Name:

ID:

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3A (55 points) **Summer 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name:

ID:

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3A (55 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2
Question 3
Question 4

Recommended reading:
Greedy Algorithms:

Name:

ID:

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3A (55 points) **Summer 2020, CU-Boulder**

1. *(5 pts) Assume you run your Huffman tree algorithm and you produce the following pre-fix codes. Describe why there must be an error in your algorithm.*

```
S = 11
c = 10
i = 110
e = 100
n = 010
```

Optimal codes are taken from the leaves, but in this case codes are taken from internal nodes (as in letter c). Pair of (S and i) and (C and e) are having same prefix, hence not satisfying the prefix code property of Huffmann coding. This makes decoding of string difficult. Even code for letter n is not satisfying the full binary tree property.

This can be checked from the Lemma 16.2 of Cormen which states that there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit. None of the above prefix code satisfies this lemma.

2. *(5 × 2 = 10 pts) Consider the given Huffman Tree.*

   (a) Decode the string ***"10011111010010001101101001"*** *encoded using the above Huffman encoding tree.*

   Following are the prefix code:
   - $b = 00$
   - $u = 1000$
   - $s = 1001$
   - $o = 101$
   - $f = 110$
   - $k = 111$

   Encoded string: 1001—111—101—00—1000—110—110—110—110—1001
   Decoded string: skobufffs.

(b) Decode the string ***"1011110010001001"*** *encoded using the above Huffman encoding tree.*

Following are the prefix code:

- $b = 00$
- $u = 1000$
- $s = 1001$
- $o = 101$
- $f = 110$
- $k = 111$

Encoded string:101—111—00—1000—10001
Decoded string: ofbus.

Name:

ID:

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3A (55 points) **Summer 2020, CU-Boulder**

3. *(15 pts) Consider the recurrence $F_n = F_{n-1} + 2F_{n-2}$, with the base cases $F_0 = 1$ and $F_1 = 2$. Suppose we have letters $v_0, \ldots, v_7$; the frequency of v_i is given by F_i, where $i \in \{0, 1, 2...6, 7\}$. Draw a Huffman tree for $v_0, \ldots, v_7$.*

$$F_2 = F_1 + 2 * F_0$$
$$F_2 = 2 + 2 * 1$$
$$F_2 = 4$$

$$F_3 = F_2 + 2 * F_1$$
$$F_3 = 4 + 2 * 2$$
$$F_3 = 8$$

$$F_4 = F_3 + 2 * F_2$$
$$F_4 = 8 + 2 * 4$$
$$F_4 = 16$$

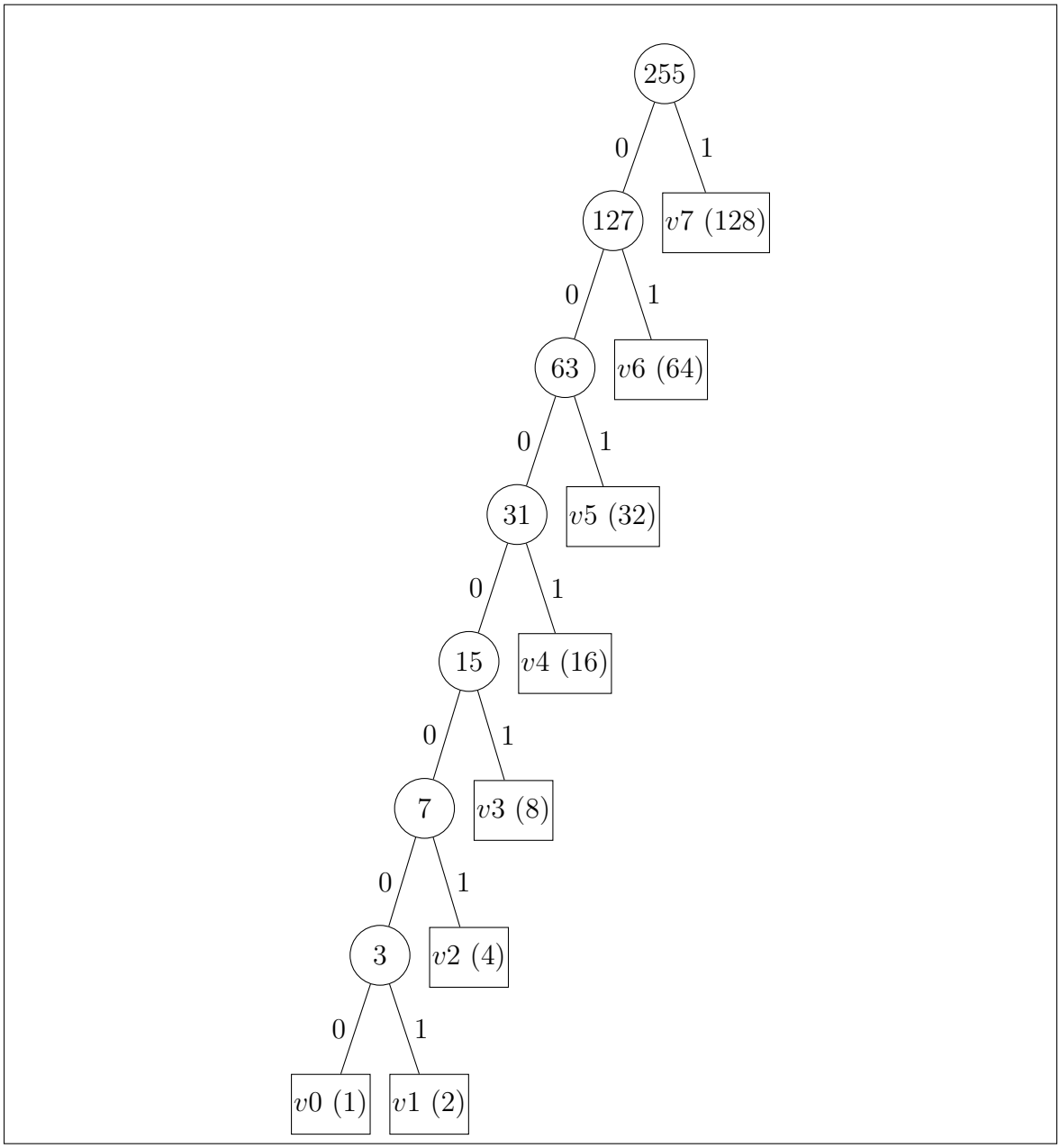
$$F_5 = F_4 + 2 * F_3$$
$$F_5 = 16 + 2 * 8$$
$$F_5 = 32$$

