Assignment 5

Ayman Shahriar UCID: 10180260 Tutorial: T02

November 19, 2020

Question 1

First, note that:

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + T(\lfloor n/4 \rfloor) + T(\lfloor n/8 \rfloor) + n \\ &\leq T(n/2) + T(n/4) + T(n/8) + n \\ &= T(n/2) + T(n/4) + T(n/8) + n \end{aligned}$$

Step 1: Let us draw the recursion tree for this recurrence relation:

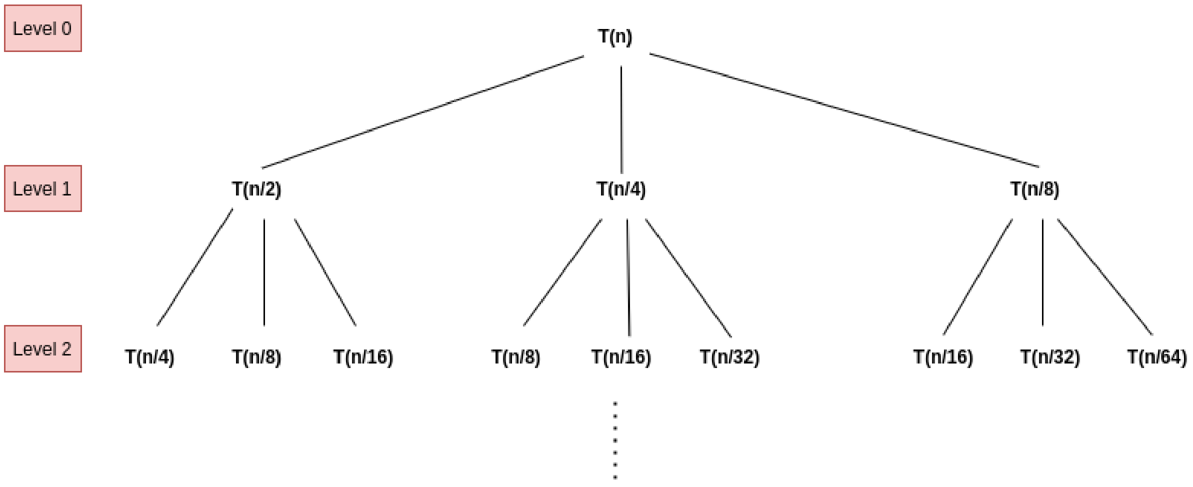

Figure 1: Recursion Tree

The number of levels in the recursion tree will be given by the height of the tree. Each $T(n)$ has 3 recursive calls, with the recursive call of the biggest size being $T(n/2)$.This means that the height of the recursive tree is bounded by $O(\log n)$.

Let us find the lower bound on the height of the tree.

Let's say for the last level x we have one leaf, so the total size of the sub-problems can be denoted as:

$$\begin{aligned}(7/8)^x &= 1\\ \text{Then } n &= (8/7)^x\\ \log n &= x(\log 8/7)\\ x &= \frac{\log n}{\log 8/7}\\ x &= \log_{8/7} n\\ x &= \Omega(\log n)\end{aligned}$$

So the height of the tree is in $\Theta(\log n)$

Step 2: Find the size of the sub-problems at each level

Level 0: $l_0 = n$

Level 1: $l_1 = (n/2) + (n/4) + (n/8) = (7/8)n$

Level 2:
$l_3 = (n/4)+(n/8)+(n/16)+(n/8)+(n/16)+(n/32)+(n/16)+(n/32)+(n/64)$
$= (49/64)n$
$= (7/8)^2 n$

We see a pattern:

$$l_i = \left(\frac{7}{8}\right)^i n$$

Remember that the number of levels in the recursion tree will be given by the height of the tree, and that the height of the tree is in $\Theta(\log n)$.
So taking the sum of all levels of recursion, we get:

$$T(n) = \sum_{i=0}^{\log n} \left(\frac{7}{8}\right)^i n = n \cdot \sum_{i=0}^{\log n} \left(\frac{7}{8}\right)^i$$

If we apply the rule

$$S_n = \sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

we get:

$$T(n) = n \cdot \sum_{i=0}^{\log n} \left(\frac{7}{8}\right)^i = \frac{(7/8)^{\log n + 1} - 1}{(7/8) - 1}$$

