COMS 311: Homework 2
Due: Feb 23, 11:59pm
Total Points: 50

Late submission policy. Any assignment submission that is late by not more than two business days from the deadline will be accepted with 20% penalty for each business day. That is, if a homework is due on Friday at 11:59 PM, then a Monday submission gets 20% penalty and a Tuesday submission gets another 20% penalty. After Tuesday no late submissions are accepted.

Submission format. Homework solutions will have to be typed. You can use word, LaTeX, or any other type-setting tool to type your solution. Your submission file should be in pdf format. Do **NOT** submit a photocopy of handwritten homework except for diagrams that can be hand-drawn and scanned. We reserve the right **NOT** to grade homework that does not follow the formatting requirements. Name your submission file: `<Your-net-id>-311-hw2.pdf`. For instance, if your netid is `asterix`, then your submission file will be named `asterix-311-hw2.pdf`. Each student must hand in their own assignment. If you discussed the homework or solutions with others, a list of collaborators must be included with each submission. Each of the collaborators has to write the solutions in their own words (copies are not allowed).

General Requirements

- When proofs are required, do your best to make them both clear and rigorous. Even when proofs are not required, you should justify your answers and explain your work.
- When asked to present a construction, you should show the correctness of the construction.

Some Useful (in)equalities

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $2^{\log_2 n} = n$, $a^{\log_b n} = n^{\log_b a}$, $n^{n/2} \leq n! \leq n^n$, $\log x^a = a \log x$
- $\log(a \times b) = \log a + \log b$, $\log(a/b) = \log a - \log b$
- $a + ar + ar^2 + ... + ar^{n-1} = \frac{a(r^n-1)}{r-1}$
- $1 + \frac{1}{2} + \frac{1}{2^2} + ... + \frac{1}{2^n} = 2(1 - \frac{1}{2^{n+1}})$
- $1 + 2 + 4 + ... + 2^n = 2^{n+1} - 1$

---

1. (**10 pts**) Using a table size of 11, and a hash function $h(x) = 3x + 2$ for positive integers, draw the resulting hash table array when the integers

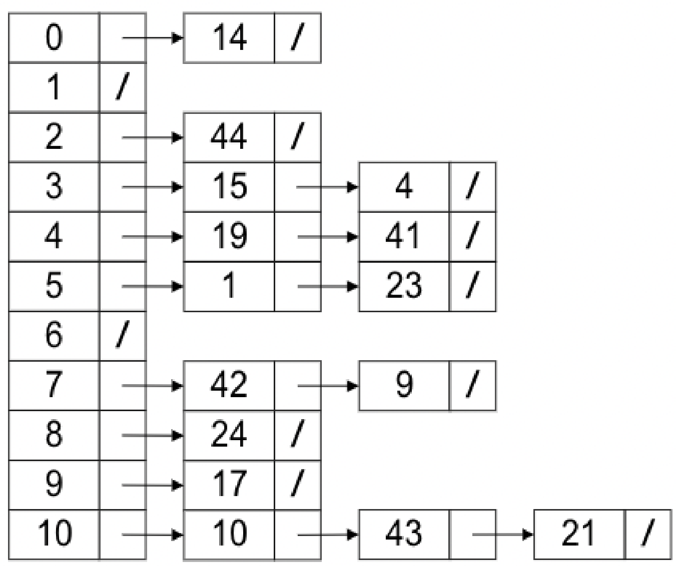
$$\mathbf{1, 17, 10, 15, 14, 43, 4, 21, 23, 24, 19, 41, 42, 9, 44}$$

are added to an empty hash table in this order. Note that for this hash function, x is stored in the table at index $h(x)$ % n, where n is the table size.

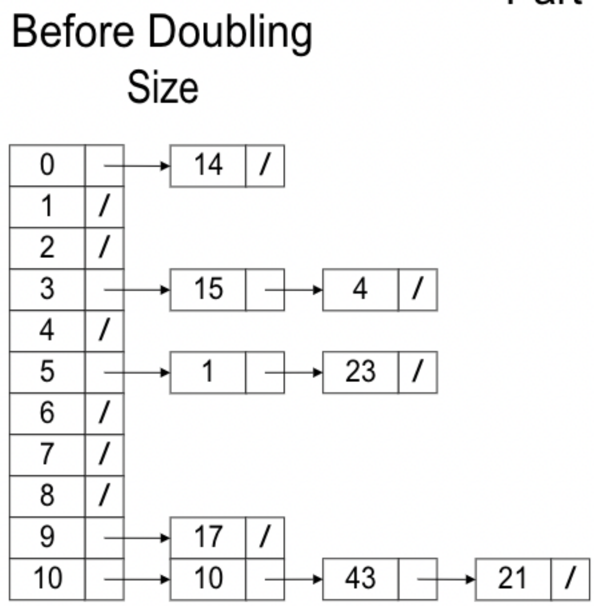
   (a) Using a chaining hash table that is not enlarged when elements are added.

