MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

EECS 376 Discussion

Daphne Tsai

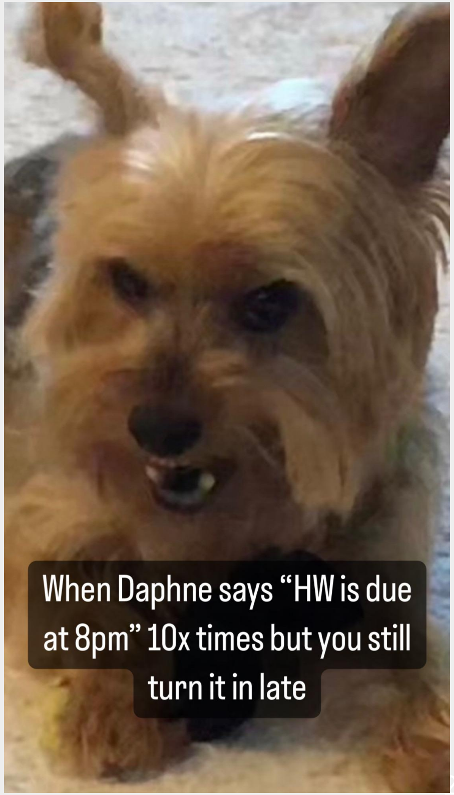
Week 2: Master Theorem, Potential Function, Divide+Conquer

9/8/23

Friday 11:30am @ NAME 138

Important Info

- Homework due Wednesdays 8pm- get it done!
- eecs376.org + Google Drive
- Change your umich password!



How to reach me

- Post questions on Piazza (I mostly handle logistics questions)
- My OH: Thursday 12-2:30pm and Friday 10-11am @ BBB atrium
- Join the OH queue (Find the link on eecs376.org)
- Email: dvtsai@umich.edu

Today + announcements

- Potential Function + Divide and Conquer
- The cool theory stuff is starting :)
- HW2 has been released- come to OH for help!
- Highly recommended to type in LaTeX (lah-tech)
- MIDAS summit! (Not sponsored I just like AI)

Master Theorem

$$\text{Let } T(n) = kT(n/b) + \Theta(n^d)$$

$$T(n) = \begin{cases} O(n^d) & \text{if } k/b^d < 1 \\ O(n^d \log n) & \text{if } k/b^d = 1 \\ O(n^{\log_b k}) & \text{if } k/b^d > 1 \end{cases}$$

Master Theorem w/ Log Factors

$$T(n) = k \cdot T(n/b) + n^d \log^w n,$$

where $k \geq 1, b > 1, d \geq 0, w \geq 0.$

$$T(n) = \begin{cases} \Theta(n^d \log^w n) & \text{if } \log_b k < d, \\ \Theta(n^d \log^{w+1} n) & \text{if } \log_b k = d, \\ \Theta(n^{\log_b k}) & \text{if } \log_b k > d. \end{cases}$$

Provide a big-O bound for $T(n) = 9T(n/3) + n^2 \log^2 n$.

$$T(n) = k \cdot T(n/b) + n^d \log^w n,$$

$$T(n) = \begin{cases} \Theta(n^d \log^w n) & \text{if } \log_b k < d, \\ \Theta(n^d \log^{w+1} n) & \text{if } \log_b k = d, \\ \Theta(n^{\log_b k}) & \text{if } \log_b k > d. \end{cases}$$

Consider the sorting algorithm *slowsort*, which can be represented with the following pseudocode. What is the most precise recurrence relation for the time complexity? What does the Master Theorem give for this relation?

```
function SLOWSORT(A[1, 2, ..., n]) // n is length of A
    SLOWSORT(A[1, ..., ⌊n/2⌋]) // sort both halves of the array recursively
    SLOWSORT(A[⌊n/2⌋ + 1, ..., n])
    if A[⌊n/2⌋] > A[n] then // largest item in first half is greater than largest in the second
        swap A[⌊n/2⌋] and A[n] // put largest item in the unsorted array at the end
    SLOWSORT(A[1, ..., n − 1]) // sort the entire array minus one element recursively
    return
```

Potential Function

- Terminates or runs forever
- How do we show this?
- s needs to reach a lower bound in a finite amount of steps
- s needs to strictly decrease on each step of the algorithm

$x \leftarrow \text{input}()$
$y \leftarrow \text{input}()$
while $x > 0$ and $y > 0$ **do**
    $z \leftarrow \text{input}()$
    if z is even **then**
        $x \leftarrow x - 1$
        $y \leftarrow y + 1$
    else
        $y \leftarrow y - 1$

```
x ← input()
y ← input()
while x > 0 and y > 0 do
    z ← input()
    if z is even then
        x ← x − 1
        y ← y + 1
    else
        y ← y − 1
        x ← x + 1 // This line differs
```

Divide and Conquer

- Divide the problem into smaller subproblems
- Subproblems do not need to overlap

Analyze the time complexity of MajorityElement and give the asymptotic time complexity as a closed-form solution.

$$\text{Let } T(n) = kT(n/b) + \Theta(n^d)$$

function $\textsc{MajorityElement}(A[1, 2, \ldots, n])$
    if $n = 1$ **then return** $A[1]$
    $x \leftarrow \textsc{MajorityElement}(A[1, \ldots, \lfloor \frac{n}{2} \rfloor])$
    $y \leftarrow \textsc{MajorityElement}(A[\lfloor \frac{n}{2} \rfloor + 1, \ldots, n])$
    if $x \neq \emptyset$ **then**
        iterate over A, counting the number of occurrences of x
        if the number of occurrences of x in A is $> \frac{n}{2}$ **then return** x
    if $y \neq \emptyset$ **then**
        iterate over A, counting the number of occurrences of y
        if the number of occurrences of y in A is $> \frac{n}{2}$ **then return** y
    return $\emptyset$

$$T(n) = \begin{cases} O(n^d) & \text{if } k/b^d < 1 \\ O(n^d \log n) & \text{if } k/b^d = 1 \\ O(n^{\log_b k}) & \text{if } k/b^d > 1 \end{cases}$$