MATH182 HOMEWORK #4
DUE July 19, 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 seven (7) of these problems. All problems will be graded, although the score for the homework will be capped at $N :=$ (point value of one problem) $\times 7$ 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. *This problem is about heaps:*

(1) What are the minimum and maximum numbers of elements in a heap of height h?
(2) Show that an n-element heap has height $\lfloor \lg n \rfloor$.
(3) Show that, with the array representation for storing an n-element heap, the leaves are the nodes indexed by $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

Exercise 2. *The code for* MAX-HEAPIFY *is quite efficient in terms of constant factors, except possibly for the recursive call in line 10, which might cause some compilers to produce inefficient code. Write an efficient* MAX-HEAPIFY *that uses an iterative control construct (a loop) instead of recursion.*

Exercise 3. *Show that there are at most $\lceil n/2^{h+1} \rceil$ nodes of height h in any n-element heap.*

Exercise 4. *Argue the correctness of* HEAPSORT *using the following loop invariant:*

> *(Loop Invariant) At the start of each iteration of the **for** loop of lines 2-5, subarray $A[1\,..\,i]$ is a max-heap containing the i smallest elements of $A[1\,..\,n]$, and the subarray $A[i+1\,..\,n]$ contains the $n-i$ largest elements of $A[1\,..\,n]$, sorted.*

Exercise 5. *What is the running time of* HEAPSORT *on an array A of length n that is already sorted in increasing order? What about decreasing order?*

Exercise 6. *Argue the correctness of* HEAP-INCREASE-KEY *using the following loop invariant:*

> *(Loop Invariant) At the start of each iteration of the **while** loop of lines 4-6, $A[\text{PARENT}(i)] \geq A[\text{LEFT}(i)]$ and $A[\text{PARENT}(i)] \geq A[\text{RIGHT}(i)]$, if these nodes exist, and the subarray $A[1\,..\,A.\textit{heap-size}]$ satisfies the max-heap property, except that there may be one violation: $A[i]$ may be larger than $A[\text{PARENT}(i)]$.*

You may assume that the the subarray $A[1\,..\,A.\textit{heap-size}]$ satisfies the max-heap property at the time HEAP-INCREASE-KEY *is called.*

Exercise 7. *A **d-ary heap** is just like a binary heap, but (with one possible exception) non-leaf nodes have d children instead of 2 children.*

(1) How would you represent a d-ary heap in an array?
(2) What is the height of a d-ary heap of n elements in terms of n and d?
(3) Give an efficient implementation of EXTRACT-MAX *in a d-ary max-heap. Analyze its running time in terms of d and n.*
(4) Give an efficient implementation of INSERT *in a d-ary max-heap. Analyze its running time in terms of d and n.*

(5) *Give an efficient implementation of* INCREASE-KEY(A, i, k)*, which flags an error if* $k < A[i]$*, but otherwise sets* $A[i] = k$ *and then updates the* d*-ary max-heap structure appropriately. Analyze its running time in terms of* d *and* n*.*

Exercise 8. *An* $m \times n$ ***Young tableau*** *is an* $m \times n$ *matrix such that the entries of each row are in sorted order from left to right and the entries of each column are in sorted order from top to bottom. Some of the entries of a Young tableau may be* ∞*, which we treat as nonexistent elements. Thus a Young tableau can be used to hold* $r \leq mn$ *finite numbers.*

(1) *Draw a* 4×4 *Young tableau containing the elements* $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$*.*
(2) *Argue that an* $m \times n$ *Young tableau* Y *is empty if* $Y[1, 1] = \infty$*. Arge that* Y *is full (contains* mn *elements) if* $Y[m, n] < \infty$*.*
(3) *Give an algorithm to implement* EXTRACT-MIN *on a nonempty* $m \times n$ *Young tableau that runs in* $O(m + n)$ *time. Your algorithm should use a recursive subroutine that solves an* $m \times n$ *problem by recursively solving either an* $(m - 1) \times n$ *or an* $m \times (n - 1)$ *subproblem. (Hint: Think about* MAX-HEAPIFY*.) Define* $T(p)$*, where* $p = m + n$*, to be the maximum running time of* EXTRACT-MIN *on any* $m \times n$ *Young tableau. Give and solve a recurrence for* $T(p)$ *that yields the* $O(m + n)$ *time bound.*
(4) *Show how to insert a new element into a nonfull* $m \times n$ *Young tableau in* $O(m + n)$ *time.*
(5) *Using no other sorting method as a subroutine, show how to use an* $n \times n$ *Young tableau in to sort* n^2 *numbers in* $O(n^3)$ *time.*
(6) *Given an* $O(m+n)$*-time algorithm to determine whether a given number is stored in a given* $m \times n$ *Young tableau.*

