Dennis Kuzminer
CSCI-UA 310-001 PS7

1.
    a. a = 2, b = 4, e = .5, f = $log_4 2$ = .5 → e = f → $T(n) = O(n^{.5} log(n))$
    b. a = 2, b = 4, e = .51, f = $log_4 2$ = .5 → e > f → $T(n) = O(n^{.51})$
    c. a = 3, b = 3, e = 1, f = $log_3 3$ = 1 → e = f → $T(n) = O(nlog(n))$
    d. a = 16, b = 4, e = 1.5, f = $log_4 16$ = 2 → e < f → $T(n) = O(n^2)$
    e. a = 7, b = 2, e = 2, f = $log_2 7$ = 2.80735 → e < f → $T(n) = O(n^{2.80735})$

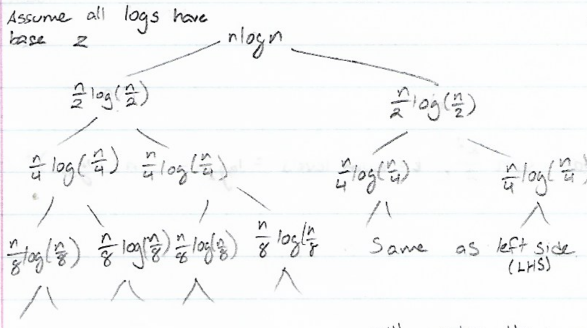
Dennis Kuzminer
CSCI-UA 310-001 PS7

2. We can take a binary search approach to this problem. We weigh ½ of the coins on one side and the other on the opposite side. Whichever is greater will be the ½ we want to consider, as this will be the pile of coins that contains the heavy coin. We can then only consider the heavier half. We can continually do this until we either have one coin in each division. This would clearly identify which coin is the heavier coin. However, if there is a case where we need to divide an odd number in half, we can divide the pile into almost half and remove one coin from the larger pile. If by chance, we pick the heavier coin, then the scale will balance and we can terminate our algorithm. On the other hand, if the scale does not balance, we can discard the coin we set aside, as we know that it is not the heavier coin and repeat the process assuming there is an equal number of coins on each side.
   Run time: For simplicity, assume that the initial pile contains 2^n coins. At each step, we are dividing the size of the comparisons by 2. In the worst case, we would find the coin at the last comparison. For example, with a pile of 16 coins, in the worst case, we compare 8 and 8 coins, 4 and 4, 2 and 2, 1 and 1, and then find the answer. This would be 4 comparisons, which would be $log_2(16) = 4$. More generally, we can find the solution in less than or equal to $log_2(2^n) \rightarrow log\ (n)$ comparisons. This is $O(log(n))$ minutes.
   Cost: Assume once more 16 coins. At each step, the cost is 8+8=16, 4+4=8, 2+2=4, and 1+1=2. Generalizing in terms of n, in total, this would be 1n+½n+¼n+⅛n. Summing we get $n(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}....)$. We can estimate the worst-case sum as $n(\frac{1}{1-\frac{1}{2}}) = 2n = O(n)$.

Dennis Kuzminer
CSCI-UA 310-001 PS7

3. Assume that all logs have a base of 2

** Assume all logs have base 2



$n \log n$
$\frac{n}{2}\log(\frac{n}{2})$
$\frac{n}{2}\log(\frac{n}{2})$
$\frac{n}{4}\log(\frac{n}{4})$
$\frac{n}{4}\log(\frac{n}{4})$
$\frac{n}{4}\log(\frac{n}{4})$
$\frac{n}{4}\log(\frac{n}{4})$
$\frac{n}{8}\log(\frac{n}{8})$
$\frac{n}{8}\log(\frac{n}{8})$
$\frac{n}{8}\log(\frac{n}{8})$
$\frac{n}{8}\log(\frac{n}{8}$
Same as left side (LHS)

Cost per level

$c_2 n \log n$

$c_2 n \log(\frac{n}{2})$

$c_2 n \log(\frac{n}{4})$

$c_2 n \log(\frac{n}{8})$

→ Total Cost: $\sum_{j=0}^{k-1} c_2 n \log(\frac{n}{2^j})$

↳ $k = \log_2 n$ = # levels

Multiply by number of levels

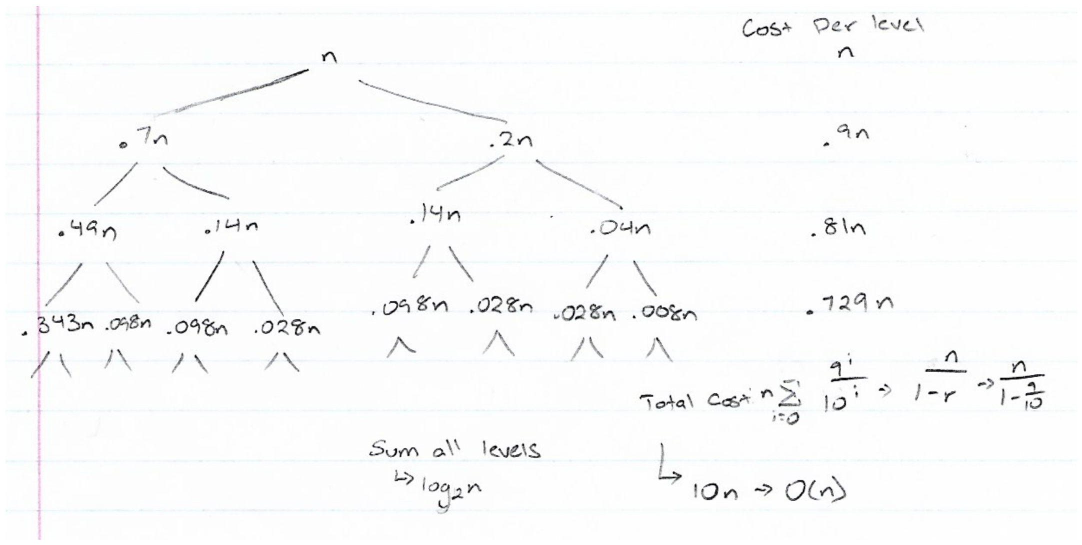
$\log n \cdot c_2 n \log(\frac{n}{2^j})$

