CS-GY 6033: Homework #4

Due on October 18, 2023

Professor Yi-Jen Chiang

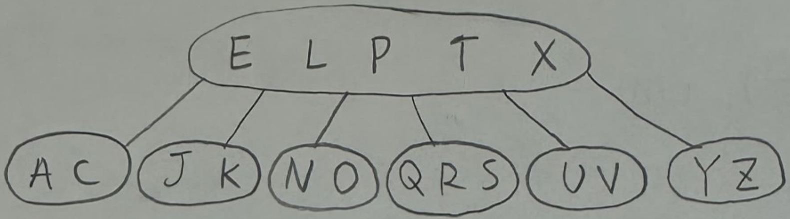
Runze Li
Nxxxxxxxx
rl50xx@nyu.edu

Tzu-Yi Chang
Nxxxxxxxx
tc39xx@nyu.edu

Jiayi Li
Nxxxxxxxx
jl156xx@nyu.edu

October 15, 2023

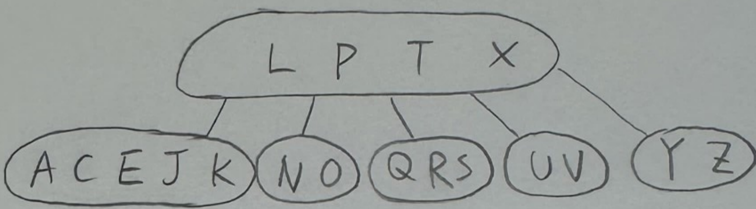
1. $t=3$

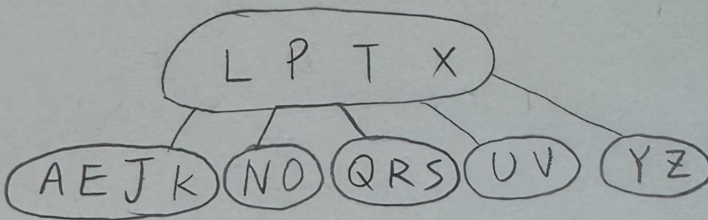


initial

delete C

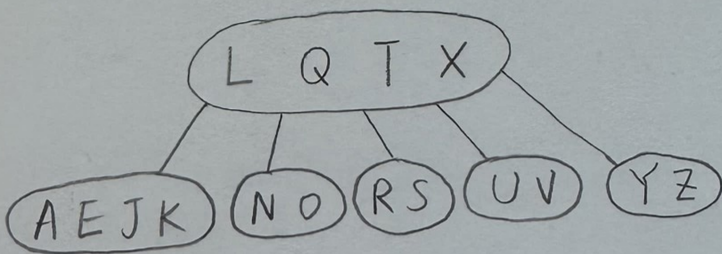
case 3b





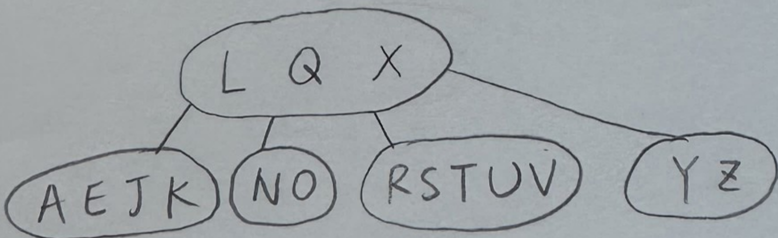
delete P

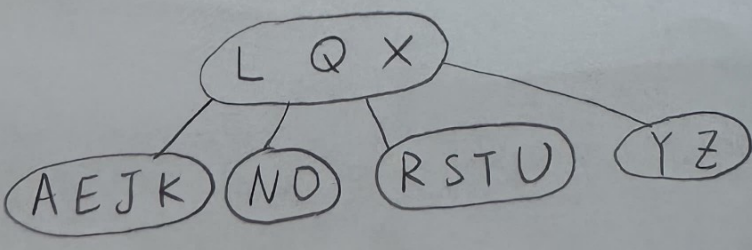
case 2b



delete V

case 3b





3. (a)

