Ravi Dayabhai

Problem Session 2

Problem 1-1. Solving recurrences

Derive solutions to the following recurrences in two ways: via a recursion tree **and** via Master Theorem. A solution should include the tightest upper and lower bounds that the recurrence will allow. Assume $T(1) \in \Theta(1)$.

(a) $T(n) = 2\,T(\frac{n}{2}) + O(\sqrt{n})$

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

(c) $T(n) = T(\frac{n}{3}) + T(\frac{n}{4}) + \Theta(n)$ assuming $T(a) < T(b)$ for all $a < b$

(SEE NEXT FEW PAGES)

Problem 1-2. Stone Searching

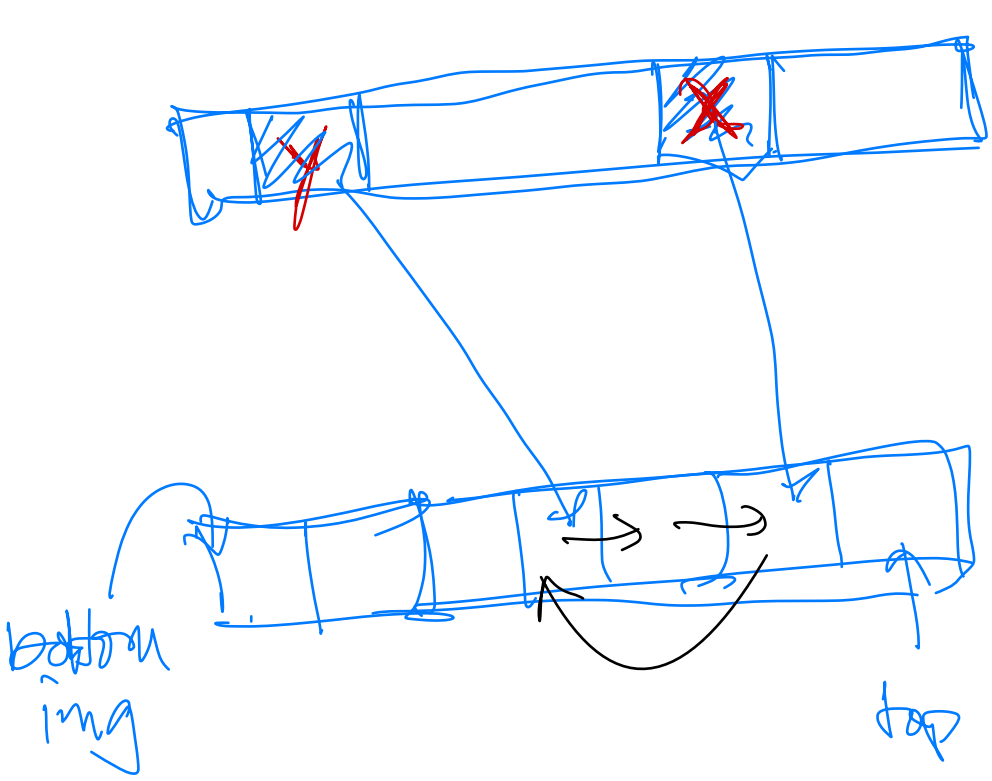
Sanos is a supervillain on an intergalactic quest in search of an ancient and powerful artifact called the Thoul Stone. Unfortunately she has no idea what planet the stone is on. The universe is composed of an infinite number of planets, each identified by a unique positive integer. On each planet is an oracle who, after some persuasion, will tell Sanos whether or not the Thoul Stone is on a planet having a strictly higher planet identifier than their own. Interviewing every oracle in the universe would take forever, and Sanos wants to find the Thoul Stone quickly. Supposing the Thoul Stone resides on planet k, describe an algorithm to help Sanos find the Thoul Stone by interviewing at most $O(\log k)$ oracles.

Problem 1-3. Collage Collating

Fodoby is a company that makes customized software tools for creative people. Their newest software, Ottoshop, helps users make collages by allowing them to overlay images on top of each other in a single document. Describe a database to keep track of the images in a given document which supports the following operations:

1. `make_document()`: construct an empty document containing no images
2. `import_image(x)`: add an image with unique integer ID `x` to the top of the document
3. `display()`: return an array of the document's image IDs in order from bottom to top
4. `move_below(x, y)`: move the image with ID `x` directly below the image with ID `y`

Operation (1) should run in worst-case $O(1)$ time, operations (2) and (3) should each run in worst-case $O(n)$ time, while operation (4) should run in worst-case $O(\log n)$ time, where n is the number of images contained in a document at the time of the operation.



