Justin Baraboo
Design and Analysis of Algorithms
Assignment 1

February 10, 2015

1 Problem 1

Use the summation method to solve the following recurrence relation:
$$T(n) = 4T(\tfrac{n}{2}) + n^2$$

Solution

$$\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n^2 \\
&= n^2 + 4(\frac{n}{2})^2 + 16T(\frac{n}{4}) \\
&= n^2 + 4(\frac{n}{2})^2 + 16(\frac{n}{4})^2 + 64T(\frac{n}{8}) \\
&= \sum_{i=1}^{\lg(n)} 4^i(\frac{n}{2^i})^2 + T(1) \\
&= \sum_{i=1}^{\lg(n)} n^2 + T(1) \\
&= n^2 \lg(n) + T(1) \\
&\in O(n^2 \lg(n))
\end{aligned}$$

$$T(n) = T(\tfrac{n}{4}) + \lg(n).$$

Assume that $n = 4^k$, then $T(4^k) = T(4^{k-1}) + \frac{1}{2\log_4(4^k)}$
Or that $W(k) = W(k-1) + \frac{1}{2k}$.

$$\begin{aligned}
W(k) &= W(k-1) + \frac{1}{2k} \\
&= W(k-2) + \frac{1}{2k} + \frac{1}{2(k-1)} \\
&= \frac{1}{2(k)} + \frac{1}{2(k-1)} + \frac{1}{2(k-2)} + W(k-3) \\
&= \sum_{i=2}^{k} \frac{1}{2i} + W(1) \\
&= \frac{H_k}{2} + W(1) - 1 \text{ ,where} H_{k-1} \text{ is the harmonic series sum} \\
&= \frac{1}{2}\lg(k) + W(1) + \text{constant} \\
&\in O(\lg(k))
\end{aligned}$$

Going back we have: $\lg(k) = \lg(log_4(n) \in O(\lg(\lg(n)))$.
So, $T(n) \in O(\lg(\lg(n)))$.

2 Problem 2

Use the master method to determine and the theta notation to represent the asymptotic growth rate of each of the following recurrence relations. Assume $T(n)$ is constant for $n \leq 4$. You need to show clear steps to justify your answers.

(a) $T(n) = 16T(\frac{n}{3}) + n^2\lg(n)$

(b) $T(n) = 8T(\frac{n}{4}) + n\sqrt{(n)}$

(c) $T(n) = 7T(\frac{n}{2}) + n^3$

Solution (a) $T(n) = 16T(\frac{n}{3}) + n^2 \lg(n)$
We will use the first rule of the Master's Theorem. In that:

$$a = 16, b = 3$$
$$\log_3(16) \approx 2.53$$
$$f(n) = n^2 \lg(n)$$

Let $\epsilon = .03$ and we will show that $f(n) \in O(n^{\frac{3}{2}})$.
From the definition of $O(g(n))$, we must find a constant $c > 0$ and a number $n_o > 0$ such that:

$$f(n) \leq cg(n) \ \forall n \geq n_o$$

that is:

$$n^2 \lg(n) \leq cn^2 n^{\frac{1}{2}}$$

Let $c = 2$ and $n_o = 20$. And the equality holds for all $n \geq n_o$.
To illustrate this, we really need to only show that $\lg(n) \leq n^{\frac{1}{2}}$. Using L'Hopital's rule we can show that

$$\lim_{n\to\infty} \lg(n)/n^{\frac{1}{2}} = \frac{\infty}{\infty}$$

So apply the rule:

$$\lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{(n)}}}$$
$$\lim_{n\to\infty} \frac{2}{\sqrt{(n)}} = 0$$

As $\lg(n)$ and $\sqrt{(n)}$ are nondecreasing functions with the above result, if we find a place where $\sqrt{(n)} \geq \lg(n)$ it will always be greater. For the values listed above it does.

Thus by the first rule of the Master's Theorem, $T(n) \in \Theta(n^{\log_3(16)})$.

(b) $T(n) = 8T(\frac{n}{4}) + n\sqrt{(n)}$

We will use the second rule of the Master's Theorem. In that:

$$a = 8, b = 4$$
$$\log_4(8) = \frac{3}{2}$$
$$f(n) = n\sqrt{(}n)$$

We will show that $f(n) \in O(n^{\frac{3}{2}})$. From the definition of $O(g(n))$, we must find a constant $c > 0$ and a number $n_o > 0$ such that:

$$f(n) \leq cg(n)\ \forall n \geq n_o$$

that is:

$$n\sqrt{(}n) \leq cn^{\frac{3}{2}}$$

Let $c = 1$ and $n_o = 5$. And the equality holds for all $n \geq n_o$.