And since $(7/8) < 1$ and $\lim_{x\to\infty}(7/8)^{\log n+1} = 0$, then

$$\sum_{i=0}^{\infty}\left(\frac{7}{8}\right)^{i} = \frac{-1}{(-1/8)} = 8$$

So this shows that

$$n \cdot \sum_{i=0}^{\log n}\left(\frac{7}{8}\right)^{i} \leq \sum_{i=0}^{\infty}\left(\frac{7}{8}\right)^{i} \leq 8n$$

And so we guess that $T(n) = \Theta(n)$

b)

In order for $T(n) = O(n)$, we have to show $T(n) \leq dn$ for some constant d.
Let us see if $T(n) \leq 8n$ is good for $T(n) = T(n/2) + T(n/4) + T(n/8) + n$.

Applying the substitution method:

$$\begin{aligned}
T(n) &= T(n/2) + T(n/4) + T(n/8) + n \\
&\leq d \cdot (n/2) + d \cdot (n/4) + n \\
&\leq (4dn + 2dn + dn)/8 + (8n/8) \\
&\leq (7/8)dn + n \\
&\leq ((7/8)d + 1)n \\
&\leq dn \text{ (if d } > \text{ 8)}
\end{aligned}$$

And in order for $T(n) = \Omega(n)$, we have to show $T(n) \geq cn$ for some constant c.

Let c = 1.
Then, $T(n) = T(n/2) + T(n/4) + T(n/8) + n \geq 1n$, as required.

Thus, we have proved that $T(n) = O(n)$ and $T(n) = \Omega(n)$, which means that $T(n) = \Theta(n)$

Question 2

a) For this recurrence we have $a = 2$, $b = 2$, $f(n) = n^3 = \Theta(n^3) \therefore c = 3$;

Note that $c = 3 > 1 = \log_2 2 = log_b a$

So case 3 is applicable to this recurrence and $T(n) = \Theta(n^3)$

b) For this recurrence we have $a = 2$, $b = 2$, $f(n) = n = \Theta(n) = \Theta(n \cdot \log^0 n)$, so $c = 1$.

Note that $c = 1 = \log_2 2 = \log_b a$

So case 2 is applicable to this recurrence and
$T(n) = \Theta(n \log^{0+1} n) = \Theta(n \log n)$

c) For this recurrence we have $a = 2$, $b = 2$, $f(n) = n^{1/2} = \Theta(n^{1/2}) \therefore c = 1/2$

Note that $c = 0.5 < 1 = \log_2 2 = \log_b a$

So case 1 is applicable to this recurrence and $T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$

d) For this recurrence we first need to prove that $n^{(1/2)} \cdot \log^2 n = O(n^{(197/300)})$

Note that since $\log^2 n$ is polylogarithmic and $n^{(47/300)}$ is polynomial, then there is are constants d and k such that for all $n \geq d$, $\log^2 n \leq k \cdot n^{(47/300)}$.

Let $n_0 = d$, $c = k$.

Then $n^{(1/2)} \cdot \log^2 n \leq n^{(1/2)} \cdot kn^{(47/300)}$ (since $n \geq d$)
$\leq k \cdot n^{(197/300)}$

So by definition of Big-O,
$n^{(1/2)} \cdot \log^2 n = O(n^{(197/300)})$
$= O(n^{(2/3)-(1/100)})$
$= O(n^{\log_8 4-(1/100)})$ (since $\log_8 4 = (2/3)$)

So for this recurrence we have $a = 4$, $b = 8$,
$f(n) = O(n^{\log_8 4-(1/100)}) \therefore \epsilon = (1/100)$

Since $\epsilon > 0$, then case 1 is applicable to this recurrence and
$T(n) = \Theta(n^{\log_8 4}) = \Theta(n^{(2/3)})$

e) For this recurrence we have $a = 16$, $b = 7$, $f(n) = n^2 = \Theta(n^2) \therefore c = 2$.

Note that $c = 2 < 1.42 < \log_7 16 = \log_b a$

So case 1 is applicable to this recurrence and $T(n) = \Theta(n^{\log_7 16})$

Question 3

a)

Preconditions: Two lists $(a_1, \ldots, a_n)$ and $b_1, \ldots, b_n$, each list contains n distinct integers in ascending order.

