Midterm Exam

Math 182, Fall 2021

Instructions: Answer each question in the space provided. If you run out of room, use the blank pages at the end. You may consult a single, double-sided page of notes. You may not use a calculator, phone or computer. If you believe there is a typo in a question, you may raise your hand to ask about it.

If you cannot figure out how to solve a problem, you may receive partial credit for solving an easier version of the same problem, as long as you state clearly that that's what you are doing.

Advice: The exam consists of five problems, plus one extra credit problem. Note that the last two problems are worth more points than the others, so if you are stuck on a problem early in the exam, it may be worth setting it aside for a while to work on the later problems.

Name: ______________________________

UID: ______________________________

Question	Points	Score
1	6	
2	9	
3	8	
4	15	
5	12	
6	0	
Total:	50	

Don't turn over this page until you are told to do so.

Question 1: Big-O and Friends (6 points)

State whether each of the following is true or false. If true, briefly explain why. If false, give a counterexample. You do not need to prove that your answers are correct.

(a) If $f, g, h, k \colon \mathbb{N} \to \mathbb{N}$ are functions such that $f \in \Theta(g)$, $h \in \Theta(k)$, and for all $n \in \mathbb{N}$, $f(n) > h(n)$ and $g(n) > k(n)$ then $f(n) - h(n) \in O(g(n) - k(n))$.

(b) For any functions $f, g \colon \mathbb{N} \to \mathbb{N}$, $\max(f(n), g(n)) \in \Theta(f(n) + g(n))$.

Question 2: Find the Running Time (9 points)

Find the running time of each of the algorithms below. You should show your work, but you do not need to prove that your answers are correct.

(a) This algorithm does nothing useful.

```
Foo(n):
  if n > 10:
    Foo(ceiling(n/3))
    Foo(ceiling(n/3))
    Foo(ceiling(n/3))
```

(b) This algorithm removes all negative numbers from an array by going through the entries in the array one-by-one. When it encounters a negative number, it removes it from the array and moves everything after it down by one position.

```
RemoveNegatives(A):
  i = 1
  while i ≤ length(A):
    if A[i] < 0:
      Delete(A, i)
    else:
      i = i + 1

Delete(A, i):
  for j = i, i + 1, ..., length(A) - 1:
    A[i] = A[i + 1]
  decrease length(A) by one
```

(c) This algorithm is a modification of mergesort. To sort a list it first recursively sorts the left and right halves of the list. To merge the two sorted halves together, it concatenates them and then calls `Mergesort` on the resulting list.

```
SillySort(A):
  n = length(A)
  if n = 0 or n = 1:
    return A
  left = SillySort(A[1...floor(n/2)])
  right = SillySort(A[floor(n/2) + 1...n])
  B = new list whose fist half is left and whose second half is right
  return Mergesort(B)
```

Question 3: Greedy Prefix Sums (8 points)

Given an array `A`, say that a *prefix sum* of `A` is any number of the form `A[1] + A[2] + . . . + A[i]` for some i between 1 and the length of A. Suppose you are trying to solve the following problem: given a list of integers $x_1, x_2, \ldots, x_n$ and a natural number $m > 1$, check whether it is possible to put $x_1, \ldots, x_n$ into an array in some order such that no prefix sum of the array is divisible by m.

Example. If the list is $1, 2, 3, 4$ and $m = 3$ then it is possible since the prefix sums of `[1, 3, 4, 2]` are $1, 4, 8, 10$ and none of these are divisible by 3. If the list is $1, 2, 3, 4$ and $m = 2$ then it is not possible since the sum of all the numbers is divisible by 2.

You come up with the following greedy algorithm to solve this problem. Start with an empty array. At each step, look for a number which is still in the list and which, when added to the array built so far, does not make the sum divisible by m. If such a number exists then remove the first such number from the list and add it to the end of the array. If no such number exists, then declare that there is no solution. In pseudocode:

```
Check(X, m):
  A = new empty array
  while X is not empty:
    if there is x in X such that sum(A) + x is not divisible by m:
      remove the first such x from X and add it to the end of A
    else:
      return False
  return True
```

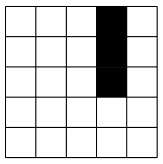
Is this algorithm correct? If so, prove that it is correct. If not, give an example of an input on which it is incorrect.

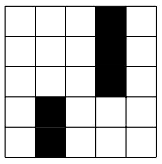
Question 4: How Many Paths? (15 points)

A *path* through an $n \times n$ grid is a sequence of squares in the grid such that each square is either horizontally or vertically adjacent to the previous square. A path is called *increasing* if each square is either to the right of or above the previous square in the path.

Suppose you are given an $n \times n$ grid where some squares have been marked as blocked. You want to know many increasing paths there are in the grid which start in the bottom left corner, end in the upper right corner and do not contain any blocked squares.

Example. Two grids are shown below, with blocked squares filled in with black. In the first grid, there are exactly 5 increasing paths from the bottom left corner to the upper right corner. In the second grid, there are none.





Assume that the input comes in the following format: `A` is an $n \times n$ array such that `A[i, j]` records whether the square in the i^{th} row and j^{th} column of the grid is blocked. In particular, assume `A[i, j]` is 0 if the square is blocked and 1 otherwise. Also assume that $(1, 1)$ corresponds to the square in the bottom left of the grid, so increasing i and j corresponds to going up and to the right in the grid, respectively.

We will use dynamic programming to solve this problem. Define $P(i, j)$ to be the number of increasing paths which start at square $(1, 1)$, end at square (i, j) and do not contain any blocked squares. For the questions below, you do not need to prove your answers are correct nor do you need to show any work.

(a) Write a recursive formula (i.e. update rule) for $P(i, j)$.

$P(i, j) =$

(b) Give the base case(s) of the recursion.

(c) What is the running time of the resulting dynamic programming algorithm?

Question 5: Search a Jumbled List (12 points)

You have decided to open up a bookstore. To make sure the books stay organized, every book is tagged with a unique integer ID and you keep the books in sorted order according to their ID. However, sometimes customers will pull a book off the shelf and put it back in a different location than they found it in. Fortunately, they always put it close to where they originally found it and no book is ever more than 2 spots away from its proper location. You would like a way to check whether a book with a given ID is contained in your store.

Design an efficient algorithm to solve this problem. Assume you are given an array of integers, L, corresponding to the ID numbers of the books in your store, and a number a, corresponding to the ID of the book you are trying to find. You may assume that no number in L is more than 2 positions away from where it would be if L was sorted in increasing order.

(a) Describe the main idea of your algorithm. You do not need to prove that your idea is correct.

(b) Show pseudocode to implement your algorithm. Note that if you run out of time to give pseudocode but the idea you described in part (a) is correct, you will still receive most of the credit for this problem.

(c) What is the running time of your algorithm? You do not need to show any work.

Question 6: Extra Credit (1 point (bonus))

This problem is optional. If you find a correct solution, you will receive one point of extra credit.

Suppose that in the previous problem, instead of every book being at most 2 spots away from its correct position, every book is at most $\log(n)$ spots away, where n is the total number of books. Find an $O(\log(n))$ time algorithm to solve this version of the problem. To receive credit for this problem you just need to describe the main idea of your algorithm; you do not need to give pseudocode or prove that your algorithm is correct.

Use this page if you run out of space on any problem. Be sure to indicate on the original page for the problem that your solution continues on a later page.

Use this page if you run out of space on any problem. Be sure to indicate on the original page for the problem that your solution continues on a later page.

Use this page if you run out of space on any problem. Be sure to indicate on the original page for the problem that your solution continues on a later page.