CS-310, Fall 2021
Assignment 3
Assigned: Nov. 3, Due: Monday, Nov. 15, 11:55PM

November 2, 2021

Total Points = 195

Problem 1. (Points: 15) Suppose you are choosing between the following 3 algorithms:

- Algorithm A solves the problem of size n by dividing it into 8 subproblems of size $n/4$, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves the problem of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time
- Algorithm C solves the problem of size n by dividing it into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each algorithm and which would you choose and why?

Problem 2. (Points: 10) Assume that $n = 2^k$ for some positive integer k. Using induction prove that if $T(n)$ is given as follows, then $T(n) = n \log n$.

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T\left(\frac{n}{2}\right) + n & \text{if } n > 2 \end{cases}$$

Problem 3. (Points: 20) Assume you have an array $A[1..n]$ of n elements. A majority element of A is any element occurring in more than $n/2$ positions (so if $n = 6$ or $n = 7$, any majority element will occur in at least 4 positions). Assume that elements cannot be ordered or sorted, but can be compared for equality. (You might think of the elements as chips, and there is a tester that can be used to determine whether or not two chips are identical.) Design an efficient divide and conquer algorithm to find a majority element in A (or determine that no majority element exists). Aim for an algorithm that does $O(n \log n)$ equality comparisons between the elements. A more difficult $O(n)$ algorithm is possible, but may be difficult to find.

Problem 4. (Points: 20) Given a binary string S of type$\{1^m0^n\}$, devise an algorithm that finds the number of zeroes in $O(\log k)$ time. Let $m + n = k$

Problem 5. (Points: 20) **[K-way Merge]** Suppose you have k sorted arrays $A_1, A_2, ...A_k$ each with n elements. You want to combine them into a single sorted array of size kn. One way to do this would be to use the merge operation we discussed in class. First merge arrays A_1, A_2 then merge the result with A_3 and so on

- Figure out how many steps this algorithm would take.
- Design a better algorithm for this problem.

Problem 6. (Points: 20) **[Fibonacci Numbers]**
Given below is a recursive algorithm for finding the n_{th} Fibonacci number

---

Algorithm 1 :

```
function FIB(n)
    if n == 0 or n == 1 then
        return 1
    else
        return FIB(n-1) + FIB(n-2)
```

- Analyze the running time of this algorithm
- A lot of computation seems to be repeated over and over in this algorithm. For example $fib(4)$ would compute $fib(3)$ and $fib(2)$ and then the called $fib(3)$ would also compute $fib(2)$. Is there a way we could avoid computing the same thing over and over again while still using a recursive approach?

Problem 7. (Points: 30) Given the following recurrence relations, find the running time (in big-O notation) using the recursion tree method and the Master Theorem.

- $$T(n) = \begin{cases} 8T\left(\dfrac{n}{2}\right) + \Theta(n^2) & \text{if } n \geq 2 \\ 1 & \text{else} \end{cases}$$
- $$T(n) = \begin{cases} 3T\left(\dfrac{n}{4}\right) + n\log n & \text{if } n \geq 2 \\ 1 & \text{else} \end{cases}$$
- $$T(n) = \begin{cases} 2T\left(\dfrac{n}{4}\right) + \sqrt{n} & \text{if } n \geq 2 \\ 1 & \text{else} \end{cases}$$

Problem 8. (Points: 20)

A bank confiscates n bank cards suspecting them to be involved in fraud. Each card corresponds to a unique account but an account can have many cards corresponding to it. Therefore, two bank cards are equivalent if they belong to the same account. The bank has a testing machine for finding out if two cards are equivalent.
They want to know that among the collection of n cards, are there more than $n/2$ cards that are equivalent. The only operation that the machine can do is to select two cards and tests them for equivalence. You are required to come up with a solution to this using only $O(n \log n)$ invocations of the testing machine.

Problem 9. (Points: 20) Prove the correctness of the following algorithm for incrementing natural numbers. Analyze the algorithm to find out how many times is line 3 executed in the worst case.

Algorithm 2 : Increment Natural Numbers

function INCREMENT(x) ▷ Returns $x + 1$
  if $x \mod 2 == 1$ **then**
    return $2 \times$ INCREMENT($\lfloor \frac{x}{2} \rfloor$)
  else
    return $x + 1$

Problem 10. (Points: 20) Prove the correctness of the following recursive algorithm for the multiplication of two natural numbers is correct, $\forall$ integer constants ≥ 2. Analyze this algorithm to find out how many times line 5 executed in the worst case.

Algorithm 3 : Multiply Two Numbers

function MULTIPLY(x,y) ▷ Returns product xy
  if $y == 0$ **then**
    return 0
  else
    return MULTIPLY(cx, $\lfloor y/c \rfloor$) $+ x(y \mod c)$