Postconditions: Return an integer that is the median of all $2n$ elements of the two lists.

b) (Assume we are using a 1 based index)

```
Function Median(integer array A, integer array B, integer n)

    if n == 1: then
        return single element of A
    if n == 2: then
        Merge the two lists
        return the average of the middle two numbers

    // Compute seperate medians of A and B, this takes O(1)
    if n is even: then
        $m_a = (A[n/2] + A[(n/2) + 1])/2$
        $m_b = (B[n/2] + B[(n/2) + 1])/2$
    else
        $m_a = A[\lceil n/2 \rceil]$
        $m_b = B[\lceil n/2 \rceil)]$

    // Now compare the medians of A and B
    if $m_a == m_b$: then
        return $m_a$

    if $m_a > m_b$: then
        if n is odd: then
            Let A' be the sorted elements of A from 1 to $\lceil n/2 \rceil$
            Let B' be the sorted elements of B from $\lceil n/2 \rceil$ to n
        else
            Let A' be the sorted elements of A from 1 to $(n/2) + 1$
            Let B' be the sorted elements of B from $(n/2)$ to $n$
        m = Median(A', B', $\lfloor n/2 \rfloor + 1$)
        return m

    if $m_a < m_b$ then
        if n is odd: then
            Let A' be the sorted elements of A from $\lceil n/2 \rceil$ to $n$
            Let B' be the sorted elements of B from 1 to $\lceil n/2 \rceil$
        else
            Let A' be the sorted elements of A from $(n/2)$ to $n$
            Let B' be the sorted elements of B from 1 to $(n/2) + 1$
        m = Median(A', B', $\lfloor n/2 \rfloor + 1$)
        return m
```

c) We will prove that our algorithm returns the median of the two lists combined by strong induction on n, where n is the length of each list.

Base case:

(n = 1): In this case, the algorithm returns the average of the two elements present in the two lists, which is their median.

(n = 2): In this case, the algorithm merges the two lists, and returns the average of the middle two elements, which is the median of this combined list.

So in the base cases, our algorithm successfully computes and returns the median of the two lists.

Inductive Step:

Suppose the algorithm returns the median for any two lists of size n, where $1 \geq n \geq k$ and $k \geq 2$.

We want to show that the algorithm will return the median of all two lists of size $k+1$.

Let A, B be our input lists with sizes $k+1$.

Note that in our algorithm, we easily compute the separate medians of A and B by taking the middle element of each list if their length is odd, or by taking the average of the middle two elements of each list if their length is even.

Suppose $K+1$ is odd. Then:

Let $A = a_1, \ldots, m_a, \ldots, a_n$ where m_a is the median of A

Let $B = b_1, \ldots, m_b, \ldots, b_n$ where m_b is the median of B

Then we have 3 cases:

Case 1: $m_a == m_b$
In this case, since both arrays have the same median, it must mean that if the arrays were merged, then the median would still be the same. That is why for this case we just return one of the medians and end the algorithm.

Case 2: $m_a > m_b$
Since A and B are sorted, it means that the median of the combined array of A and B cannot be larger than m_a and cannot be smaller than m_b, so we do not need to search for the median within elements $[m_{a+1}, ..., a_n]$ and $[b_1, ..., m_{b-1}]$. So the median of the combined array must be within the elements $[m_b, ..., b_n]$ and $[a_1, ..., m_a]$, and that is why in this case we recursively compute the median of these two sub-arrays and return that as the median of the merged list.
And since $\lfloor (k+1)/2 \rfloor < k+1$, then the inductive hypothesis applies for the two recursive calls, and so our algorithm returns the correct median of A and B in this case.

Case 3: $m_a < m_b$
Since A and B are sorted, it means that the median of the combined array of A

and B cannot be larger than m_b and cannot be smaller than m_a, so we do not need to search for the median within elements $[m_{b+1}, ..., b_n]$ and $[a_1, ..., m_{a-1}]$. So the median of the combined array must be within the elements $[m_a, ..., a_n]$ and $[b_1, ..., m_b]$, and that's why in this case we recursively compute the median from these two sub-arrays and returns that as the median of our merged list. And since $\lfloor (k+1)/2 \rfloor < k+1$, then the inductive hypothesis applies for the two recursive calls, so our algorithm returns the correct median of A and B in this case.

So we have shown that when $k+1$ is odd, our algorithm computes the correct median.