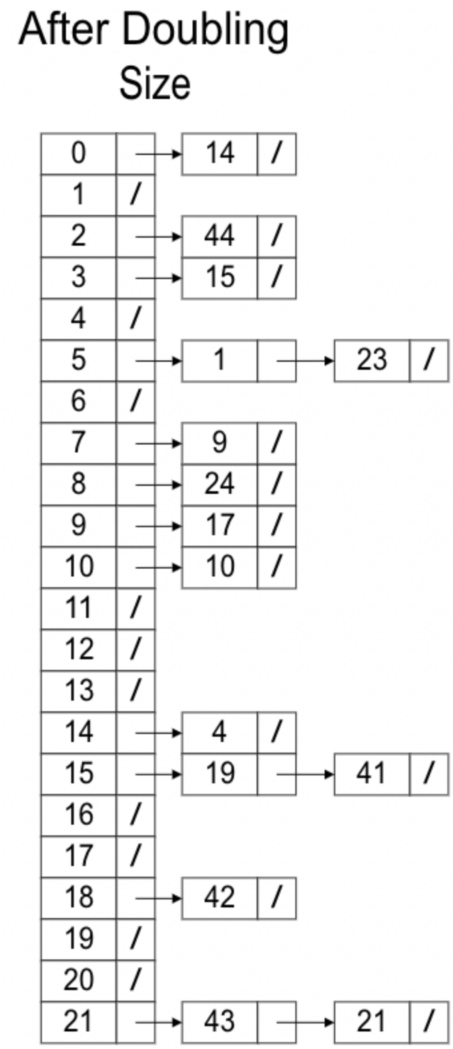
   (b) Using a chaining hash table that is doubled in size when the table reaches a load factor of $\alpha > 0.75$.

Problem 1
Part a



Problem 1
Part b





2. (**10 pts**) Consider the following recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 7 \cdot T(\frac{n}{2}) + 1 & \text{if } n > 1 \end{cases}$$

(a) Use the Master theorem to show that $T(n) \in \Theta(n^{\log_2(7)})$.

(b) Use induction to prove that $T(n) = \frac{1}{6}(7n^{\log_2(7)} - 1)$.

(a)
$T(n) = 7 \cdot T(\frac{n}{2}) + 1$
a = 7, b = 2, f(n) = 1
For the master theorem, we must compare $n^{log_b(a)}$ to $f(n)$
$n^{log_2(7)}$ is polynomial greater than 1, $f(n) \cdot n^d = n^{log_2(7)}$ for d = $n^{log_2(7)}$, so case 1 applies.
Case one says $T(n) = \Theta(n^{log_b(a)})$, hence: $T(n) = \Theta(n^{log_2(7)})$

(b)

Base Case: T(1) = 1, $(1)(6(7 + 1^{log_2 7} - 1)) = (1/6) \cdot (7 \cdot 1 - 1) = (1/6)(6) = 1$
1 = 1 so base case holds.
Inductive Hypothesis: $\forall k \geq 1$, T(k) = $\frac{1}{6}(7^{log_2 7} - 1)$