2. (a) $a=2 \quad b=2 \quad d=3$

$$\frac{a}{b^d} = \frac{2}{2^3} = \frac{1}{4} < 1$$

$\therefore T(n) = \Theta(n^d)$ applies

$\therefore T(n) = \Theta(n^3)$

(c) $a=8 \quad b=4 \quad d=\frac{3}{2}$

$$\frac{a}{b^d} = \frac{8}{4^{\frac{3}{2}}} = 1$$

$\therefore T(n) = \Theta(n^d \log n)$ applies

$\therefore T(n) = \Theta(n^{\frac{3}{2}} \log n)$

(b) $a=1 \quad b=\frac{10}{7} \quad d=2$

$$\frac{a}{b^d} = \frac{1}{(\frac{10}{7})^2} = \frac{49}{100} < 1$$

$\therefore T(n) = \Theta(n^d)$ applies

$\therefore T(n) = \Theta(n^2)$

(d) $a=5 \quad b=9 \quad d=\frac{1}{2}$

$$\frac{a}{b^d} = \frac{5}{9^{\frac{1}{2}}} = \frac{5}{3} > 1$$

$\therefore T(n) = \Theta(n^{\log_b a})$ applies

$\therefore T(n) = \Theta(n^{\log_9 5})$

3. (a)
$$\begin{aligned} T(n) &= T(n-2) + n^3 \\ &= T(n-4) + (n-2)^3 + n^3 \\ &= T(n-6) + (n-4)^3 + (n-2)^3 + n^3 \\ &= \cdots \\ &= T(n-2t) + (n-2t+2)^3 + \cdots + (n-2)^3 + n^3 \end{aligned}$$

when $n = 2(t+1)$

$$\begin{aligned} T(n) &= T(2) + 4^3 + 6^3 + \cdots (2t)^3 + (2t+2)^3 \\ &= C_1 + 2^3 \times [2^3 + 3^3 + \cdots + t^3 + (t+1)^3] \\ &= C_1 + 8(\sum_{k=1}^{t+1} k^3 - 1) \\ &= C_1 + 8 \times \frac{(t+1)^2 (t+2)^2}{4} - 8 \\ &= \Theta(t^4) \\ &= \Theta((\frac{n}{2} - 1)^4) \\ &= \Theta(n^4) \end{aligned}$$

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

$= 4[4T(\frac{n}{16}) + \frac{n}{4}\log^2\frac{n}{4}] + n\log^2 n$

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

$= 4^2[4T(\frac{n}{4^3}) + \frac{n}{4^2}\log^2\frac{n}{4^2}] + n\log^2\frac{n}{4} + n\log^2 n$

$= 4^3 T(\frac{n}{4^3}) + n\log^2\frac{n}{4^2} + n\log^2\frac{n}{4} + n\log^2 n$

$= \dots$

$= 4^t T(\frac{n}{4^t}) + n\sum_{k=0}^{t-1}\log^2\frac{n}{4^k}$

when $n = 4^t$, $t = \frac{1}{2}\log n$

$T(n) = nT(1) + n\sum_{k=0}^{t-1}(\log n - \log 4^k)^2$

$= c_1 n + n\sum_{k=0}^{t-1}\log^2 n - 2n\sum_{k=0}^{t-1}\log n \cdot \log 4^k + n\sum_{k=0}^{t-1}\log^2 4^k$

$= c_1 n + tn\log^2 n - 2n\log n\sum_{k=0}^{t-1}\log 4^k + n\sum_{k=0}^{t-1}k^2\log^2 4$

$= c_1 n + \frac{1}{2}n\log^3 n - 2n\log n \cdot \log(4^0 \times 4^1 \times \dots \times 4^{t-1}) + 4n\sum_{k=0}^{t-1}k^2$

$= c_1 n + \frac{1}{2}n\log^3 n - 2n\log n \cdot \log 4^{\frac{t(t-1)}{2}} + 4n \cdot \frac{(t-1)t(2t-1)}{6}$

$= c_1 n + \frac{1}{2}n\log^3 n - (\frac{1}{2}n\log^3 n - n\log^2 n) + \frac{2}{3}n \cdot (\frac{1}{4}\log^3 n -$
$\frac{3}{4}\log^2 n + \frac{1}{2}\log n)$

$= c_1 n + \frac{1}{6}n\log^3 n + \frac{1}{2}n\log^2 n + \frac{1}{3}n\log n$

$= \Theta(n\log^3 n)$

4. $T(n) = 3T(\frac{n}{3}) + \log_3 n$

Set $\log_3 n = m \Rightarrow n = 3^m$

Re-write $T(3^m) = 3T(\frac{3^m}{3}) + m$

$$T(3^m) = 3T(3^{m-1}) + m$$
$$= 3[3T(3^{m-2}) + (m-1)] + m$$
$$= 3^2 T(3^{m-2}) + 3(m-1) + m$$
$$= 3^2[3T(3^{m-3}) + (m-2)] + 3(m-1) + m$$
$$= 3^3 T(3^{m-3}) + 3^2(m-2) + 3(m-1) + m$$
$$= \cdots$$
$$= 3^k T(3^{m-k}) + 3^{k-1}(m-(k-1)) + \cdots + 3(m-1) + m$$

$$S = m + 3(m-1) + 3^2(m-2) + 3^3(m-3) + \cdots + 3^{k-1}(m-(k-1))$$
$$3S = 3m + 3^2(m-1) + 3^3(m-2) + \cdots + 3^{k-1}(m-(k-2)) + 3^k(m-(k-1))$$

