DAA Assignment

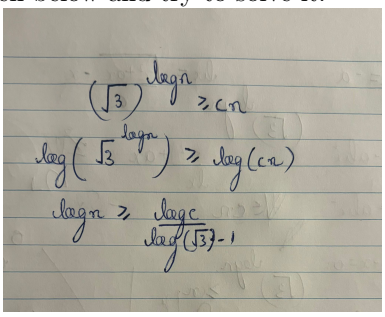
Aaryan Bhagat (abhag017)

January 2024

1 Part 1

1.1 Q1

Assume that for a certain positive rational number c we assume the inequality given below and try to solve it.



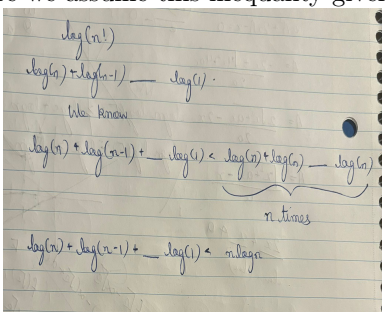
As we can see that this inequality is satisfiable for a given c and an arbitrarily large n.

Now if we reverse the inequality we can still see that for an arbitrarily large c the inequality will hold lets say till some large value of n = n1.

So we have a tight bound hence the symbol Θ.

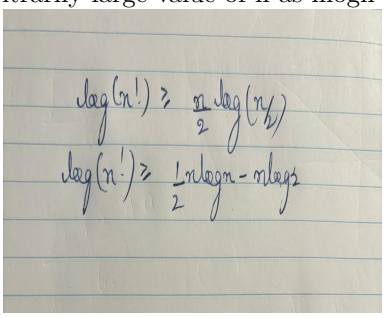
1.2 Q3

Here we assume this inequality given below



Then we understand that for any positive rational number c, RHS will always be greater for an arbitrarily large value of n, equality only possible at n=1 for a large value of c so we get big O.

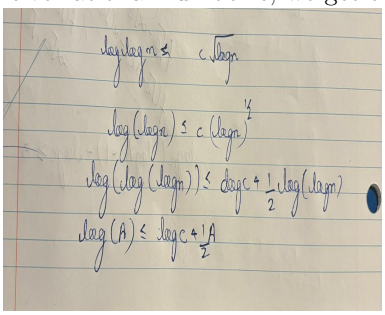
but we also have this inequality whch will exist for a small value of c and an arbitrarily large value of n as nlogn >> nlog2 so we get Ω



hence both inequalities exist hence we get Θ

1.3 Q2

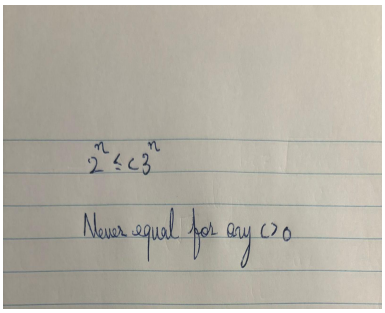
Assuming the inequality of the form given in the image and for an arbitrarily positive rational number c, we get the equation as described



Since we Assume A = log(log(n)), what we see is that if we assume log(c) to be 0 then after n = 16 LHS starts to rise slowly as compared to RHS, so for an arbitrarily big value of c, we will see that for a corresponding big value of n=n1 LHS will start rising above the RHS and before that it will be opposite. Hence a tight bound, hence the symbol Θ

1.4 Q4

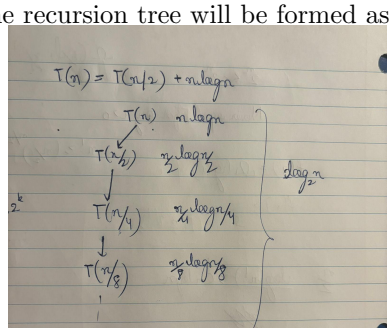
Here it is simple to see from the given image that for any positive rational number c, RHS is always greater than LHS for any n > 0. Hence small o.



2 Part 2

2.1 Q1

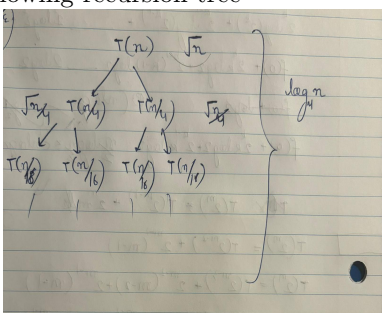
The recursion tree will be formed as follows



After solving the answer will be $\Theta(n * \log_2 n)$

2.2 Q2

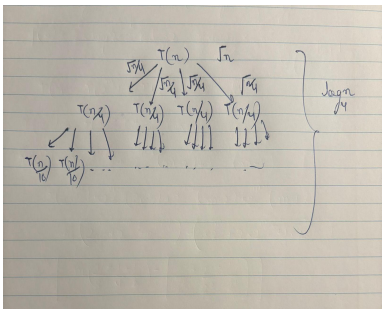
Following recursion tree



After solving the answer for this is $\Theta(\sqrt{n} \ln n)$

2.3 Q3

Tree



Answer after solving this is $\Theta(n)$

3 Part 3

3.1 Q1

If n is even

- Divide the pile into 2 equal halves and put on each side of the scale.
- The side which is less haevy, take that pile, ignore the other pile altogether, divide that pile into 2 equal halves and again put on both sides of the scale.
- Repeat this process until completion.
- Total $\log_2 n$ steps.

