
}
// b2 has reached end therefore copy all the elements from b1
while($i < m - lower$)
{
a[l++]=b1[i++];
}
// b1 has reached end therefore copy all the elements from b2
while($j < h - m + 1$)
{
a[l++]=b2[j++];
}
}

Total time complexity of code will be T(n), where

$$T(n) = T(MergeSort) + T(Merge)$$

To MergeSort last $\sqrt{n}$ elements, time complexity is O($\sqrt{n} * \log \sqrt{n}$) And to Merge n elements, time complexity is O(n)
Therefor total complexity is O($\sqrt{n} * \log \sqrt{n}$) + O(n)
Since $\log \sqrt{n} < \sqrt{n}$
Multiplying with $\sqrt{n}$ on both sides
$\sqrt{n} * \log \sqrt{n} < \sqrt{n} * \sqrt{n}$
Therefore for very large n, O(n) will dominate and time complexity will be O(n) □

CSCI-GA.1170-001/002 Fundamental Algorithms September 16, 2014

Solutions to Problem 2 of Homework 2 (10 points)

Name: Sahil Goel *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: At every step, the result is multiplied by 3 and it is recursively called $\log_2 n$ times Therefore the function computes $3^{\log_2 n}$ □

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

Solution:

$$T(n) = T(n/2) + T(n/2) + T(n/2)$$

$$T(n) = 3 * T(n/2)$$

Replacing n with 2^k

$$T(2^k) = 3 * T(2^{k-1})$$

Replacing $T(2^k) with S(k)$

$$S(k) = 3 * S(k-1)$$

Dividing by 3^k

$$S(k)/3^k = S(k-1)/3^{k-1}$$

Assume $S(k)/3^k = P(k)$

$$P(k) = P(k-1)$$

Therefore

$$P(k) = P(0) = 1$$

$$P(k) = 1$$

$$S(k) = 3^k * 1$$

$$T(2^k) = 3^k$$

Since $k = \log_2 n$

$$T(n) = 3^{\log_2 n}$$

$$T(n) = n^{\log_2 3}$$
$$T(n) = n^{1.58}$$

□

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

Solution: (a) part remains same but answer to (b) part changes
Now BLA(n/2) is only called one time as compared to three times in previous step.
Assuming multiplication with 3 takes c time

$$T(n) = T(n/2) + c$$

Substituting $n = 2^k$

$$T(2^k) = T(2^{k-1}) + c$$
$$S(k) = S(k-1) + c$$
$$S(k) = S(k-2) + c + c$$

Similarly

$$S(k) = c + c + c + \ldots + c(ktimes)$$
$$S(k) = kc$$
$$T(2^k) = kc$$

Substituting $k = \log_2 n$

$$T(n) = \log_2 n * c$$

Therefore total complexity is $O(\log_2 n)$

□

CSCI-GA.1170-001/002 Fundamental Algorithms September 16, 2014

Solutions to Problem 3 of Homework 2 (14 points)

Name: Sahil Goel *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:

$$f(n) = n^{\log_b a}$$
$$f(n) = n^{\log_4 8}$$
$$f(n) = n^{3/2\log_2 2}$$
$$f(n) = n^{\frac{3}{2}}$$
$$g(n) = n^{\log_b a - \epsilon}$$
$$g(n) = n^{3/2-\epsilon}$$

Since g(n)=n and $\epsilon = 1/2$

$$T(n) = O(f(n))$$
$$T(n) = O(n^{\frac{3}{2}})$$

□

(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: This solution has 2 parts, one will be wrong guess because of loose bounds and other will be right guess because of tight bounds.
Case 1:
Let us assume T(n)=O(g(n))
and We guess $g(n) = n^{3/2}$
Therefore the condition that should hold is
There $\exists c > 0$ s.t. $T(n) \leq c * (n^{3/2})$
For n=1

$$1 = T(1) \leq c * (1)^{3/2}$$

Therefore $c \geq 1$
Assuming the solution is true for 1,2,...,n-1 , we have to prove it is true for n where $n > 1$

