CSE 40113: Algorithms

Sample Midterm Exam – Spring 2025

Name: Solutions	Email Address:

1. This is a closed-book and closed-notes exam. You are allowed two "cheat sheet" on standard 8.5 by 11 inch paper, handwritten on the front and back.
2. Print your full name and your email address in the boxes above.
3. Print your name or initials at the top of every page.
4. Please write clearly and legibly. If I can't read your answer, I can't give you credit.
5. When asked to design an algorithm, you must also provide a runtime analysis for that algorithm. However, proofs of correctness are NOT required for algorithms on this exam. However, problems that explicitly ask you to prove something still require you to do proofs!
6. There are 5 problems on the exam. Your grade will be calculated based only on 4 out of the 5 problems. Please feel free to try all of them, though; I'll take the maximum of the 4 for your final grade.
7. Remember, these are NOT necessarily in order of difficulty. Please read all the problems first, and don't allow yourself to get stuck on a single problem.

#	1	2	3	4	5	Total
Max	20	20	20	20	20	80
Score						

1. Suppose we are choosing between the following 3 algorithms:

   - Algorithm A solves a problem with input size by n by dividing it into four subproblems, each of size $n/2$, recursively solving each subproblem, and then combining the solutions in quadratic time.
   - Algorithm B solves problems of size n by recursively solving a subproblem of size $n-1$ and one of size $n-2$, and then combining the solutions in constant time.
   - Algorithm C solves problems of size n by dividing them into three subproblems, each of size $n/6$, and then combining the solutions in constant time.

   What are the running times of each of these problem (in big-O notation), and which would you choose as the best?

6 points

Alg A: $A(n) = 4A(\frac{n}{2}) + O(n^2)$

Master thm: $a=4$ $b=2$ $d=2$

$\Rightarrow A(n) = \Theta(n^2 \log n)$

6 points

Alg B: $B(n) = B(n-1) + B(n-2) + O(1)$

Same as Fibonacci

$\Rightarrow B(n) = \Theta(\phi^n)$

where $\phi = \frac{1+\sqrt{5}}{2}$

6 points

Alg C: $C(n) = 3C(\frac{n}{6}) + O(1)$ $\Theta(n^0)$

MT: $a=3$ $b=6$ $d=0$

$\Rightarrow C(n) = \Theta(n^{\log_6 3})$

$\boxed{\text{C is best}}$ ← 2 point

2. Design and analyze an efficient algorithm to solve the following problem: Given a set of n integers, does the set contain a pair of elements a, b such that $a + b = 0$?

   (Hint: You can do better than $O(n^2)$.)

```
Input: $A[1..n]$
Sort A
$\ell \gets 1$
$r \gets n$
found ← false
while $(\ell < r)$
    if $(A[\ell] + A[r]) = 0$
        found ← true
    else if $(A[\ell] + A[r]) < 0$
        $\ell \gets \ell + 1$
    else if $(A[\ell] + A[r]) > 0$
        $r \gets r - 1$

return found
```

$O(n \log n)$ to sort, then $O(n)$ iterations in loop $\Rightarrow O(n \log n)$ time

16 points for algorithm, 4 for runtime

3. Consider a graph with n vertices. Recall that a subset of the vertices is called *independent* if no two of them are joined by an edge. Finding large independent sets is difficult in general, but can be done on some simple classes of graphs, as we discussed in class.

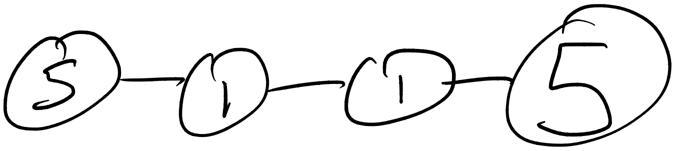
   For this problem, we will consider only graphs, so the vertices can be written as $v_1, v_2, \ldots, v_n$; for every i between 1 and $n-1$, we then add the edge between each v_i and v_{i+1}.. With each vertex v_i, we associate a weight w_i.

   Formally, your input is just an array of weights $W[1..n]$; you don't really need to explicitly store edges and vertices here, since each vertex i connects to its neighbors $i-1$ and $i+1$ (except for the endpoints, vertex 1 and vertex n, which only have one neighbor each).