If n is odd

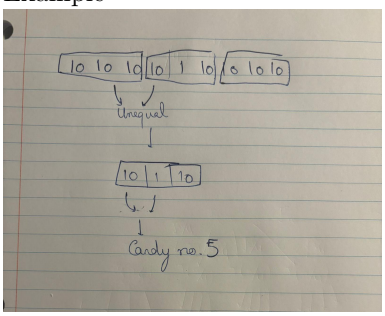
- Take a random candy out of the pile, put it aside.
- Do the above process on the remaining pile.
- If you find the bad candy in the pile in $\log_2(n-1)$ steps then good.
- If you do not then the candy you took out is the bad candy.

So in total $\lceil \log_2 n \rceil steps$.

3.2 Q2

- Divide n into 3 piles
- Weight any 2 piles if they are equal then the third pile has the bad candy. If they are not equal then the less weight pile has the bad candy.
- Choose the less weight pile, again divide into 3 parts and repeat.
- Base case is when n=3.

Will result in $\lceil \log_3 n \rceil steps.$
Example



3.3 Q3

The algorithms we are devising are coparison based so a proof using information theoretic approach might help out here.

For given n candies and 1 bad candy, there are n unique permutations (bad candy as index 1-n), we can divide into 2 parts and compare, or we can divide into 3 and compare any 2 of the 3.

We cannot divide into more than 3 as then we will need more than 1 comparison per division.

And just like sorting we need to distinguish all possible n permutations. So a tree is formed having 3 branches per node.

Hence we get total of $\lceil \log_3 n \rceil$
comparisons as lowest possible.

3.4 Q4

If n is even:

- Divide the pile into 2 equal halves and put on the balance.
- If equal then each bad candy is in one pile. Follow solution to q2 for each pile.
- If one pile is less than the other pile, ignore the heavier pile, then follow step 1.
- One more base case happens when we have n=3 left, those 2 candies having the same weight they are the bad candies.

If n is odd:

- Same as above
- Only change is when you get the base case of n=3, compare any 2, if one is less, then compare the bigger one with the 3rd one. If they are equal then the less candy and the random candy you took out are the bad candies.

- If they are not equal then the 2 less candies are the bad candies.
- If both are equal then compare one of them with the candy you took out in the start. If it is less then the 2 candies who are equal are the bad candies.
- If the random candy is less than the random candy and the 3rd candy are the bad candies.

Both cases time complexity is O ($\log_2 n$)

3.5 Q5

3.6 Q6

Here the permutations remain the same so the order of division also remains the same.

However in each level if we have 3 piles, there are 2 cases

- Compare 2 piles and they are equal then the bad candy is in the third pile.
- Compare 2 piles and they are not equal then take 1st pile and compare with the 3rd pile, if equal then the 2nd pile has bad candy, if not equal then the 1st pile has bad candy.
- Base case when you got 3 candies, compare 2, if equal then 3rd candy is bad candy, compare to find if its heavier or lighter
- If they are not equal then similar as step 2 but it is singular candy instead of pile.

For minimum number of comparisons as we see each layer has 2 comparisons in worst case. Solving it will result in $\lceil \log_3 2n \rceil$

4 Part 4

Applied newton's method, used a starting point of x just greater than 1. The behaviour of the function similar to that of a U shaped curve so this method is guaranteed to find an answer.

The step size and iterations I found after a number of hit and trial.

Cost of this algorithm is O(iterations)

Submission ID 242493273

5 Part 5

Take example of an array [10, 8, 9, 7, 3, 5]

We see that 8 will need 3 swaps as 7, 3, 5 are less than 8 but are ahead of it.

Similarly for 10 it will be 5 swaps and so on

Immediately we get the idea of using some modified form of merge sort.

Take case of this array inside the merge sort algorithm with the following state

Assume totalswaps = 0

A = [8, 9, 10] B = [3, 5, 7]

Assume number of swaps for A = a

Similarly number of swaps for B = b

Now when we will merge these array, we have to calculate the swaps obtained after merging.

Take count = 0

3 < 8 (count++) [3]

5 < 8 (count++) [3, 5]

7 < 8 (count++) [3, 5, 7]

8 added (totalswaps = totalswaps + count) [3, 5, 7, 8] (totalswaps = 3)

9 added (totalswaps = totalswaps + count) (because 9 also has to be swapped with 3, 5, 7 in the original array)

....

We get totalswaps = 9

now add swaps of A (a) and swaps of (b) to get total swaps = 9 + a + b

Cost of this algorithm is same as mergesort.

Submission Id 242489913