↳ We can reason that the "2^j" term makes the inequality smaller; therefore, we can say that removing this term will maintain the inequality correctness, as we are making this RHS term greater. So, we can ~~b~~bound $T(n) \le d_2 n (\log_2 n)^2$. Next for the lower bound, assume n is a power of $2 = 2^k$.

$\sum_{j=0}^{k} c_1 n \log(\frac{2^k}{2^j}) \rightarrow \sum c_1 n (k-j) \rightarrow$ Rearrange $\rightarrow c_1 n \sum^{k} j \rightarrow c_1 n \left(\frac{k(k+1)^2}{2}\right) \rightarrow$ Using the same reasoning from the RHS, we can lower the LHS and still preserve correctness of the inequality. Therefore, for simplicity, we can lower the LHS to $c_1 n \frac{k^2}{2}$, k = num levels $= \log_2 n = c_1 n (\log_2 n)^2 / 2 \rightarrow$ This bounds $d_1 n (\log_2 n)^2 \le T(n)$

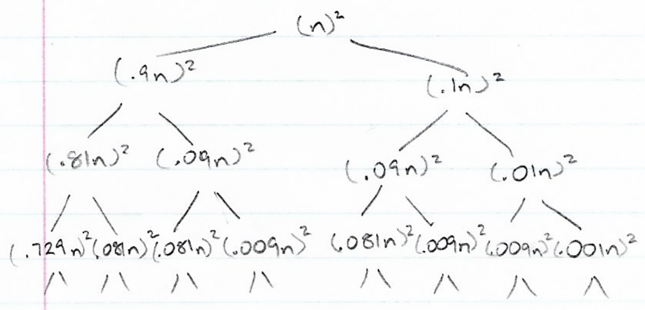
Dennis Kuzminer
CSCI-UA 310-001 PS7

4.



Dennis Kuzminer
CSCI-UA 310-001 PS7

5.



Cost per level

n^2

$.82n^2$

$.6724n^2$

$.551368n^2$

Total Cost: $n^2 \sum_{i=0}^{\infty} \left(\frac{82}{100}\right)^i \rightarrow \frac{n^2}{1-r} \rightarrow \frac{n^2}{1-\frac{82}{100}}$

$\hookrightarrow 5.556n^2 \rightarrow O(n^2)$

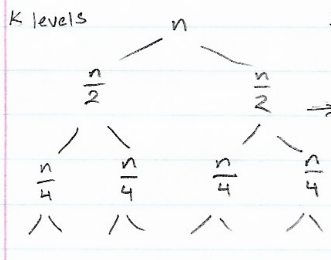
Sum all levels

$\hookrightarrow \log_2 n$

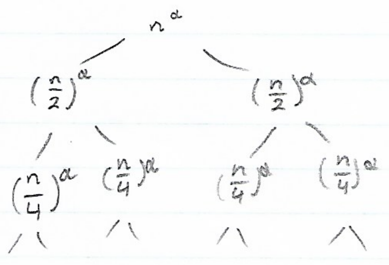
Dennis Kuzminer
CSCI-UA 310-001 PS7

6.

To take a divide and conquer approach, we can divide the problem into subproblems
Assume n is a power of 2 for simplicity. More specfically $n = 2^k$. Therefore, we can split the binomials $(x - a_i)$.



If we want to multiply we can "mul" the $\rightarrow$ subproblems from the $\rightarrow$ bottom (smallest subproblems) to the top $\rightarrow$ Time/subproblem $O(d^\alpha)$, $d = \frac{n}{2^j}$, $j \in [0..k)$



General trend in total cost per level: $2^j \cdot (\frac{n}{2})^\alpha$, $j \in [0, k)$ $\rightarrow$ Total cost $\sum_{j=0}^{k-1} 2^j (\frac{n}{2})^\alpha \rightarrow$ $\sum_1^1 (\frac{n^\alpha}{2^{j-\alpha}}) \rightarrow n^\alpha \sum_1^1 \frac{1}{2^{j-\alpha}}$ $\rightarrow$ This sumenation is a geometric power series that can be approximated by $\underset{\text{a constant,}}{\frac{1}{1-r}} \rightarrow \frac{1}{1-\frac{1}{2}} = 2$.

So the total cost using the "mul" function is $2n^\alpha$. We know that $1 < \alpha < 2$. Therefore, the total cost is $o(n^2)$