MATH182 HOMEWORK #2
DUE July 5, 2020

Note: while you are encouraged to work together on these problems with your classmates, your final work should be written in your own words and not copied verbatim from somewhere else. You need to do at least six (6) of these problems. All problems will be graded, although the score for the homework will be capped at $N :=$ (point value of one problem) $\times 6$ and the homework will be counted out of N total points. Thus doing more problems can only help your homework score. For the programming exercise you should submit the final answer (a number) *and* your program source code.

Exercise 1. *Consider the following basic problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional $n \times n$ array B in which $B[i,j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$, that is, the sum $A[i] + A[i+1] + \cdots + A[j]$, (The value of array entry $B[i,j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.)*

Here is a simple algorithm to solve this problem.

```
1  for $i = 1$ to $n$
2      for $j = i+1$ to $n$
3          Add up array entries $A[i]$ through $A[j]$
4          Store the result in $B[i,j]$
```

(1) *For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).*

(2) *For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)*

(3) *Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem – after all, it just iterates through the relevant entries of the array B, filling in a value for each – it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.*

Exercise 2. *Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.*

BUBBLESORT(A)

```
1  for $i = 1$ to $A.length - 1$
2      for $j = A.length$ downto $i + 1$
3          if $A[j] < A[j-1]$
4              exchange $A[j]$ with $A[j-1]$
```

(1) *Let A' denote the output of* BUBBLESORT(A). *To prove that* BUBBLESORT *is correct, we need to prove that it terminates and that*

$$(\dagger) \qquad A'[1] \leq A'[2] \leq \cdots \leq A'[n],$$

where $n = A.length$. In order to show that BUBBLESORT *actually sorts, what else do we need to prove?*

The next two parts will prove inequality (†).

(2) *State precisely a loop invariant for the* ***for*** *loop in lines 2-4, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter.*

(3) *Using the termination condition of the loop invariant proved in part (2), state a loop invariant for the* ***for*** *loop in lines 1-4 that will allow you to prove inequality (†). Your proof should use the structure of the loop invariant proof presented in this chapter.*

(4) *What is the worst-case running time of bubblesort? How does it compare to the running time of insertion sort?*

Exercise 3. *Let $A[1\,..\,n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an* ***inversion*** *of A.*

(1) *List the five inversion of the array $\langle 2, 3, 8, 6, 1\rangle$.*

(2) *What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?*

(3) *What is the relationship between the running time of insertion sort and the number of inversion in the input array? Justify your answer.*

(4) *Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (*Hint*: Modify merge sort.)*

Exercise 4. *Most computers can perform the operations of subtraction, testing the parity (odd or even) of a binary integer, and halving more quickly than computing remainders. This problem investigates the* ***binary gcd algorithm****, which avoids the remainder computations used in Euclid's algorithm.*

(1) *Prove that if a and b are both even, then $\gcd(a, b) = 2 \cdot \gcd(a/2, b/2)$.*

(2) *Prove that if a is odd and b is even, then $\gcd(a, b) = \gcd(a, b/2)$.*

(3) *Prove that if a and b are both odd, then $\gcd(a, b) = \gcd((a - b)/2, b)$.*

(4) *Design an efficient binary gcd algorithm for input integers a and b, where $a \geq b$, that runs in $O(\lg a)$ times. Assume that each subtraction, parity test, and halving takes unit time.*

Exercise 5. *Let*

$$p(n) \;=\; \sum_{i=0}^{d} a_i n^i,$$

where $a_d > 0$, be a degree d polynomial in n, and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties:

(1) *If $k \geq d$, then $p(n) = O(n^k)$.*

(2) *If $k \leq d$, then $p(n) = \Omega(n^k)$.*

(3) *If $k = d$, then $p(n) = \Theta(n^k)$.*

(4) *If $k > d$, then $p(n) = o(n^k)$.*

(5) *If $k < d$, then $p(n) = \omega(n^k)$.*

Exercise 6. *Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following conjectures.*

(1) *$f(n) = O(g(n))$ implies $g(n) = O(f(n))$.*

(2) *$f(n) + g(n) = \Theta(\min(f(n), g(n)))$.*

(3) *$f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$, where $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for sufficiently large n.*

(4) *$f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.*

(5) *$f(n) = O((f(n))^2)$.*

(6) *$f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.*

(7) $f(n) = \Theta(f(n/2))$.
(8) $f(n) + o(f(n)) = \Theta(f(n))$.

Exercise 7. *Suppose you have algorithms with the six running times listed below. (Assume these are the exact number of operations performed as a function of the input size n.) Suppose you have a computer that can perform 10^{10} operations per second, and you need to compute a result in at most an hour of computation. For each of the algorithms, what is the largest input size n for which you would be able to get the result within an hour?*

(1) n^2
(2) n^3
(3) $100n^2$
(4) $n \lg n$
(5) 2^n
(6) 2^{2^n}.