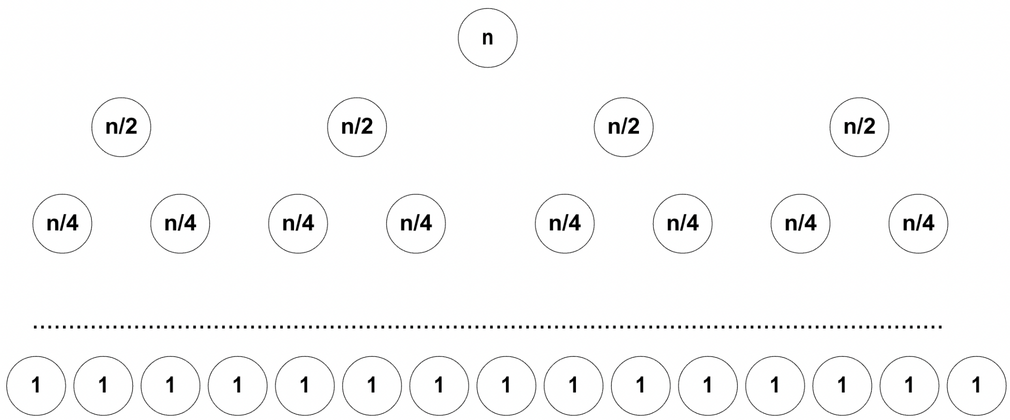
We will be proving T(k + 1) = $\frac{1}{6} \cdot 7(k+1)^{log_2(T)} - 1$

T(k+1) = $\frac{1}{6} \cdot (7(k+1)^{log_2 7} - 1)$
T(k+1) = $7 \cdot T(\frac{k+1}{2} + 1)$
$= 7 \cdot (\frac{1}{6}(7(\frac{k+1}{2})^{log_2(7)} - 1) + 1$ **via IH**
$= 7 \cdot (\frac{1}{6}(7(\frac{(k+1)^{log_2(7)}}{7}) - 1) + 1$
$= 7 \cdot (\frac{1}{6}(k+1)^{log_2 7} - 1) + 1$
$= \frac{1}{6}(7(k+1)^{log_2 7} - 1)$

Therefore T(k + 1) = $\frac{1}{6}(7(k+1)^{log_2 7} - 1)$ using induction.

3. (**10 pts**) Without using the Master Theorem, give the asymptotic upper bounds on the following recurrence relations. Note that methods found in chapter 4 of the text may be useful.

$$(a) : T(n) = \begin{cases} 1 & \text{if } n < 2 \\ 4 \cdot T(\frac{n}{2}) + n & \text{if } n \geq 2 \end{cases}$$



Level	Problem Size	Number of Nodes	Work Per Node	Level Total
1	n	1	n	n
2	n/2	4	n/2	2n
3	n/4	16	n/4	4n
...	...	...	...	...
i	n/(2^i-1)	4^i	n/(2^i-1)	2^i n

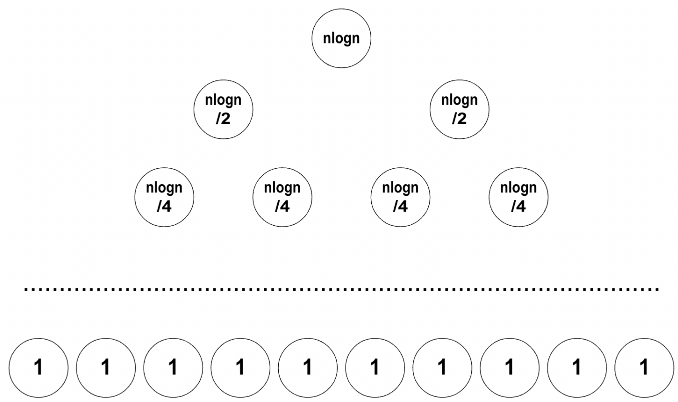
Using this tree and table, we can write a summation using the level total in terms of i, where i is the variable that increases. Since the tree will reach the level where work per node is one at depth log$n$, this summation will go from i = 0 to log$n$.

$\sum_{i=0}^{logn} 2^i n = n \cdot \sum_{i=0}^{logn} 2^i = n \cdot (2n - 1) = 2n^2 - n = \Theta(n^2)$

This summation will hold because each the work per leaf decreases by a factor of two but the number of nodes per layer increases by a factor of 4, as seen in the recursion tree diagram. This is further shown in the table, where the total work per layer column is $2^i n$ work per layer.

