Homework #2
CMSC351H - Spring 2019

PRINT Name:__________________________:

- Grades depend on neatness and clarity.
- Write your answers with enough detail about your approach and concepts used, so that the grader will be able to understand it easily. You should ALWAYS prove the correctness of your algorithms either directly or by referring to a proof in the book.
- Write your answers in the spaces provided. If needed, attach other pages.
- The grades would be out of 16. Four problems would be selected and everyone’s grade would be based only on those problems. You will also get 4 bonus points for trying to solve all problems.

1. [Prob 3.4,Pg 56] By using Theorem 3.1 in the book (similar theorem is mentioned in the class), prove that \((\log_2 n)^{100} = O\left(\frac{1}{n^{10}}\right)\).
2. Prove that $T(n) = O(n \log n)$, where $T(n)$ is defined by the following recurrence relation ($c$ is some constant)

$$T(n) = \begin{cases} 
c & \text{if } n = 1, \\
10T\left(\left\lfloor \frac{n}{10} \right\rfloor \right) + cn & \text{if } n > 1.
\end{cases}$$
3. [Prob 3.8, Pg 56] Prove that $T(n)$, which is defined by the recurrence relation

$$T(n) = \begin{cases} 
4 & \text{if } n = 2, \\
2T\left(\left\lceil \frac{n}{2} \right\rceil \right) + 2n \log_2 n & \text{if } n > 2.
\end{cases}$$

satisfies $T(n) = O(n \log^2 n)$. 
4. [Prob 3.22-a,Pg 58] Solve the following recurrence relation. It is sufficient to find the asymptotic behavior of \( T(n) \). (Hint: Substitute another variable for \( n \))

\[
T(n) = \begin{cases} 
1 & \text{if } n = 2, \\
4T(\lfloor \sqrt{n} \rfloor) + 1 & \text{if } n > 2.
\end{cases}
\]
5. The sieve of Eratosthenes\(^1\) is an algorithm for finding all the prime numbers \(p\) such that \(p \leq n\). The algorithm starts with marking every number \(i \leq n\) except 1 as prime numbers. Next, it iterates on \(i\) from 2 to \(\sqrt{n}\) and if \(i\) is still marked as a prime number, marks every number which \(i\) divides as a not prime number. Since for each composite number \(c \leq n\), there is a prime number \(p \leq \sqrt{n}\) such that \(p < c\) and \(p\) divides \(c\), number \(c\) is marked as not prime at some step before the for loop reaches number \(c\). This way, all numbers marked as prime numbers at the end of the algorithm are the prime numbers from 2 to \(n\). The running time of the algorithm equals

\[
T(n) = \sum_{p \leq \sqrt{n}} O\left(\frac{n}{p}\right)
\]

a) The running time of the algorithm, \(T(n)\), is upper-bounded by \(\sum_{i=1}^{n} O\left(\frac{n}{i}\right)\). Prove that \(T(n) = O(n \log n)\) by showing that \(\sum_{i=1}^{n} O\left(\frac{n}{i}\right) = O(n \log n)\).

b) Use the following two facts to prove \(T(n) = O(n \log \log n)\).

- Let \(\pi(n)\) be the number of primes numbers \(p \leq n\). Then, \(\pi(n) = \theta\left(\frac{n}{\log n}\right)\) holds for all \(1 < n\).
- \(\sum_{i=3}^{n} \frac{1}{\log i} = O(\log \log n)\). Why?

6. [Prob 3.12, Pg 57] Solve the following full-history recurrence relation:

\[ T(n) = \begin{cases} 
1 & \text{if } n = 1, \\
n + \sum_{i=1}^{n-1} T(i) & \text{if } n > 1.
\end{cases} \]
7. [Prob 5.22, Pg 116] Towers of Hanoi: There are \( n \) disks of different sizes arranged on a peg in decreasing order of sizes. There are two other empty pegs. The purpose of the puzzle is to move all the disks, one at a time, from the first peg to another peg in the following way. Disks are moved from the top of one peg to the top of another. A disk can be moved to a peg only if it is smaller than all other disks on that peg. In other words, the ordering of disks by decreasing sizes must be preserved at all times. The goal is to move all the disks in as few moves as possible.

a) Design an algorithm (by induction) to find a minimal sequence of moves that solves the towers of Hanoi problem for \( n \) disks.

b) How many moves are used in your algorithm? Construct a recurrence relation for the number of moves, and solve it.

c) Prove that the number of moves in part b is optimal; that is, prove that no algorithm can use fewer moves (use induction).
8. Finding the majority: We have \( n \) numbers such that one number has appeared at least \( \left\lfloor \frac{n}{2} \right\rfloor + 1 \) times. Design an algorithm that finds this number by at most \( n \) comparisons between given numbers. (Hint: Use a similar idea as in Celebrity Problem).
9. A *corrupted linked* list is a linked list in which the pointer of its last element points to another element in the list instead of being NIL. Given a pointer to the first element of a linked list of size $n$, design an algorithm with $O(1)$ space and $O(n)$ running time which prints `"YES"` if the given linked list is corrupted and prints `"NO"` otherwise. Note that you cannot change the data in the linked list.
10. Given a sequence of (not necessary positive) integers \(x_1, x_2, \ldots, x_n\), find a subsequence \(x_i, x_{i+1}, \ldots, x_j\) (of consecutive elements) such that the sum of the numbers in it is **even and maximum** over all subsequences of consecutive elements with an even sum of elements. For example if the sequence is 5, -1, 4, -10, 3, 3, 3, the maximum even consecutive subsequence is 5, -1, 4. The running time of your algorithm should be in \(O(n)\).