By subtraction, we get

$$2S = -m + 3 + 3^2 + 3^3 + \cdots + 3^{k-1} + 3^k(m-(k-1))$$
$$S = \frac{1}{2}\left[\frac{3(1-3^{k-1})}{1-3} + 3^k(m-(k-1)) - m\right]$$
$$T(3^m) = 3^k T(3^{m-k}) + \frac{1}{2}\left[\frac{3(1-3^{k-1})}{1-3} + 3^k(m-(k-1)) - m\right]$$

Assume $3^{m-k} = 1 \Rightarrow m-k=0 \Rightarrow m=k$

$$T(3^m) = 3^m T(1) + \frac{1}{2}\left[\frac{3(1-3^{m-1})}{-2} + 3^m - m\right]$$

$$= 0 + \frac{1}{2}\left[-\frac{3}{2}\left(1-\frac{3^m}{3}\right) + 3^m - m\right]$$

$$\Rightarrow T(n) = \frac{1}{2}\left[-\frac{3}{2}\left(1-\frac{n}{3}\right) + n - \log_3 n\right)$$

$$= \frac{1}{2}\left[-\frac{3}{2} + \frac{1}{2}n + n - \log_3 n\right)$$

$$= -\frac{3}{4} + \frac{3}{4}n - \frac{1}{2}\log_3 n$$

$$= \frac{3}{4}n - \frac{1}{2}\log_3 n - \frac{3}{4}$$

5. 0) While $n \bmod 7 \neq 0$

$\{$ Find min and remove it.

If $k == 1$, return min

$O(n)$ else $k \leftarrow k-1$

$n \leftarrow n-1$

$\}$

At most $O(n) \cdot 6 = O(n)$ worst-case time to make ($n \bmod 7 == 0$)

Now $n \bmod 7 == 0$

$O(n)$ 1) Partition the current n items into $\frac{n}{7}$ groups of 7 items each.

$O(n)$ 2) Sort each group of 7 items, take the median from it. ($\frac{n}{7}$ medians)

$T(\frac{n}{7})$ 3) Apply the algorithm recursively on the $\frac{n}{7}$ medians and find the median X among them.

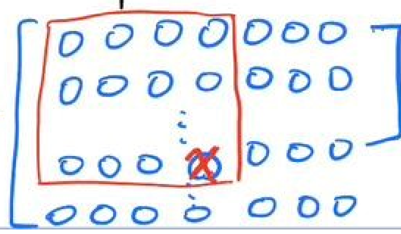
$O(n)$ 4) Compare all items with X to get sets S_1 (items $< X$) and S_2 (items $> X$)

$T(|S_1|)$ or $T(|S_2|)$ 5) Recurse on either S_1 or S_2 (or return X and stop)

Analysis: Consider the number of items "t" that are $\leq X$

$t \geq (\frac{n}{7} \cdot \frac{1}{2}) \cdot 4 = \frac{2n}{7}$

$\frac{n}{7}$



$\frac{n}{7} \cdot \frac{1}{2}$ groups

S_2 : items $> X$

$$|S_2| = n - t \leq n - \frac{2n}{7} = \frac{5n}{7}$$

$T(n) \leq T(\frac{n}{7}) + T(\frac{5n}{7}) + an$, a : const

Solve by Substitution :

Let $T(n) = cn$ for some constant c

$$cn \leq \frac{cn}{7} + \frac{5cn}{7} + an$$

$$= \frac{6cn}{7} + an$$

$$\Rightarrow \frac{c}{7} \leq a$$

$\Rightarrow$ Take $c = 7a$

$$T(n) = cn$$

$$= 7an$$

$$= O(n)$$

Problem 6

To design a divide-and-conquer algorithm to find the subsequence of consecutive element with the minimum product, we design a algorithm and the steps are as follows:

1. If $n = 1$, return the single element.
2. Split the sequence into two subsequences called **leftSeq** and **rightSeq** along the middle.
3. **Recursively** find the minimum product subsequences within the left and right subsequences.
4. In order to calculate the minimum product subsequence that includes both halves, we need to keep track of the minimum product, which can ensures that the minimum product subsequence spans across the midpoint. Because of some negative values, we need to maintain a maximum value *imax* and a minimum value *imin*. Every time we find a negative value, we need to exchange *imax* with *imin*, which can always get the maximum and minimum values. In this step, the time complexity is $O(n)$.
5. Finally, return the minimum product subsequence among the one in **leftSeq**, the one in **rightSeq** and the one that **crosses the midpoint**.