Had the right idea! Skipped implementation details due to time. ONE GOTCHA:

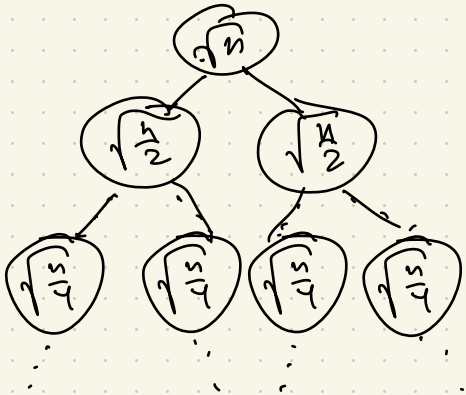
This should be doubly linked list so move_below can run in O(log n)!

① A (a) $T(n) = 2T(\frac{n}{2}) + O(\sqrt{n})$

By Master Method: $a = 2, b = 2, d = \frac{1}{2} \Rightarrow a > b^d$

$a > b^d \Rightarrow O(n)$

By recursion tree:



$$T(n) \le \sum_{i=0}^{\log_2 n} 2^i \cdot \left(\frac{n}{2^i}\right)^{\frac{1}{2}}$$

$$\le cn^{\frac{1}{2}} \cdot \sum_{i=0}^{\log_2 n} 2^i \cdot 2^{-\frac{1}{2}i}$$

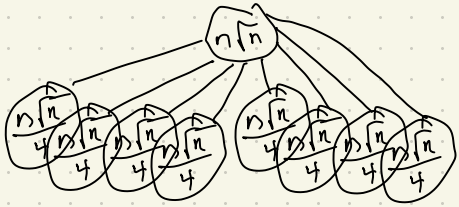
$$\le cn^{\frac{1}{2}} \cdot \sum_{i=0}^{\log_2 n} 2^{\frac{1}{2}i} = O\left(n^{\frac{1}{2}} \cdot n^{\frac{1}{2}\log_2 2}\right)$$

$= O(n)$, $\Omega(n) \Rightarrow \Theta(n)$

(Θ(1) work at each of n leaves gives lower bound)

① B (b) $T(n) = 8T(\frac{n}{4}) + O(n\sqrt{n})$

By recursion tree:



$$T(n) \le cn\sqrt{n} \cdot \sum_{i=0}^{\log_4 n} 8^i \cdot \left(\frac{1}{4^i}\right)^{\frac{3}{2}}$$

$$\le cn\sqrt{n} \sum_{i=0}^{\log_4 n} 2^{3i} \cdot 2^{-2i\left(\frac{3}{2}\right)}$$

$= O(n\sqrt{n}\log n)$

By Master Method: $a = 8, b = 4, d = \frac{3}{2} \Rightarrow a = b^d \Rightarrow$

$O(n\sqrt{n} \cdot \log n)$

① (c) $T(n) = T(\frac{n}{3}) + T(\frac{n}{4}) + \Theta(n)$ assuming $T(a) < T(b)$ for all $a < b$

By Akra-Bazzi:

(conditions met) first, find suitable p such that $1 = (\frac{1}{3})^p + (\frac{1}{4})^p$. Then evaluate the integration bounds on $T(x) = \Theta\left(x^p\left(\int_1^x \frac{u}{u^{p+1}}\,du + 1\right)\right)$.

$$= \Theta\left(x^p + x^p\int_1^x u^{-p}\,du\right) = \Theta\left(x^p + x^p\cdot\left(\frac{1}{1-p}\right)u^{1-p}\Big|_1^x\right)$$

$$= \Theta\left(x^p + x^p\left(\left(\frac{1}{1-p}\right)\left(x^{1-p} - 1\right)\right)\right) = \Theta\left(\frac{x - px^p}{1-p}\right).$$

Noticing that p must be negative and $0 < |p| < 1$, we're left with: $\Theta(x) \Rightarrow$ $T(n) = \Theta(n)$

By recursion tree: Using the fact that $T(a) < T(b)$ $\forall\, a, b$ $a < b$ means $T(n) \le 2T(\frac{n}{3}) + \Theta(n)$. (So, we expect (by Master Method) that $T(n) = O(n)$.)

$$T(n) \le cn \cdot \sum_{i=0}^{\log_3 n} 2^i \cdot \left(\frac{1}{3^i}\right) = cn \cdot \sum_{i=0}^{\log_3 n} \left(\frac{2}{3}\right)^i = O(n).$$

By using the fact that $T(\frac{n}{3}) > T(\frac{n}{4})$, the lower bound $\Omega(T(n))$ can be found by solving:

$T(n) \ge T(\frac{n}{4}) + \Theta(n) \Rightarrow$ $\Omega(n)$, so $T(n) = \Theta(n)$.