$$T(n) = 8 * T(n/4) + n$$

Since $T(n) \leq c * (n^{3/2})$

$$8 * (c * (n/4)^{3/2}) + n \leq c * (n^{3/2})$$

$$\frac{8 * c * n^{3/2}}{4^{3/2}} + n \leq c * n^{3/2}$$

$$c * n^{3/2} + n \leq c * n^{3/2}$$

Therefore c term gets cancelled out, so the proof goes wrong

Case II
Let us guess $g(n) = n^{3/2} - n$ and T(n)=O(g(n)
Therefore the condition that should hold is
There $\exists c > 0$ s.t. $T(n) \leq c1 * (n^{3/2}) - c2 * n$
For n=1

$$1 = T(1) \leq c1 * (1)^{3/2} - c2$$

$$c1 \geq 1 + c2$$

Assuming the solution is true for 1,2,. . . ,n-1 , we have to prove it is true for n where $n > 1$

$$T(n) = 8 * T(n/4) + n$$

Since $T(n) \leq c1 * (n^{3/2}) - c2 * (n)$

$$8 * [c1 * (n/4)^{3/2} - c2 * (n/4)] + n \leq c1 * n^{3/2} - c2 * n$$

$$c1 * n^{3/2} - 2 * c2 * n + n \leq c1 * n^{3/2} - c2 * n$$

$$-2 * c2 + 1 \leq -c2$$

$$1 \leq c2$$

and Since $c1 \geq 1 + c2$

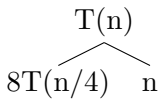
$$c1 \geq 2$$

$$c2 \geq 1$$

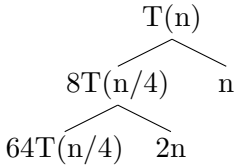
The minimum value of g(n) will be $2 * n^{3/2} - n$ □

(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).

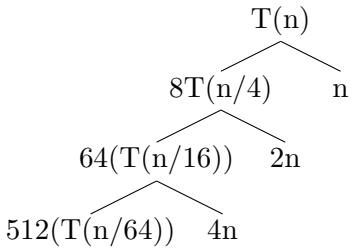
Solution: [.NP **b** c]



Expanding tree to second step



Expanding more



Similarly expanding and summing all the value

$$\begin{aligned} & n + 2n + 4n + \ldots + 2^{log_4 n} n \\ \implies & n + 2n + 4n + \ldots + \sqrt{n} * n \\ \implies & n(1 + 2 + 4 + 8 + \ldots + \sqrt{n} * n) \\ \implies & n(2 * \sqrt{n} - 1) \\ \implies & 2 * n^{3/2} - n \end{aligned}$$

Hence $T(n) = 2 * n^{3/2} - n = O(n^{3/2})$

□

(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:

$$T(n) = 8 * T(n/4) + n$$

Substituting $n = 4^k$

$$T(4^k) = 8 * T(4^{k-1}) + 4^k$$

Lets $T(4^k) = S(k)$

$$S(k) = 8 * S(k-1) + 4^k$$

Dividing whole equation by 8^k

$$\frac{S(k)}{8^k} = \frac{S(k-1)}{8^{k-1}} + \frac{1}{2^k}$$

Now lets $P(k) = S(k)/8^k$

$$P(k) = P(k-1) + \frac{1}{2^k}$$

$$P(k) = P(k-2) + \frac{1}{2^{k-1}} + \frac{1}{2^k}$$

$$P(k) = \frac{1}{2^k} + \frac{1}{2^{k-1}} + \ldots + \frac{1}{2} + \frac{1}{2^0}$$

$$P(k) = 2 * [1 - \frac{1}{2^{k+1}}]$$

$$P(k) = 2 - \frac{1}{2^k}$$

Now $S(k) = P(k) * 8^k$

$$S(k) = 2 * 8^k - \frac{8^k}{2^k}$$

$$T(4^k) = 2 * 8^k - \frac{8^k}{2^k}$$

Now $k = \log_4 n$

$$T(n) = 2 * n^{3/2} - n$$

□

(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: The fastest one is master's theorem because we just have to check the values of function f(n) and g(n) which is very easy to calculate and just compare the 2 functions
The easiest is also I think master's theorem if we don't consider the exact preciseness
The Most precise method is substitution method because we are able to find even the constant factors
The order according to me
For easiness, Most easiest one first
$Master'sTheorem > SubstitutionMethod > RecursiveTee > Guessandverify$
For fastness, fastest one first
$Master'sTheorem > RecursiveTree > SubstitutionMethod > Guessandverify$
For Preciseness, most precise one first
$Substitutionmethod > RecursiveTree > Master'sTheorem > GuessandVerify$

