CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2024**

Homework #1

Name(s): $FIRST_1$ $LAST_1$, $FIRST_2$ $LAST_2$, $FIRST_3$ $LAST_3$

Homework Policy

- If you leave a question completely blank, you will receive 20% of the grade for that question. This however does not apply to the extra credit questions.
- You may also consult all the materials used in this course (lecture notes, textbook, slides, etc.) while writing your solution, but no other resources are allowed.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "sort the array in $\Theta(n \log n)$ time using merge sort".
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question). See "Practice Homework" for an example.
- The extra credit problems are generally more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.
- **Groups:** You are allowed to form groups of size *two* or *three* students for solving each homework (you can also opt to do it alone if you prefer). The policy regarding groups is as follows:
  - You can pick different partners for different assignments (e.g., from HW1 to HW2) but for any single assignment (e.g., HW1), you have to use the same partners for all questions.
  - The members of each group only need to write down and submit a single assignment between them, and all of them will receive the same grade.
  - For submissions, only one member of the group submits the full solutions on Canvas and lists the name of their partners in the group. The other members of the group also need to submit a PDF on Canvas that contains only a single line, stating the name of their partner who has submitted the full solution on Canvas (Example: Say A, B, and C are in one group; A submits the whole assignment and writes down the names A, B, and C. B and C only submit a one-page PDF with a single line that says "See the solution of A").
  - You are allowed to discuss the questions with any of your classmates even if they are not in your group. **But each group must write their solutions independently.**

Problem 1. This question reviews asymptotic notation. You may assume the following inequalities for this question (and throughout the course): For any constant $c \geq 1$,

$$(\log n)^c = o(n) \quad , \quad n^c = o(n^{c+1}) \quad , \quad c^n = o((c+1)^n) \quad , \quad (n/2)^{(n/2)} = o(n!).$$

(a) Rank the following functions based on their asymptotic value in the increasing order, i.e., list them as functions $f_1, f_2, f_3, \ldots, f_9$ such that $f_1 = O(f_2)$, $f_2 = O(f_3), \ldots, f_8 = O(f_9)$. Remember to write down your proof for each equation $f_i = O(f_{i+1})$ in the sequence above. **(15 points)**

$\sqrt{n}$	$\log n$	$n^{\log n}$
$100n$	2^n	$n!$
9^n	$3^{3^{3^3}}$	$\dfrac{n}{\log^2 n}$

Hint: For some of the proofs, you can simply show that $f_i(n) \leq f_{i+1}(n)$ for all sufficiently large n which immediately implies $f_i = O(f_{i+1})$.

(b) Consider the following four different functions $f(n)$:

$$1 \qquad \log\log n \qquad n^2 \qquad 2^{2^n}.$$

For each of these functions, determine which of the following statements is true and which one is false. Remember to write down your proof for each choice. **(10 points)**

- $f(n) = \Theta(f(n-1))$;
- $f(n) = \Theta(f(\frac{n}{2}))$;
- $f(n) = \Theta(f(\sqrt{n}))$;

Example: For the function $f(n) = 2^{2^n}$, we have $f(n-1) = 2^{2^{n-1}}$. Since $2^{2^{n-1}} = 2^{\frac{1}{2}\cdot 2^n} = (2^{2^n})^{1/2}$.

$$\lim_{n\to\infty} \frac{f(n)}{f(n-1)} = \lim_{n\to\infty} \frac{2^{2^n}}{2^{2^{n-1}}} = \lim_{n\to\infty} \frac{2^{2^n}}{(2^{2^n})^{1/2}} = \lim_{n\to\infty} (2^{2^n})^{1/2} = +\infty.$$

As such, $f(n) \neq O(f(n-1))$ and thus the first statement is false for 2^{2^n}.

(b) For each of the following functions, state whether $f(n) = O(g(n))$ or $f = \Omega(g(n))$, or if both are true, then write $f = \Theta(g(n))$. No proofs required for this problem. **(10 points)**

(a) $f(n) = n^2 - 7n$ and $g(n) = n^3 - 10n^2$
(b) $f(n) = (\sqrt{n})^3$ and $g(n) = n^2 - (\sqrt{n})^3$
(c) $f(n) = n^{log_3(4)}$ and $g(n) = n\log^3(n)$
(d) $f(n) = 2^{\log_2(n)}$ and $g(n) = n$
(e) $f(n) = log^5(n)$ and $g(n) = n/\log(n)$
(f) $f(n) = 4^n$ and $g(n) = 5^n$
(g) $f(n) = \log_4(n)$ and $g(n) = \log_5(n)$
(h) $f(n) = n^3$ and $g(n) = 2^n$
(i) $f(n) = \sqrt{n}$ and $g(n) = \log^3(n)$.
(j) $f(n) = n\log(n)$ and $g(n) = n^2$

Problem 2. Your goal in this problem is to analyze the *runtime* of the following (imaginary) recursive algorithms for some (even more imaginary) problem:

(A) Algorithm A divides an instance of size n into 5 subproblems of size $n/5$ each, recursively solves each one, and then takes $O(n)$ time to combine the solutions and output the answer.

(B) Algorithm B divides an instance of size n into 5 subproblems of size $n/2$ each, recursively solves each one, and then takes $O(n^2)$ time to combine the solutions and output the answer.

(C) Algorithm C divides an instance of size n into 5 subproblems of size $n/2$ each, recursively solves each one, and then takes $O(n^{0.8})$ time to combine the solutions and output the answer.

(D) D divides an instance of size n into 2 subproblems, one with size $n/4$ and one with size $n/5$, recursively solves each one, and then takes $O(n)$ time to combine the solutions and output the answer.

(E) Algorithm E divides an instance of size n in to 3 subproblems of size $n-1$ each, recursively solves each one, and then takes $O(1)$ time to combine the solutions and output the answer.