② Sonos can interview oracles, starting at planet 2^0, and then repeatedly visit the next exponentially higher planet until an oracle doesn't say "higher", say at planet $n = 2^m$. Binary search the 2^{m-1} planets between planets 2^{m-1} and n. The algorithm will take $m = \log n$ steps to find a planet $n = kc$ and $n \neq k$ only differ by a constant factor c in $(\frac{1}{2}, 1]$, so the runtime of the algorithm is given by: $O(\log 2^m + \log 2^{m-1})$
$= O(m) = O(\log n) = O(\log kc) = O(\log k)$.

④ [4,5,6,2,3,1,4,1,1,1]

Problem 1-4. Brick Blowing

Porkland is a community of pigs who live in n houses lined up along one side of a long, straight street running east to west. Every house in Porkland was built from straw and bricks, but some houses were built with more bricks than others. One day, a wolf arrives in Porkland and all the pigs run inside their homes to hide. Unfortunately for the pigs, this wolf is extremely skilled at blowing down pig houses, aided by a strong wind already blowing from west to east. If the wolf blows in an easterly direction on a house containing b bricks, that house will fall down, along with every house east of it containing strictly fewer than b bricks. For every house in Porkland, the wolf wants to know its **damage**, i.e., the number of houses that would fall were he to blow on it in an easterly direction.

(a) Suppose $n = 10$ and the number of bricks in each house in Porkland from west to east is `[34, 57, 70, 19, 48, 2, 94, 7, 63, 75]`. Compute for this instance the damage for every house in Porkland.

(b) A house in Porkland is **special** if it either (1) has no easterly neighbor or (2) its adjacent neighbor to the east contains at least as many bricks as it does. Given an array containing the number of bricks in each house of Porkland, describe an $O(n)$-time algorithm to return the damage for every house in Porkland **when all but one house** in Porkland is special.

(c) Given an array containing the number of bricks in each house of Porkland, describe an $O(n \log n)$-time algorithm to return the damage for every house in Porkland.

(d) Write a Python function `get_damages` that implements your algorithm.

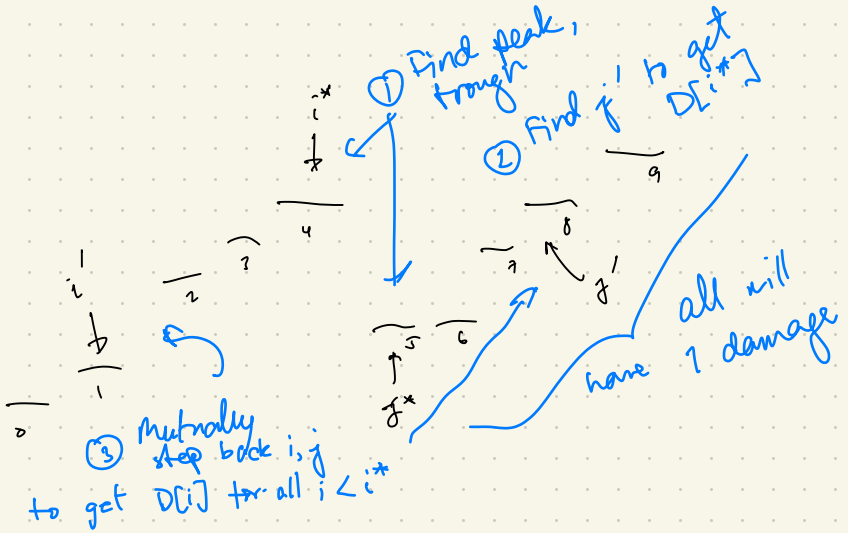
(SEE NEXT PAGE)