Now suppose $K+1$ is even. Then:

Let $A = a_1, \ldots, m_{a1}, m_{a2} \ldots, a_n$ where $m_a = (m_{a1} + m_{a2})/2$ is the median of A

Let $B = b_1, \ldots, m_{b1}, m_{b2} \ldots, b_n$ where $m_b = (m_{b1} + m_{b2})/2$ is the median of B

Then we also have 3 cases:

Case 1: $m_a == m_b$
This will be the same as when $(k+1)$ is odd

Case 2: $m_a > m_b$
This will be similar to case 2 when $k+1$ is odd, except that the median cannot be larger than m_{a2} and cannot be smaller than m_{b1}, so the median of the combined array must be within the elements $[m_{b1}, ..., b_n]$ and $[a_1, ..., m_{a2}]$. So our algorithm recursively finds the median from these two sub-arrays and returns that as the median of our merged list. And since $\lfloor (k+1)/2 \rfloor < k+1$, then the inductive hypothesis applies for the two recursive calls, so our algorithm returns the correct median of A and B in this case.

Case 3: $m_a < m_b$
This will be similar to case 3 when $k+1$ is odd, except that the median cannot be larger than m_{b2} and cannot be smaller than m_{a1}, so the median of the combined array must be within the elements $[m_{a1}, ..., a_n]$ and $[b_1, ..., b_{a2}]$. So our algorithm recursively finds the median from these two sub-arrays and returns that as the median of our merged list. And since $\lfloor (k+1)/2 \rfloor < k+1$, then the inductive hypothesis applies for the two recursive calls, so our algorithm returns the correct median of A and B in this case.

So we have shown that when $k+1$ is odd, our algorithm computes the correct median, as required to prove the inductive step.

Thus, we have shown that in all cases, our algorithm returns the median of the merged list.

(Also note that in each recursive call, the size of each input array decreases by floor(n/2)+1, so our algorithm eventually terminates on all valid inputs.)

d) The runtime of our algorithm can be expressed as the following recurrence relation (where n is the size of each input array):

$$T(n) = T(\lfloor n/2 \rfloor + 1) + c \text{ (where c is a constant)}$$

This is the correct recurrence relation because:

- The algorithm calls itself recursively at most once (excluding the recursive calls that the recursive calls themselves make).
- Every time the algorithm recursively calls itself, the size of the input arrays decreases by $\lfloor n/2 \rfloor + 1$, so the runtime of each recursive call will be $T(\lfloor n/2 \rfloor + 1)$
- Except for the recursive call, every other step taken by the algorithm runs in constant time. This includes computing the seperate medians of the two input lists, comparing the medians of the two input lists, subdividing the two lists, and returning the median after the end of the algorithm, etc. So besides the recursive call, everything else in the algorithm takes a constant number of steps. So c represents the constant number of steps taken by the non-recursive components of the algorithm.

e) We will use the substitution method to show that $T(n) \leq k \cdot \log{(n-2)}$ for some constant k

$$\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor + 1) + c \\
&\leq k \log{(\lfloor n/2 \rfloor + 1 - 2)} + c \\
&\leq k \log{((n/2) - 1)} + c \\
&\leq k \log{((n-2)/2)} + c \\
&\leq k \log{(n-2)} - k\log 2 + c \\
&\leq k \log{(n-2)} \qquad \text{(this step holds for } k\log 2 \geq c \text{)}
\end{aligned}$$

So using the substitution method, we have proved that $T(n) \leq k \cdot \log{(n-2)}$ for some constant k, which means that $T(n) = O(\log n)$

Now we will use the substitution method to show that $T(n) \geq k \log n$ for some constant k

$$\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor + 1) + c \\
&\geq T(n/2) + c \quad \text{(since } \lfloor n/2 \rfloor + 1 \geq (n/2) \text{ for all } n \geq 1 \text{)} \\
&\geq k \log{(n)} + c \\
&\geq k \log n
\end{aligned}$$

So using the substitution method, we have proved that $T(n) \leq k \log n$, which means that $T(n) = \Omega(\log n)$

Since $T(n) = O(\log n)$ and $T(n) = \Omega(\log n)$, this means that $T(n) = \Theta(\log n)$.

Thus, we have proved that the runtime of our algorithm is in $\Theta(\log n)$