Hence, T(n) = $\Theta(n^2)$.

$$(b) : T(n) = \begin{cases} 1 & \text{if } n < 2 \\ 2 \cdot T(\frac{n}{2}) + n \log n & \text{if } n \geq 2 \end{cases}$$



Level	Problem Size	Number of Nodes	Work Per Node	Level Total
1	n log n	1	n log n	n log n
2	(n log n) / 2	2	(n log n) / 2	n log n
3	(n log n) / 4	4	(n log n) / 4	n log n
...	...	...	...	n log n
i	(n log n) / 2^(i-1)	2^(i -1)	(n log n) / 2^(i-1)	n log n

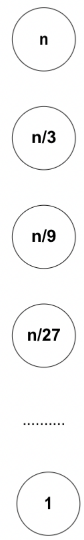
For this tree, we see that a and b are equal, so the tree work per layer is consistent. Each layer requires $n\log n$ work because the number of nodes increase at the same rate that the problem size decreases.

Since the work per layer is consistent and we can calculate the depth ($\log n$, the problem size is halved each layer), we can multiply these to find the total runtime.

Work per layer $\cdot$ number of layers $= n \log n \cdot \log n = n \log^2 n$.

Hence, T(n) $= \Theta$ $(n \log^2 n)$.

$$(c) : T(n) = \begin{cases} 1 & \text{if } n < 3 \\ T(\frac{n}{3}) + n & \text{if } n \geq 3 \end{cases}$$



Level	Problem Size	Number of Nodes	Work Per Node	Level Total
1	n	1	n	n
2	n / 3	1	n / 3	n / 3
3	n / 9	1	n / 9	n / 9
...	...	1	...	...
i	n / (3^i-1)	1	n / (3^i-1)	n / (3^i-1)

For this problem, we need to use a summation to find the total runtime because the total work per layer changes. This summation will go from $i = 0$ to $\log_3 n$ and will sum $\frac{n}{3^i}$, the work per layer total for layer i.

$\sum_{i=0}^{log_3 n} \frac{n}{3^i} = \frac{1}{2}(3n - 1) = \Theta(n)$

Therefore T(n) $= \Theta(n)$

$$(d): T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 9 \cdot T(\frac{n}{3}) + n^{2.5} & \text{if } n \geq 3 \end{cases}$$

$$(e): T(n) = \begin{cases} 1 & \text{if } n < 3 \\ T(n-2) + n^2 & \text{if } n \geq 3 \end{cases}$$

4. (**10 pts**) If Possible, use the Master theorem to give bounds on the following recurrence relations.

$$(a): \quad T(n) = \begin{cases} 1 & \text{if } n < 2 \\ 4 \cdot T(\frac{n}{2}) + n & \text{if } n \geq 2 \end{cases}$$

a = 4, b = 2, $f(n) = n$
$n^{log_b(a)} = n^{log_2(4)} = n^2 \rightarrow n \leq n^2,\ f(n) \leq n^{log_b(a)}$

Case 1 of Master theorem applies here.
Case 1: If $\exists\ \epsilon > 0$ such that $f(n) \in n^{log_b(a)-\epsilon}$, then $T(n) = \Theta(n^{log_b(a)})$.

Since $f(n) \in \Theta(n^{log_b(a)-\epsilon}) = n \in \Theta(n^{2-\epsilon}) = n \in \Theta(n^{2-1}) = n \in \Theta(n)$, so case 1 holds.

$T(n) \in \Theta(n^{log_b(a)}) = \Theta(n^{log_2(4)})$

Hence $T(n) \in \Theta(n^2)$

$$(b): \quad T(n) = \begin{cases} 1 & \text{if } n < 2 \\ 2 \cdot T(\frac{n}{2}) + n \log n & \text{if } n \geq 2 \end{cases}$$

a = 2, b = 2, $f(n) = n$ log n
$n^{log_b(a)} = n^{log_2(2)} = n^1 = n$

Case 2 applies here
Case 2: If $\exists\ k \geq 0$ such that $f(n) \in \Theta(n^{log_b(a)} \cdot log^k n)$, then $T(n) \in \Theta(n^{log_b(a)} \cdot log^{k+1} n)$

$f(n) \in \Theta(n^{log_b(a)} \cdot log^k n) = n$ log $n \in \Theta(n^{log_2(2)} \cdot log^k n) = n$ log $n \in \Theta(n^1 \cdot log^1 n)$
$= n$ log $n \in \Theta(n$ log $n)$, so case 2 holds.