The pseudo code **Find_Min_Product_of_Subsequence** is as follows:

Algorithm 1 Find_Min_Product_of_Subsequence

Input: $arr, left, right$ ▷ An array with $right - left + 1$ items
Output: $minProduct, subsequence$ ▷ Return the subsequence with the minimum product

if $left == right$ **then**
  Return $\{minProduct : arr[left], subsequence : arr[left]\}$
end if
$mid \leftarrow (left + right)/2$
$leftProd, leftSeq \leftarrow Find\_Min\_Product\_of\_Subsequence(arr, left, mid)$
$rightProd, rightSeq \leftarrow Find\_Min\_Product\_of\_Subsequence(arr, mid + 1, right)$
$minProd \leftarrow INT\_MAX$
$imax, imin \leftarrow 1,\ 1$
for $i = left$ to $right$ **do**
  if $arr[i] < 0$ **then**
    exchange $imax$ with $imin$
  end if
  $imax \leftarrow max(imax * arr[i], arr[i])$
  $imin \leftarrow min(imin * arr[i], arr[i])$
  $minProd \leftarrow min(imin, minProd)$
end for
Find the minimum product among $leftProd, rightProd$ and $minProd$
Return minimum product and its corresponding subsequence $\{minProduct, subsequence\}$

Let $T(n)$ be the time complexity of the algorithm when given a sequence of the size n. The divide-and-conquer step involves two recursive calls on sequences of size $n/2$, so the time complexity for this step is $2 * T(n/2)$. The fourth step in which we calculate the minimum product subsequence spanning the midpoint takes $\Theta(n)$ time. Thus, the time complexity of the algorithm can be written as follows:

$$T(n) = 2 * T(n/2) + \Theta(n)$$

Using "Baby" Master Theorem, we can find $a = 2, b = 2$ and $d = 1$. So we can get $\frac{a}{b^d} = \frac{2}{2^1} = 1$.

Finally, we can get the time complexity of the algorithm $T(n) = O(n^d logn) = O(nlogn)$.

Problem 7

To find the diameter of a tree with an arbitrary number of children per internal node in worst case $O(n)$ time, we can use **divide-and-conquer** algorithm. The algorithm will maintain two pieces of information for each node: the maximum depth of a subtree rooted at that node and the diameter of the tree that can be reached from that node. Here's the algorithm **Find_Diameter**:

Algorithm 2 Find_Diameter

Input: *node*
Output: *diameter* ▷ Return the maximum diameter of T

```
diameter ← 0
Divide_And_Conquer(node, diameter)
Return diameter
```

Algorithm 3 Divide_and_Conquer

Input: *node*, &*diameter* ▷ *diameter* is a parameter
Output: *treeHeight* ▷ Return the maximum diameter of T

```
if node == NULL then
    Return 0
end if
maxHeight, secondHeight ← 0, 0
for each child of node do
    childHeight ← Divide_and_Conquer(child, diameter)
    if childHeight > maxHeight then
        secondHeight ← maxHeight
        maxHeight ← childHeight
    else if childHeight > secondHeight then
        secondHeight ← childHeight
    end if
end for
diameter ← max(diameter, maxHeight + secondHeight)
Return 1 + maxHeight
```

In this algorithm, we perform divide-and-conquer algorithm. For each node, we calculate the two deepest children for the diameter, and we return the maximum height of the tree.

In the Divide and Conquer algorithm, the time complexity is $kT(\frac{n}{k})$, where k is the maximum number of children in an internal node($k > 1$). When we get the $maxHeight$ and $secondHeight$, the time complexity is $O(k) = \Theta(1)$. So the time complexity of the algorithm is:

$$T(n) = kT(\frac{n}{k}) + \Theta(1)$$

Using "Baby" Master Theorem, we can find $a = k, b = k$ and $d = 0$. So we can get $\frac{a}{b^d} = \frac{k}{k^0} = k > 1$.

Finally, we can get the time complexity of the algorithm $T(n) = O(n^{log_b a}) = O(n)$.