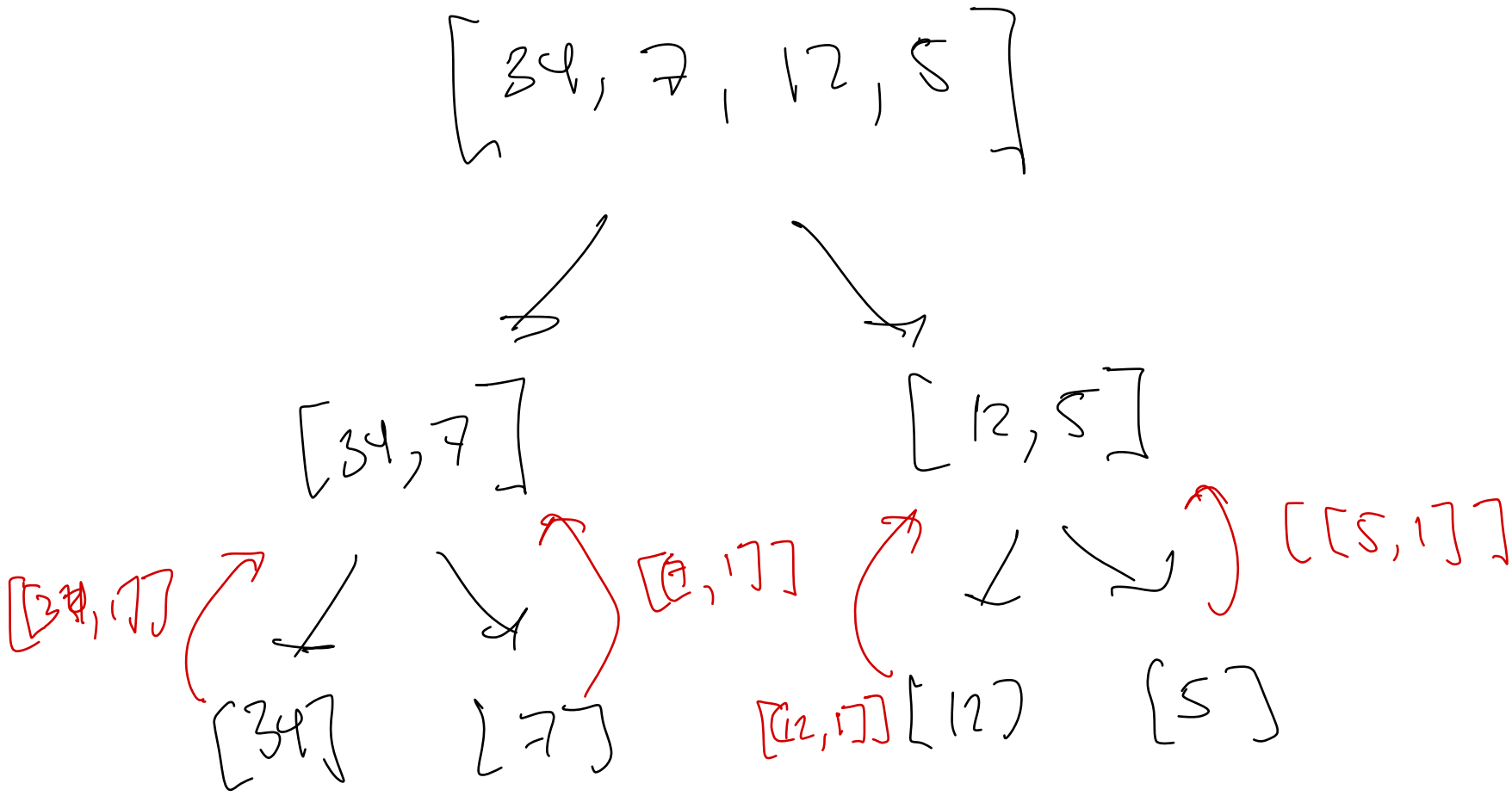
SEE CODE

```python
def get_damages(H):
    '''
    Input:  H | list of bricks per house from west to east
    Output: D | list of damage per house from west to east
    '''
    D = [1 for _ in H]
    ##################
    # YOUR CODE HERE #
    ##################
    return D
```

This is basically modified merge sort that counts inversions across the two sorted halves in each "merge" step; keep track of original indices of houses to update D with incremental inversions.

④ (B) Initialize two pointers $i=0$, $j=1$ (note: we don't need to worry about case where $\text{len}(A)<2$ because this case never produces exactly one non-special house). Keep track of $\Delta = A[j]-A[i]$. Increment i,j until $\Delta \geq 0$ or $j = \text{len}(A)$. Once $\Delta \geq 0$, set $i^* = i$, $j^* = j$. Increment j until $A[j] \geq A[j^*]$ or $j = \text{len}(A)$; call this j'. We can initialize our damage array D with all 1s. $D[i^*] += j' - i^* - 1$, so we just have to calculate damages for indices before i^*. To do this we can decrement i (if possible; if not, we're done); for this i, we can then decrement j until $A[i] > A[j]$ or $j = i^*$ (in which case we're done). If $A[i] > A[j]$ where $j > i^*$, $D[i] += j - i^*$. We can then repeat this process until sooner of $i<0$ or $j=i^*$. In the worst case, it will take $n-1$ steps to find i^*, j^* and then at most n steps to find damages for all $i < i^* \Rightarrow O(n)$ time complexity.





7, 34, 12, 2

[[7,1], [34,2]]

[[5,1], [12,2]]