$$F_6 = F_5 + 2 * F_4$$
$$F_6 = 32 + 2 * 16$$
$$F_6 = 64$$

$$F_7 = F_6 + 2 * F_5$$
$$F_7 = 64 + 2 * 32$$
$$F_7 = 128$$

Here the table of letters with corresponding frequency calculated from the recurrence relation.

v_i	$F_n = F_{n-1} + 2 * F_{n-2}$	Prefixcode
v_0	1	0000000
v_1	2	0000001
v_2	4	000001
v_3	8	00001
v_4	16	0001
v_5	32	01
v_6	64	01
v_7	128	1



4. *(25 pts) In this question we consider the changemaking problem, of making change for n cents using the smallest number of coins. Suppose we have coins with denominations of $v_1 > v_2 > ...v_r$ for r coins types, where each coin's value v_i is a positive integer. Your goal is to determine how many coins of each denomination d_i are required to*

reach the sum n, such that the total number of coins used is minimized (i.e the value of $\sum_{i=1}^{r} d_i = k$ *is minimized).*

Note that the sum of all included coins should be n. (i.e $\sum_{i=1}^{r} d_i v_i = n$*)*

(a) *(15 pts) A greedy algorithm for making change is the* ***cashier's algorithm****. In cashier's algorithm at each iteration we add a coin of highest value ensuring that it does not take us past the value n. The above step is repeated until the sum reaches n. If it is not possible to reach the sum n the algorithm returns no solution as the answer. Consider the following pseudocode meant to implement the cashier's algorithm where n is the amount of money to make change for and v is a vector of the coin denominations sorted in descending order and the vector d should hold the count of number of coins in each denomination.*

```
1  get_change(n, v, r):
2  {
3      d[1...r] = 1
4      while(n>0)
5      {
6          k=r
7          while(k>0 and v[k]>n)
8          {
9              k++
10         }
11         if(k <= 0)
12         {
13             return 'no solution'
14         }
15         else
16         {
17             n = n-v[k]
18         }
19     }
20     return d
21 }
```

This code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

Bug 1: $d[1 \cdots r]$ should be initialized to zero,as in the begining no denomination is selected.

Bug 2: Value of d is not been updated in the while loop, all denomination values would be equal for any cases.

Bug 3: Condition of 2^{nd} while loop should be $(k > 0 and v[k] \geq n)$ and value of k should be decremented by 1.Otherwise with the given condition, the code runs into infinite loop.

(b) (10 *pts*) *Assume that a country has the following coin denominations.* $\{\$1, \$6, \$10, \$15\}$. *Provide two examples for which the greedy algorithm does not yield an optimal solution for making change. In both the examples also show the optimal solution adds that adds up to the same value using smaller set of coins.*

Eg 1: For $n = 12$, greedy method would first select \$10 and then it will select 2 \$1 coins ie $d[1, 6, 10, 15] = [2, 0, 1, 0]$.Total of 3 coins selected. If we draw recurrence tree for (\$1,\$6,\$10,\$15,12) →(\$10,2)→(\$1,1)→(\$1,0), we see 2 similar pattern,where \$1 is repeated twice.
Optimal solution would be selecting 2 \$6 coins. ie $d[1, 6, 10, 15] = [0, 2, 0, 0]$. Total of 2 coins selected.

Eg 2: For $n = 13$, greedy method would first select \$10 and then it will select 3 \$1 coins ie $d[1, 6, 10, 15] = [3, 0, 1, 0]$.Total of 4 coins selected.
Optimal solution would be selecting 2 \$6 and 1 \$1 coins. ie $d[1, 6, 10, 15] = [1, 2, 0, 0]$. Total of 3 coins selected.

5. ***Extra Credit (5% of total homework grade)*** *For this extra credit question, please refer the leetcode link provided below or click here. Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.*

   Please provide your solution with proper comments which carries points as well.

   https://leetcode.com/problems/jump-game/

```
class Solution {

public:

bool canJump(vector<int>& nums)
{
//2 pointer approach, Time complexity:O(n), space complexity:O(1).
//one ptr is the start and another ptr is placed at the second last element

        int n = nums.size()-1;
        int lptr=0,rptr=nums.size()-2;

//Traversing the array from right to left until the rptr
// reaches the beyond the index lptr

        while(lptr<=rptr)

        {
                jmp_stp = n-rptr;

//Check relative to that index,whether jump is possible or not
//If jmp is possible, update n to index of right pointer
//else decrement index of right ptr

                if(nums[rptr]>=jmp_stp)

                {
                        n = rptr;
                        rptr--;
                }
```

```
31
32                  else
33                          rptr--;
34
35
36          }
37  //If the requuired jump is equal to 0, last index has been reached.
38
39          if(n==0)
40                  return true;
41          else
42                  return false;
43          }
44  };
```