   (a) Construct an example showing why the following different simple *greedy* algorithm does NOT always work.

```
$S_1 \leftarrow \{v_i$ with $i$ odd$\}$
$S_2 \leftarrow \{v_i$ with $i$ even$\}$
oddsum $\leftarrow$ sum of all weights in $S_1$
evensum $\leftarrow$ sum of all weights in $S_2$
if evensum $>$ oddsum
    return $S_2$
else
    return $S_1$
```

5 points



max is 10,
greedy gives 6

   (b) Give an algorithm that takes an array $W[1..n]$ of vertex weights and returns an independent set of maximum total weight. (Note that this is different from the usual largest independent set problem, since here we take the weights into account!) Your running time should be polynomial in n.

15 points:
10 alg,
5 runtime

```
MaxIS(W[1..n]):
    Initialize array B[1..n]
    B[n] ← W[n]
    B[n-1] ← max{W[n-1], W[n]}
    for i ← n-1 down to 1
        B[i] ← max{B[i+1],
                   W[i]+B[i+2]}
    return B[1]
```

Runtime:
$O(n)$
for loop

4. You are consulting for a trucking company that does a large amount of shipping between Boston and New York. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight w_i. The trucking station is quite small, so at most one truck can be at the station at any time. Company policy requires that boxes are shipped in the order they arrive, since otherwise a customer might get upset at the unfair ordering.

   At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way. But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Your job is to prove that they are in fact just fine: Prove that for a given set of boxes with specified weights w_i that the greedy algorithm currently in use actually minimizes the number of trucks that are needed.

Proof by contradiction:

Suppose opt ≠ greedy

Then at some point, opt must make different a first different decasion than greedy

→ say at truck i.

We know boxes can't shuffle, & greedy will continue to pack if room, so opt must send it with space left. Modify opt by adding the next box, & leave rest unchanged

↪ greedier, & no worse than opt

5. The new swap-puzzle game Candy Swap Saga XIII involves n cute animals numbered from 1 to n. Each animal holds one of three types of candy: circus peanuts, Heath bars, and Cioccolateria Gardini chocolate truffles. You also have a candy in your hand; at the start of the game, you have a circus peanut.

   To earn points, you visit each of the animals in order from 1 to n. For each animal, you can either keep the candy in your hand or exchange it with the candy the animal is holding.

   - If you swap your candy for another candy of the same type, you earn one point.
   - If you swap your candy for a candy of a different type, you lose one point. (Yes, your score can be negative.)
   - If you visit an animal and decide not to swap candy, your score does not change.

   You must visit the animals in order, and once you visit an animal, you can never visit it again. Describe and analyze an efficient algorithm to compute your maximum possible score. Your input is an array $C[1 \dots n]$, where $C[i]$ is the type of candy that the i^{th} animal is holding.

Recursion: Best(i, type)

max score possible from $C[i..n]$ if I have candy "type"

$$= \begin{cases} \text{if type} = C[i], \\ \quad 1 + \text{Best}(i+1, \text{type}) \\ \text{if type} \neq C[i] \\ \quad \max \begin{cases} (-1) + \text{Best}(i+1, C[i]) \\ \text{Best}(i+1) \end{cases} \end{cases}$$

Convert each type of candy to a number, 1, 2, or 3, & store n×3 array →

✓ Let "peanuts" = 1
"health" = 2 &
(scratch paper)
"truffles" = 3

```
BestCandy(C[1..n], mycandy):
    for $i \leftarrow 1$ to 3
        if $C[n] = i$
            $Best[n, i] \leftarrow 1$
        else $Best[n, i] \leftarrow 0$
    for $k \leftarrow n-1$ to 1
        for $i \leftarrow 1$ to 3
            if $C[k] = i$
                $Best[k, i] \leftarrow 1 + Best[k+1, i]$
            else
                $Best[k, i] \leftarrow \max\{Best[k+1, i], Best[k+1, C[k]] - 1\}$
    return Best[1, mycandy]
```

O(n) time total

(scratch paper)