CMPE/CISC-365*
Test #2
October 22, 2019

Student Number (Required) _____

Name (Optional)_____

This is a closed book test. You may refer to one 8.5 x 11 data sheet.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, **but will not be re-marked under any circumstances.**

The test will be marked out of 50.

Question 1	/16
Question 2	/12
Question 3	/20
Question 4	/2
TOTAL	/50

"There is a very fine line between loving life and being greedy for it."

— Maya Angelou

QUESTION 1 (16 Marks)

Suppose we have a computer which is based on the trinary system, rather than binary. The fundamental unit of memory of such a system is called a trit (instead of bit). We represent everything with tritstrings consisting of 0's, 1's and 2's. In such a system, the standard representation of the letter "A" might be "102210", "B" might be "102211" etc.

Part A : [8 Marks]

Adapt the Huffman Coding scheme to the trinary system, and give a clear description of your modified algorithm for constructing variable length trinary codes. You are not required to prove that your algorithm produces optimal trinary codes.

Solution:

Sort the characters in the source document according to their frequency (same as the original algorithm)

Build a trinary tree as follows:
choose the three characters with the lowest frequency, add a parent that has their combined frequencies, and put a 0, a 1 and a 2 on the edges joining them to their parent.
Remove the three characters from the set and add their parent (as a new character) to the set.
Repeat until there is a single root that represents the combination of all the characters.

Marking:

Sorting the set:	2 marks
Choosing the three smallest:	2 marks
Adding 0, 1, 2 to their codestrings:	2 marks
Replacing them by a combination item with their summed frequencies:	2 marks

A student whose answer shows a good understanding of the basic Huffman algorithm should get at least 4/8, even if they make errors in translating it to the trinary version.

They are not required to present the algorithm in terms of building a tree. They can describe the process as "add 0, 1, 2 respectively to the codestrings for the characters represented by the three lowest frequency items"

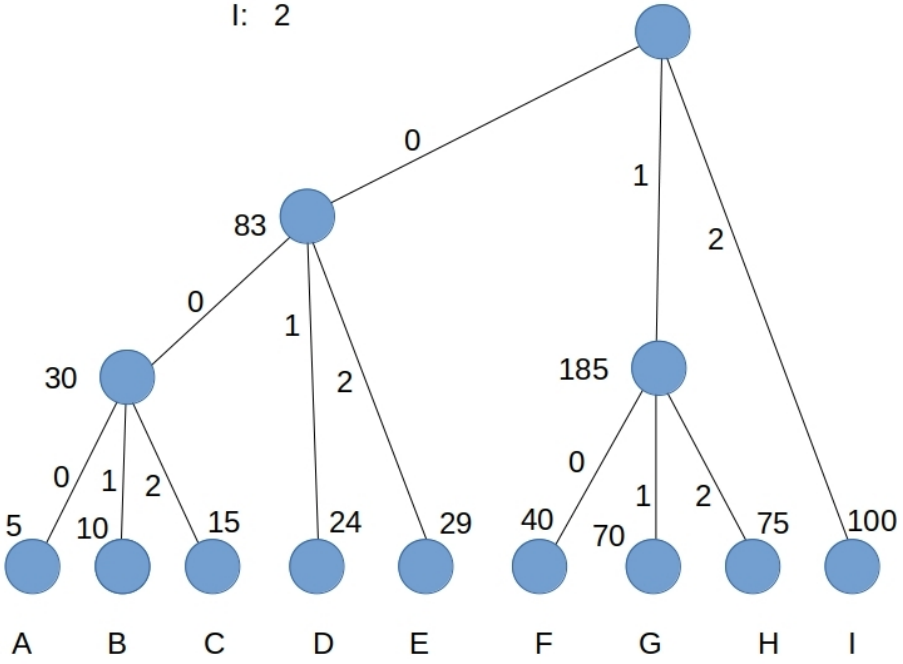
Part B : [8 marks]

Show the application of your modified algorithm to the following set of letters, where each letter is followed by its observed frequency. Show the tree and codes that your algorithm constructs:

A	5
B	10
C	15
D	24
E	29
F	40
G	70
H	75
I	100

Codestrings:

A: 000
B: 001
C: 002
D: 01
E: 02
F: 10
G: 11
H: 12
I: 2



Marking:

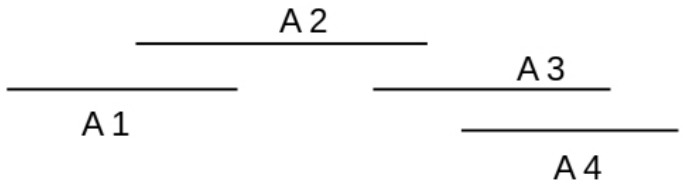
The assignment of 0,1 and 2 to the edges of the tree are arbitrary so the codestrings they construct may be very different than mine, but the lengths should be the same ("I" should have a codestring of length 1, etc.)