We will show that $f(n) \in \Omega(n^{\frac{3}{2}})$. From the definition of $\Omega(g(n))$, we must find a constant $c > 0$ and a number $n_o > 0$ such that:

$$f(n) \geq cg(n)\ \forall n \geq n_o$$

that is:

$$n\sqrt{(}n) \geq cn^{\frac{3}{2}}$$

Let $c = 1$ and $n_o = 5$. And the equality holds for all $n \geq n_o$.
This is justified as they are exactly the same function.
So $f(n) \in \Theta(n^{\frac{3}{2}})$.

Thus by the second rule in the Master's Theorem, $T(n) \in \Theta(n^{\frac{3}{2}} \lg(n))$.

(c) $T(n) = 7T(\frac{n}{2}) + n^3$

We will use the third rule of the Master's Theorem. In that:

$$a = 7, b = 2$$
$$\log_4(8) \approx 2.81$$
$$f(n) = n^3$$

Let $\epsilon = .10$, and we will show that $f(n) \in \Omega(n^{2.91})$. From the definition of $\Omega(g(n))$, we must find a constant $c > 0$ and a number $n_o > 0$ such that:

$$f(n) \geq cg(n)\ \forall n \geq n_o$$

that is:

$$n^3 \geq cn^{2.91}$$

Let $c = 1$ and $n_o = 5$ and the equality holds for all $n \geq n_o$.
This is justified that $n^3 = n^{2.91} \times n^{.09} \geq n^{2.91}$ for the values above and $n \geq n_o$.

Next we must show that:

$$af(\tfrac{n}{b}) \leq cf(n)$$

or that:

$$7(\tfrac{n}{2})^3 \leq cn^3$$

Let $c = 7/8$ and $n_o = 5$ and the equality holds for all $n \geq n_o$. This is justified as they are then the same function.

Thus by the third rule of the Master's Theorem, $T(n) \in \Theta(n^3)$.

3 Problem 3

Prove that any full binary tree with n leaves has height $\lceil \lg(n) \rceil$ if all the leaves are located in the bottom two levels.

Solution We will use the following definitions and aspects of binary trees in our proof:

A binary tree is full if every non leaf node has exactly two children.
A complete binary tree is a tree in which every tree level is completely filled.
The height of a binary tree is the maximum number of edges from the root to any leaf.

First, we will show that a complete binary tree with m leaves has height $\lg(m)$ through induction.
Base case: 1 leaves. You have just the root,which is also the leaf, no edges, and thus Height $= 0$ and $\lg(1) = 0$.
Induction : Assume this relation holds for $k < m$ and show for m, where $m > 1$: Let k be the number of nodes in the (last -1) row, which exists as $m > 1$. $m = 2k$ and more importantly $k < m$. So the Height of the nested complete binary tree from the root to the (last -1) row is $\lg(k)$. The number of edges from this row to the last is 1 for any leaf. Thus the Height of the entire tree is:
$\lg(k) + 1 = \lg(2k) = \lg(m)$.

Now that we have shown this for a complete tree, we will solve the problem. If our binary tree is complete, we are done. So assume our binary tree is not complete. Since our binary tree is full and not complete it must be of at least height 2, so a (last -1) row will always exist. Let m be the number of nodes in the (last -1) row and r be the number of nodes in the last row. We have that $m = n - \frac{r}{2}$. Since all the leaves are in the last two rows, the nested binary tree from root to (last -1) row represents a complete binary tree. Thus, it has height: $\lg(m)$
And each leaf in the last row is only one edge away from the (last -1) row, and thus each leaf in the last row has a maximum number of edges from the root. Thus the entire tree has height:

$$\lg(m) + 1 = \lg(2m).$$

We must now show that $\lceil \lg(n) \rceil = \lg(m) + 1$.

$$\begin{aligned} r &> 2 \text{ as the tree is not complete and full} \\ m &< n \text{ as r is nonzero} \\ n &< 2m \text{ as the tree is not complete and full} \end{aligned}$$

So:

$$\lg(m) < \lg(n) < \lg(2m)$$
$$\lg(m) < \lg(n) < \lg(m) + 1$$

Note that the $\lg(x)$ function is a nondecreasing function. So $\lg(n)$ takes a number strictly between two consecutive integers (as $\lg(m)$ is an integer value). So $\lceil \lg(n) \rceil = \lg(m) + 1$, so the height is $\lceil \lg(n) \rceil$.

4 Problem 4

Solution For this problem we will use the following:

For a given $n \leq j < m \leq k$:
where $B[n] < B[m]$ and $S[j] > S[k]$, $B[n], S[j]$ are optimal to buy and sell.
where $B[n] > B[m]$ and $S[j] < S[k]$, $B[m], S[k]$ are optimal to buy and sell.
where $B[n] < B[m]$ and $S[j] < S[k]$, $B[n], S[k]$ are optimal to buy and sell.

where $B[n] > B[m]$ and $S[j] > S[k]$, then the pair $B[n], S[j]$ and $B[m], S[k]$ which has the largest difference between them is optimal.