Exercise 8. *Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$. Justify all consecutive comparisons.*

(1) $g_1(n) = 2^{\sqrt{\lg n}}$
(2) $g_2(n) = 2^n$
(3) $g_3(n) = n^{4/3}$
(4) $g_4(n) = n(\lg n)^3$
(5) $g_5(n) = n^{\lg n}$
(6) $g_6(n) = 2^{2^n}$
(7) $g_7(n) = 2^{n^2}$

Exercise 9. *Dr. I. J. Matrix has observed a remarkable sequence of formulas:*

$$9 \times 1 + 2 = 11, \quad 9 \times 12 + 3 = 111, \quad 9 \times 123 + 4 = 111, \quad 9 \times 1234 + 5 = 11111.$$

(1) Write the good doctor's great discovery in terms of the Σ-notation.
(2) Your answer to part (1) undoubtedly involves the number 10 as base of the decimal system; generalize this formula so that you get a formula that will perhaps work in any base b.
(3) Prove your formula from part (2). The summation formulas from HW1 might be helpful.

Exercise 10. *Let $(F_n)_{n\geq 0}$ be the sequence of Fibonacci numbers, i.e., $F_0 = 0, F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.*

(1) Prove that for all $n \geq 0$, $F_n = (\phi^n - \hat{\phi}^n)/\sqrt{5}$ where $\phi = (1+\sqrt{5})/2$ and $\hat{\phi} = (1-\sqrt{5})/2$. Hint: Use that $\phi, \hat{\phi}$ are both roots of the quadratic polynomial $x^2 - x - 1$.
(2) Let $T(n)$ be the running time of Fibonacci. *Prove that $T(n) = \Theta(F_n)$.*
(3) Prove that $F_n = \Theta(\phi^n)$. Conclude that $T(n) = \Theta(\phi^n)$ runs in exponential time.

Exercise 11 (Programming Exercise)**.** *Recall the nth triangular number is given by $T_n = n(n+1)/2$, so the first few triangular numbers are*

$$1,\ 3,\ 6,\ 10,\ 15,\ 21,\ 28,\ 36,\ 45,\ 55,\ \ldots$$

We can list the divisors of the first seven triangular numbers:

$$\begin{aligned} 1 &: 1 \\ 3 &: 1, 3 \\ 6 &: 1, 2, 3, 6 \\ 10 &: 1, 2, 5, 10 \\ 15 &: 1, 3, 5, 15 \\ 21 &: 1, 3, 7, 21 \\ 28 &: 1, 2, 4, 7, 14, 28 \end{aligned}$$

We see that 28 is the first triangular number to have more than five divisors. What is the value of the first triangular number to have over a thousand divisors?

Exercise 12 (Programming exercise)**.** *The following iterative sequence is defined for the set of positive integers:*

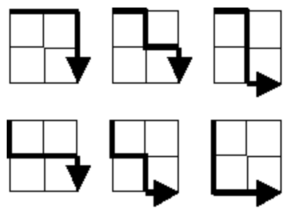
$$\begin{aligned} n &\to n/2 \quad (\textit{if } n \textit{ is even}) \\ n &\to 3n+1 \quad (\textit{if } n \textit{ is odd}) \end{aligned}$$

Using the rule above and starting with 13*, we generate the following sequence:*

$$13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

It can be seen that this sequence contains 10 terms. It is conjectured that all starting numbers will finish at 1*. Which starting number, under two million, produces the longest chain? [Note: once the chain starts the terms are allowed to go above two million.]*

Exercise 13 (Programming exercise)**.** *Starting in the top left corner of a* 2×2 *grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner. How*



many such routes are there through a 30×30 *grid?*

Exercise 14 (Programming exercise)**.** *In the United Kingdom the currency is made up of pound (£) and pence (p). There are eight coins in general circulation:*

$$1p,\ 2p,\ 5p,\ 10p,\ 20p,\ 50p,\ \pounds 1\ (100p),\ \textit{and } \pounds 2\ (200p)$$

It is possible to make £2 in the following way:

$$1 \times \pounds 1 + 1 \times 50p + 2 \times 20p + 1 \times 5p + 1 \times 2p + 3 \times 1p$$

How many different ways can £2 be made using any number of coins? Here we don't care about the order of the coins, just how many of each of them there are.

Exercise 15 (Programming exercise)**.** *An irrational decimal is created by concatenating the positive integers:*

$$0.123456789101112131415161718192\,0\ldots$$

It can be seen that the 12*th digit of the decimal expansion is* 1*.*

If d_n represents the nth digit of the fractional part, find the value of the following expression:

$$d_1 \times d_{10} \times d_{100} \times d_{1000} \times d_{10000} \times d_{100000} \times d_{1000000}$$