CS 170 Homework 1

Due **1/31/2022, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

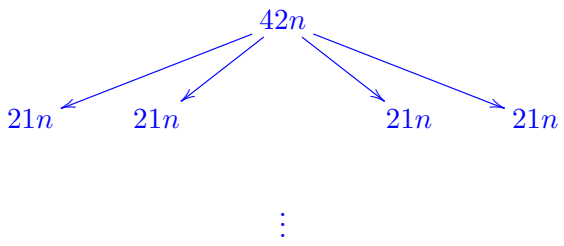
In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 Recurrence Relations

For each part, find the asymptotic order of growth of T; that is, find a function g such that $T(n) = \Theta(g(n))$. In all subparts, you may ignore any issues arising from whether a number is an integer.

(a) $T(n) = 4T(n/2) + 42n$

Solution: Use the master theorem. Or:



The first level sums to $42n$, the second sums to $84n$, etc. The last row dominates, and we have $\log n$ rows, so we have $42 \cdot 2^{\log n} \cdot n = \Theta(n^2)$.

(b) $T(n) = 4T(n/3) + n^2$

Solution: Use the master theorem (the case $d > \log_b a = \log_3 4$), or expand like the previous question. The answer is $\Theta(n^2)$.

(c) $T(n) = T(3n/5) + T(4n/5)$ (We have $T(1) = 1$)

Solution: The Master theorem does not directly apply to this problem. We will try a different approach where we will first attempt to guess a solution. Firstly, note that we have $T(n) \geq 2T(n/2)$. Therefore, we have $T(n) \geq O(n)$. By a similar reasoning, we have $T(n) \leq 2T(4n/5)$. This gives $T(n) \leq O\left(n^{\log_{5/4} 2}\right)$. From these two inferences, we guess that $T(n)$ is probably of the form an^b for some values of a and b. To determine the value of a, we have $T(1) = 1 = a$. Therefore, we have $T(n) = n^b$. Now, to determine the value of b, we use the recurrence relation:

$$T(n) = n^b = T(3n/5) + T(4n/5) = \left(\frac{3}{5}\right)^b n^b + \left(\frac{4}{5}\right)^b n^b$$

The above equation is uniquely satisfied for $b > 0$ at $b = 2$ as the function $f(b) = (3/5)^b + (4/5)^b$ is decreasing in b. Therefore, the solution to the above recurrence is $T(n) = n^2$. Therefore, the answer to the $\Theta(n^2)$.

3 Computing Factorials

Consider the problem of computing $N! = 1 \times 2 \times \cdots \times N$.

(a) N is $\log N$ bits long (this is how many bits are needed to store a number the size of N). Find an $f(N)$ so that $N!$ is $\Theta(f(N))$ bits long. Simplify your answer as much as possible, and give an argument for why it is true.

Note: You may not use Stirling's formula.

Solution: When we multiply an m bit number by an n bit number, we get an $(m+n)$ bit number. When computing factorials, we multiply N numbers that are at most $\log N$ bits long, so the final number has at most $N \log N$ bits.

But if you consider the numbers from $\frac{N}{2}$ to N, we multiply at least $\frac{N}{2}$ numbers that are at least $\log N - 1$ bits long, so the resulting number has at least $\frac{N(\log N - 1)}{2}$ bits.

This means that our number has $\Theta(N \log N)$ bits.

(b) Give a simple (naive) algorithm to compute $N!$. Use $\Theta(mn)$ as the runtime for multiplying an m-bit number and an n-bit number.

For this problem, you don't need to write a proof of correctness (that is, just state your algorithm and analyze its runtime).

Solution: We can compute $N!$ naively as follows:

```
factorial (N)
    f = 1
    for i = 2 to N
        f = f · i
```

Running time : $\Theta(N^2 \cdot \log^2 N)$.

We have N iterations, each one multiplying an $N \cdot \log N$-bit number (at most) by an $\log N$-bit number. Using the naive multiplication algorithm, each multiplication takes time $O(N \cdot \log^2 N)$. Hence, the running time is $O(N^2 \log^2 N)$.

A lower bound of $\Omega(N^2 \cdot \log^2 N)$ follows because the product of the first $N/2$ numbers will be at least $\Omega(N \log N)$ bits long by the previous part, and the last $N/2$ multiplications take $\Omega(N \log^2 N)$ time each.

4 Decimal to Binary

Given the n-digit decimal representation of a number, converting it into binary in the natural way takes $O(n^2)$ steps. Give a divide and conquer algorithm to do the conversion and show that it does not take much more time than Karatsuba's algorithm for integer multiplication.

Just state the main idea behind your algorithm and its runtime analysis; no proof of correctness is needed as long as your main idea is clear.

Solution: Similar to Karatsuba's algorithm, we can write x as $10^{n/2} \cdot a + b$ for two $n/2$-digit numbers a, b. The algorithm is to recursively compute the binary representations of $10^{n/2}, a$, and b. We can then compute the binary representation of x using one multiplication and one addition.

The multiplication takes $O(n^{\log_2(3)})$ time, the addition takes $O(n)$ time. So the recurrence we get is $T(n) = 3T(n/2) + O(n^{\log_2(3)})$. This has solution $O(n^{\log_2(3)} \log n)$.

(There is an $O(n^{\log_2(3)})$-time algorithm: We can compute the binary representation of 10^n in time $O(n^{\log_2(3)})$ by doing the multiplications $10 * 10, 10^2 * 10^2, 10^4 * 10^4 \ldots$ - note that the multiplication of $10^{n/2} * 10^{n/2}$ dominates the total runtime of these multiplications. This gives the recurrence $T(n) = 2T(n/2) + O(n^{\log_2(3)})$ instead, with solution $T(n) = O(n^{\log_2(3)})$.)

5 Pareto Optimality

(a) Given a set of points $P = \{(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)\}$, a point $(x_i, y_i) \in P$ is Pareto-optimal if there does not exist any $j \neq i$ such that such that $x_j > x_i$ and $y_j > y_i$. In other words, there is no point in P above and to the right of (x_i, y_i). Design a $O(n \log n)$-time divide-and-conquer algorithm that given P, outputs all Pareto-optimal points in P. **Give a 3-part solution**: Algorithm description, proof of correctness, and runtime analysis.

(Hint: Split the array by x-coordinate. Show that all points returned by one of the two recursive calls is Pareto-optimal, and that you can get rid of all non-Pareto-optimal points in the other recursive call in linear time).

(b) Code your solution! Go to https://judge.cs170.org, sign in, go to the coding problem corresponding to the algorithm question above, and follow the directions. It is highly recommended that you design the whole algorithm first and convince yourself it is correct before writing any code.

Please report any technical issues on Piazza.

Solution: Let L be the left half of the points when sorted by x-coordinate, and R be the right half. Recurse on L and R, let L', R' be the sets of Pareto-optimal points returned. Every point in R' is Pareto-optimal, since all points in L have smaller x-coordinates and can't violate Pareto-optimality of points in R'. For each point in L', it's Pareto-optimal iff its y-coordinate is larger than y_{max}, the largest y-coordinate in R. We can compute y_{max} in a linear scan, and then remove all points in L' with a smaller y-coordinate. We then return the union of L', R'.

This runs in $T(n) = 2T(n/2) + O(n) = O(n \log n)$ time.