We will divide the arrays until we only have one element in each, which represents the optimal place to buy. Then we will combine them using the above heuristics. Since the lower calls will always have lower indexes than the upper calls we can generalize it this way. This algorithm assumes you can't buy apples for a negative value. The psuedo code follows:

```
FindMaxProfit( B[u...r], S[u...r])
{
   \\basecase
   if( u == r) return u,u

   \\division
   mid = floor(( u+r) /2 )
   b1,s1 = FindMaxProfit( B[u... mid], S[u...mid] )
   b2,s2 = FindMaxProfit( B[mid+1...r],S[mid+1...r])

   if( B[b1] <= B[b2] & S[s1] >= S[s2] )
      return b1,s1
   if( B[b1] > B[b2] & S[s1] <= S[s2] )
      return b2,s2
   if( B[b1] < B[b2] & S[s1] < S[s2] )
      return b1,s2
   else \\ B[b1] > B[b2] & S[s1] > S[s2]
      if( (S[s1] - B[b1] ) >= (S[s2] - B[b2] ) )
         return b1,s1
      else \\ (S[s1] - B[b1] ) < (S[s2] - B[b2] )
         return b2,s2
}
```

Call FindMaxProfit(B[0...n], S[0..n]). It returns the the index of the best buy and sell place.
Our basic operation is the comparisons operator and our input size is the number of places to buy/sell. This algorithm is given by the following recurrence relation:

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + 7$$
$$T(n) = 2T(\tfrac{n}{2}) + 7$$

We will use the Masters theorem where:

$$\begin{aligned} a = 2, b = 2 \\ \log_2(2) = 1 \\ f(n) = 7 \end{aligned}$$

It is trivial that $7 \in O(n)$ as you let $c = 7$ and it holds for all $n > 2$. Thus $T(n) \in \Theta(n)$.

5 Problem 5

Solution For this problem, for each house, we will apply a divide and conquer binary search to find which school has the smallest scalar distance. If two schools have the same, we accept the one with the lower index.
The scalar distance is the absolute value of the distance from the west end of the street from the school and the house.
We are able to perform a binary search on the inputed array as it has all ready been sorted from least to greatest distance from the west end.
The binary search works by comparing the value of the scalar distance of the midpoint with the two points (if they exist) about it. It then recurses on the side of which side had the better scalar distance or picks the mid point if it's best. It "conquers" by picking the point and incrementing the Answer array. No merge is necessary as it is a search function.

```
binarySchoolSearch( hDist, S[u...r], A[0...m] )
{
   \\base case
   if( u == r)
      A[u] = A[u] + 1

   \\division
   mid = floor( (u+r) /2 )
   dist1 = | hDist - S[mid] |
   if( mid > u)
      dist2 = | hDist - S[mid -1]|
```

```
      if (dist2 <= dist1)
         binarySchoolSearch( hDist, S[u... mid-1], A[0...n] )
   if( mid < r)
      dist3 = | hDist - S[mid +1]|
      if( dist3 < dist1 )
         binarySchoolSearch( hDist, S[mid+1...r],A[0...n] )
   else
      A[mid] = A[mid] +1

   \\since we are searching there is no merge
}

FindBestSchool( H[0...n], S[0..m])
{
   A[0...m] \\an array initialized with all 0s
   for( houseIndex in H[0...n] )
      binarySchoolSearch( H[houseIndex], S[0...m], A[0...m] )

   bestSchool = Max( A[0],A[1]...A[m])
   return bestSchool
}
```

Utilize FindBestSchool and the index of the best school will be returned. Our basic operation is the comparison operator of the binary search. For it's recursion algorithm we have the following relationship:

$$T(m) = T(\lceil \tfrac{m}{2} \rceil) + 2$$

or

$$T(m) = T(\lfloor \tfrac{m}{2} \rfloor) + 2$$

which both give:

$$T(m) = T(\tfrac{m}{2}) + 2$$

We will use the Master's Theorem where:

$$a = 1, b = 2$$
$$\log_2(1) = 0$$
$$f(m) = 2$$

We will use the 2nd rule and it is trivial to see that $2 \in \Theta(constant)$. Thus: $T(m) \in \Theta(\lg(m))$.

This call is called for n houses. Giving the final growth relationship as: $\Theta(n \lg(m))$. However, the search doesn't have to recurse the whole way down if it finds (and often will) a best midpoint. So it's really $O(n \lg(m))$. Note, that finding the max of an array of n numbers can be done in $O(n)$ and so does not affect the growth relationship.