For each algorithm, write a recurrence for its runtime and *use the recursion tree method* to solve this recurrence and find the *tightest asymptotic* upper bound on runtime of the algorithm. **(25 points)**

Problem 3. Each of the algorithms below takes as input a positive integer n and then performs some steps. For each algorithm, state the running time of the algorithm as a function of n, using Θ-notation.
(15 points)

a)
- For $i = 1$ to $n/3$
  - $j \leftarrow 10$
  - While ($j \leq i/20$)
    - Do placeholder stuff that takes $O(1)$ time
    - $j \leftarrow j + 10$

b)
- While ($n \geq 1$)
  - $n \leftarrow n/3$
  - do placeholder stuff that takes $O(1)$ time.

c)
- While ($n \geq 2$)
  - $n \leftarrow \sqrt{n}$
  - Do placeholder stuff that takes $O(1)$ time.

d)
- For $i = 1$ to n
  - For $j = 1$ to i^2
    - Do placeholder stuff that takes $O(1)$ time

e)
- For $i = 1$ to n
  - For $j = 1$ to $\log(i)$
    - For $k = 1$ to n
      - Do placeholder stuff that takes $O(1)$ time

Problem 4. In this problem, we consider a non-standard sorting algorithm called the *Slow Sort*. Given an array $A[1:n]$ of n integers, the algorithm is as follows:

• **Slow-Sort**$(A[1:n])$:

1. If $n < 100$, run merge sort (or selection sort or insertion sort) on A.
2. Otherwise, run **Slow-Sort**$(A[1:n-1])$, **Slow-Sort**$(A[2:n])$, and **Slow-Sort**$(A[1:n-1])$ again.

We now analyze this algorithm.

(a) Prove the correctness of **Slow-Sort**. **(15 points)**

(b) Write a recurrence for **Slow-Sort** and use the recursion tree method to solve this recurrence and find the *tightest asymptotic* upper bound on the runtime of **Slow-Sort**. **(10 points)**

Problem 5. Karatsuba multiplication reduces the time complexity of multiplying two n-digit numbers to $O(n^{\log_2 3}) \approx O(n^{1.585})$, using the recurrence:

$$T(n) = 3T(n/2) + O(n)$$

The algorithm splits the numbers as follows:

$$x = x_1 \cdot 10^{n/2} + x_0, \quad y = y_1 \cdot 10^{n/2} + y_0$$

and computes three products: $x_0 \cdot y_0$, $x_1 \cdot y_1$, and $(x_0 + x_1) \cdot (y_0 + y_1)$.

Two values you are multiplying might be n/2 + 1 digits each, rather than n/2 digits, since the addition might have led to a carry. For instance compute $(x_1 + x_0)(y_1 + y_0)$ for $x = 53$ and $y = 52$. Notice the carry and the extra digit. So the recurrence will be

$$T(n) = 2T(n/2) + T(n/2 + 1) + O(n)$$

Prove that this issue won't affect our analysis and we still $O(n^{\log_2 3})$ complexity. **(20 points)**

Problem 6. Toom-Cook multiplication generalizes Karatsuba by dividing numbers into more parts. The Toom-3 algorithm splits two n-digit numbers x and y into three smaller parts:

$$x = x_2 \cdot 10^{2k} + x_1 \cdot 10^k + x_0, \quad y = y_2 \cdot 10^{2k} + y_1 \cdot 10^k + y_0$$

If we multiply we need to perform 9 multiplications $x_i y_j$ of size $n/3$.

1. Write down the recurrence relation for the time complexity of the simple divide and conquer algorithm and solve it.
2. Toom-3 algorithm instead only uses 5 smaller multiplications to calculate all the 9 products needed. Find the time complexity of Toom-3.

(20 points)

Problem 7. Merge Sort usually splits an array into two equal halves. However, consider a version where the array is split into parts of size $2n/3$ and $n/3$.

a) Write the recurrence relation for the time complexity of Merge Sort with this unequal split.

b) Solve the recurrence and find the time complexity.

(20 points)

Problem 8. Consdier the problem of computing the Fibonacci numbers:

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$$

a) Write an algorithm to compute the Fibonacci number F_n with $O(n)$ additions.

Note that F_n grows exponentially and has $\Theta(n)$ digits which means that the additions operation $F_n + F_{n+1}$ actually takes $O(n)$ time. Compute the time complexity of the algorithm.

b) We start by writing the equations $F_1 = F_1$ and $F_2 = F_0 + F_1$ in matrix notation:

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Similarly,

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

In general,

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

So, in order to compute F_n, it suffices to raise this 2×2 matrix, call it X, to the nth power.

i) Show that $O(\log n)$ matrix multiplications suffice for computing X^n by repeated squaring. Note that two 2×2 matrices can be multiplied using 4 additions and 8 multiplications.

ii) Let $M(n)$ be the running time of an algorithm for multiplying n-bit numbers. Prove that the running time of this second algorithm is $O(M(n) \log n)$.

(20 points)

Challenge Yourself. We have an n-story building and a series of magical vases that work as follows: there is some unknown level L in the building that if we throw these vases down from any of the levels $L, L+1, \ldots, n$, they will definitely break; however, no matter how many times we throw the vases down from any level below L nothing will happen them. Our goal in this question is to determine this level L by throwing the vases from different levels of the building (!).

For each of the scenarios below, design an algorithm that uses asymptotically the smallest number of times we throw a vase (so the measure of efficiency for us is the number of vase throws).

(a) When we have only one vase. Once we break the vase, there is nothing else we can do. **(+2 points)**

(b) When we have four vases. Once we break all four vases, there is nothing else we can do. **(+4 points)**

(c) When we have an unlimited number of vases. **(+4 points)**