□

CSCI-GA.1170-001/002 Fundamental Algorithms September 16, 2014

Solutions to Problem 4 of Homework 2 (12 points)

Name: Sahil Goel *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:

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

Substituting $n = (3/2)^k$

$$T((3/2)^k) = T((3/2)^{k-1}) + (3/2)^{k/2}$$

Substituting $T((3/2)^k) = S(k)$

$$S(k) = S(k-1) + (3/2)^{k/2}$$

$$S(k) = S(k-2) + (3/2)^{(k-1)/2} + (3/2)^{k/2}$$

$$S(k) = 2 + (3/2)^{1/2} + \ldots + (3/2)^{(k-1)/2} + (3/2)^{k/2}$$

$$S(k) = 1 + 1 + (3/2)^{1/2} + \ldots + (3/2)^{(k-1)/2} + (3/2)^{k/2}$$

$$S(k) = 1 + \frac{[(3/2)^{(k+1)/2}] - 1}{(3/2)^{1/2} - 1}$$

$$S(k) = 1 + \frac{[(3/2)^{1/2}(3/2)^{k/2}] - 1}{(3/2)^{1/2} - 1}$$

Substituting $k = \log_{3/2} n$

$$T(n) = 1 + \frac{[(3/2)^{1/2}(3/2)^{(\log_{3/2} n)/2}] - 1}{(3/2)^{1/2} - 1}$$

$$T(n) = 1 + \frac{(3/2)^{1/2} * (n)^{(1/2)} - 1}{(3/2)^{1/2} - 1}$$

$$T(n) = 9.169\sqrt{n} - 7.169$$

Therefore
$T(n) = \Theta(\sqrt{n})$ □

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

Solution:

$$T(n) = T(n/2) + \log n$$

Substituting $n = 2^k$

$$T(2^k) = T(2^{k-1}) + \log 2^k$$
$$T(2^k) = T(2^{k-1}) + k \log 2$$
$$T(2^k) = T(2^{k-1}) + k$$

lets $S(k) = T(2^k)$

$$S(k) = S(k-1) + k$$
$$S(k) = S(k-2) + k - 1 + k$$
$$S(k) = S(0) + 1 + 2 + \ldots + k - 1 + k$$

Since S(0)=T(1)

$$S(k) = 2 + \frac{k * (k+1)}{2}$$
$$S(k) = \frac{4 + k^2 + k}{2}$$

Substituting k=log n

$$T(n) = \frac{4 + (\log n)^2 + (\log n)}{2}$$

Therefore $T(n) = \Theta((\log n)^2)$

□

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

Solution:

$$T(n) = T(\sqrt{n}) + 1$$

Substituting $n = 2^k$

$$T(2^k) = T(2^{k/2}) + 1$$

Substituting $S(k) = T(2^k)$

$$S(k) = S(k/2) + 1$$

Substituting $m = 2^k$

$$S(2^m) = S(2^{m-1}) + 1$$

Lets $P(m) = S(2^m)$

$$P(m) = P(m-1) + 1$$
$$P(m) = P(m-2) + 1 + 1$$

$$P(m) = 1 + 1 + \ldots + 1 + P(1)$$

$$P(m) = m - 1 + P(1)$$

Since $T(2) = T(1.41) + 1 \Longrightarrow T(2) = T(1) + 1 \Longrightarrow T(2) = 3$

$$P(m) = 1 + 1 + \ldots + 1 + 3$$

$$P(m) = m + 2$$

Since $m = logk$

$$S(k) = \log k + 2$$

Since $k = \log n$

$$T(n) = \log(\log n) + 2$$

Therefore $T(n) = \Theta(\log \log n)$

□