Hence $T(n) \in \Theta(n^{log_b(a)} \cdot log^{k+1} n)$
$T(n) \in \Theta(n^{log_2(2)} \cdot log^{1+1} n)$

Therefore $T(n) \in \Theta(n \cdot log^2 n)$

$$(c): \quad T(n) = \begin{cases} 1 & \text{if } n < 3 \\ 3 \cdot T(\frac{n}{3}) + n & \text{if } n \geq 3 \end{cases}$$

a = 3, b = 3, $f(n) = n$
$n^{log_b(a)} = n^{log_3(3)} = n^1 = n$

Case 2 applies here
Case 2: If $\exists\ k \geq 0$ such that $f(n) \in \Theta(n^{log_b(a)} \cdot log^k n)$, then $T(n) \in \Theta(n^{log_b(a)} \cdot log^{k+1} n)$

$f(n) \in \Theta(n^{log_b(a)} \cdot log^k n) = n \in \Theta(n^{log_3(3)} \cdot log^0 n) = n \in \Theta(n \cdot 1) = n \in \Theta(n)$, so case 2 holds.

Therefore $T(n) \in \Theta(n^{log_b(a)} \cdot log^{k+1} n) = T(n) \in \Theta(n^{log_3(3)} \cdot log^{0+1} n) = T(n) \in \Theta(n^1 \cdot log^1 n) = T(n) \in \Theta(n \cdot logn)$

Hence $T(n) \in \Theta(n \text{ log } n)$

$$(d): \quad T(n) = \begin{cases} 1 & \text{if } n < 2 \\ 2 \cdot T(\frac{n}{4}) + \sqrt{n} & \text{if } n \geq 2 \end{cases}$$

a = 2, b = 4, $f(n) = \sqrt{n}$
$n^{log_b(a)} = n^{log_4(2)} = n^{\frac{1}{2}} = \sqrt{n}$

Case 2 applies here
Case 2: If $\exists\ k \geq 0$ such that $f(n) \in \Theta(n^{log_b(a)} \cdot log^k n)$, then $T(n) \in \Theta(n^{log_b(a)} \cdot log^{k+1} n)$

$f(n) \in \Theta(n^{log_b(a)} \cdot log^k n) = n \in \Theta(n^{log_4(2)} \cdot log^0 n) = \sqrt{n} \in \Theta(\sqrt{n} \cdot 1) = \sqrt{n} \in \Theta(\sqrt{n})$, so case 2 holds.

Therefore $T(n) \in \Theta(n^{log_b(a)} \cdot log^{k+1} n) = T(n) \in \Theta(n^{log_4(2)} \cdot log^{0+1} n) = T(n) \in \Theta(\sqrt{n} \cdot log^1 n) = T(n) \in \Theta(\sqrt{n} \cdot logn)$

Hence $T(n) \in \Theta(\sqrt{n} \text{ log } n)$

5. (**10 pts**) The algorithm below computes nothing useful. It takes as a parameter an array A of integers. Note that $A.length$ returns the length of the array, and $A.sub(s, l)$ returns a new array (with elements copied) of length l with values copied from A starting at index s.

Algorithm 1 Wacky(A)

if $A.length == 1$ **then**
  $A[0] += 1;$
else
  int $m = \lfloor A.length/2 \rfloor;$
  $Wacky\ (A.sub(0, m));$
  $Wacky\ (A.sub(m, m));$
  $Wacky\ (A.sub(0, 1));$

(a) Write a recurrence relation that gives the running time of **Wacky**.

(b) Use the Master theorem to give a bound on the running time in terms of n.

$(a):$
$T(n) = a \cdot T(\frac{n}{b}) + f(n)$
a = 2 because there are 2 recursive calls with an array length greater than 1
b = 2
f(n) = 1 because the remainder of the function takes O(1) time

The recurrence relation for this algorithm: $T(n) = 2 \cdot T(\frac{n}{2}) + 1$

$(b):$
a = 2, b = 2, f(n) = 1
$n^{log_b(a)} = n^{log_2(2)} = n^1 = n$

Case 1 applies here
Case 1: If $\exists\ \epsilon > 0$ such that $f(n) \in n^{log_b(a)-\epsilon}$, then $T(n) = \Theta(n^{log_b(a)})$.

Since $f(n) \in n^{log_b(a)-\epsilon} = 1 \in n^{1-\epsilon} = 1 \in n^{1-1} = 1 \in 1$, so case 1 holds.

$T(n) \in \Theta(n^{log_b(a)}) = \Theta(n^{log_2(2)}) = \Theta(n)$

Hence $T(n) \in \Theta(n)$