Exercise 9. *Explain how to implement two stacks in one array* $A[1 \,..\, n]$ *in such a way that neither stack overflows unless the total number of elements in both stacks together is* n*. The* PUSH *and* POP *operations should run in* $O(1)$ *time.*

Exercise 10. *Consider a modification of the rod-cutting problem in which, in addition to a price* p_i *for each rod, each cut incurs a fixed cost of* c*. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.*

Exercise 11. *Modify* MEMOIZED-CUT-ROD *to return not only the value but the actual solution, too.*

Exercise 12 (Programming Exercise)**.** *Let* $d(n)$ *be defined as the sum of proper divisors of* n *(numbers less than* n *which divide evenly into* n*). If* $d(a) = b$ *and* $d(b) = a$*, where* $a \neq b$*, then* a *and* b *are an* ***amicable pair*** *and each of* a *and* b *are called* ***amicable numbers****.*

For example, the proper divisors of 220 *are* $1, 2, 4, 5, 10, 11, 20, 22, 44, 55$ *and* 110*; therefore* $d(220) =$ 284*. The proper divisors of* 284 *are* $1, 2, 4, 71$ *and* 142*; so* $d(284) = 220$*.*

Evaluate the sum of all the amicable numbers under 10000*. (This page might be helpful:* `https://en.wikipedia.org/wiki/Divisor_function`*)*

Exercise 13 (Programming Exercise)**.** *A* ***perfect number*** *is a number for which the sum of its proper divisors is exactly equal to the number. For example, the sum of the proper divisors of* 28 *would be* $1 + 2 + 4 + 7 + 14 = 28$*, which means that* 28 *is a perfect number.*

A number n *is called* ***deficient*** *if the sum of its proper divisors is less than* n *and it is called* ***abundant*** *if this sum exceeds* n*.*

As 12 *is the smallest abundant number,* $1 + 2 + 3 + 4 + 6 = 16$*, the smallest number that can be written as the sum of two abundant numbers is* 24*. By mathematical analysis, it can be shown that all integers greater than* 28123 *can be written as the sum of two abundant numbers. However, this upper limit cannot be reduced any further by analysis even though it is known that the greatest number that cannot be expressed as the sum of two abundant numbers is less than this limit.*

Find the sum of all the positive integers which cannot be written as the sum of two abundant numbers.

Exercise 14 (Programming Exercise)**.** *The number* 3797 *has an interesting property. Being prime itself, it is possible to continuously remove digits from left to right, and remain prime at each stage:* $3797, 797, 97,$ *and* 7*. Similarly we can work from right to left:* $3797, 379, 37,$ *and* 3*.*

Find the sum of the only eleven primes that are both truncatable from left to right and right to left.

Note: $2, 3, 5,$ *and* 7 *are not considered to be truncatable primes.*

Exercise 15 (Programming Exercise)**.** *If* p *is the perimeter of a right angle triangle with integer length sides,* $\{a, b, c\}$*, there are exactly three solutions for* $p = 120$*.* $\{20, 48, 52\}$*,* $\{24, 45, 51\}$*,* $\{30, 40, 50\}$*.*

For which value of $p \leq 1000$*, is the number of solutions maximised? (This page might be helpful:* `https://en.wikipedia.org/wiki/Pythagorean_triple`*)*

Exercise 16 (Programming Exercise)**.** *It is possible to show that the square root of two can be expressed as an infinite continued fraction.*

$$\sqrt{2} = 1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \cdots}}}$$

By expanding this for the first four iterations, we get:

$$\begin{aligned} 1 + \frac{1}{2} &= \frac{3}{2} \\ 1 + \frac{1}{2 + \frac{1}{2}} &= \frac{7}{5} \\ 1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}} &= \frac{17}{12} \\ 1 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}} &= \frac{41}{29} \end{aligned}$$

The next tree expansions are $99/70$*,* $239/169$*, and* $577/408$*, but the eighth expansion,* $1393/985$*, is the first example where the number of digits in the numerator exceeds the number of digits in the denominator.*

In the first one-thousand expansions, how many fractions (when put in lowest terms) contain a numerator with more digits than the denominator? (This page might be helpful: `https://en.wikipedia.org/wiki/Continued_fraction`*)*