It is important to correctly extract the codestrings from the tree (or alternative representation). Some students may read the codestrings from the bottom up rather than from the top down, getting (for example) "100" for "B". This breaks the prefix rule and makes the code unusable.

Showing the steps of the execution:	3 marks
Showing the codestrings correctly:	5 marks
Showing the codestrings incorrectly (see explanation above):	2 marks

QUESTION 2 (12 Marks)

Suppose we have a set of n activities, each with a known start time s_i and finish time f_i . The activities may overlap. Our task is to assign the activities to rooms so that each room contains a non-overlapping subset of the activities. The goal is to use as few rooms as possible.



In this example we need two rooms: one room for A1 and A3 and the other room for A2 and A4.

A greedy algorithm for this problem: sort the activities based on start time, then assign activities to rooms. Use a new room only if the next activity overlaps with activities in all existing rooms.

```
Sort the activities based on start time and renumber them so that
                    s_1 ≤ s_2 ≤ ··· ≤ s_n
    Room_set = {1}
    Busy_until[1] = 0
    for i = 1 to n:
        if there is any room x in Room_set with Busy_until[x] ≤ s_i :
            assign Activity i to Room x
            Busy_until[x] = f_i
        else:
            add a new room to Room_set
            assign Activity i to the new room
            set Busy_until[new room] = f_i
```

Question 2 continues on the next page.

Suppose the Algorithm puts Activities $1, 2, \ldots i$ into Room 1 and then puts Activity $i + 1$ into Room 2.

Prove that there is an optimal solution that does exactly the same thing.

Hint: Let O be an optimal solution ...

Solution:

Let O be an optimal solution, and suppose it does something different with the first i+1 Activities. Renumber the rooms so that Activity 1 is in Room 1. This does not change the number of rooms so this renumbered solution is still optimal. Call it O'

Let Activity j+1 be the first Activity that O' does not put in Room 1. (That is, O' puts Activity 1, 2, ..., j in Room 1.) If j = i, then renumbering the room that contains Activity i+1 to be Room 2 exactly matches the algorithm's action. If $j \neq i$ then it must be true that j < i, since if j > i then the algorithm would not have put Activity i+1 into Room 2.

Let the Room containing Activity j+1 be Room k. Swap Activity j+1 and all following activities in Room k with all activities in Room 1 that follow Activity j. Because the earliest activity being swapped into Room k must have start time $\geq s_{j+1}$, this is a feasible solution, and since it doesn't use more rooms it is also optimal.

This new optimal solution agrees with the Algorithm's solution more than the previous one did. We can repeat this swapping action until all of Activity 1, ... Activity i are in Room 1, and Activity i+1 is in Room 2. This is exactly what the algorithm does.

Marking:

The main thing to look for here is whether the student understands how to approach this type of problem. The details are less critical.

Recognizing that our goal is to take an arbitrary optimal solution and manipulate it to create another one that matches the algorithm's choices: 4 marks

Recognizing that no optimal solution can put Activities 1, 2, ..., i+1 into the same room: 4 marks

Recognizing that we can swap Activities (or groups of Activities) between rooms without creating time-conflicts: 4 marks

Please give part marks to answers that show partial success with these aspects of the proof.

If a student takes a completely different approach and you are not sure how to grade it, please contact me.

QUESTION 3 (20 marks)

Let $S = \{s_1, s_2, ..., s_n\}$ be a set of n positive integers – possibly containing duplicates. Let k be a positive integer.

Problem: Find a **maximum-size** subset A of S that has sum $\leq$ k

For example, let S = {7, 4, 12, 1, 3, 18, 1, 240, 10} and k = 19

The solution is A = {7, 4, 1, 3, 1} (in any order) which has size 5.

Part A : [6 marks]

Create a Greedy Algorithm to solve this problem. State your algorithm in clear pseudo-code.

Solution:

Sort the values into ascending order, so $s_1 \leq s_2 \leq \ldots s_n$

```
total = 0
i = 1
solution = {}                # empty set
while total + s_i <= k:
        total = total + s_i
        i = i+1
        solution.append(s_i)          # or "add s_i to the solution"
```

Marking:

Sort:	2 marks
Loop:	4 marks

No penalty if they forget to initialize the solution be empty – it's an important implementation detail but not an essential conceptual part of the algorithm.

If a student gives an incorrect algorithm, but remembered that Greedy Algorithms always sort the set then iterate through the sorted list, they should get 4/6

Part B : [14 marks]

Prove that your algorithm finds an optimal solution. Use any valid proof technique.

Solution:

Let A be the algorithm's solution and let O be any optimal solution. Sort O into ascending order.

If A and O are identical, then A is optimal.

Suppose A and O are equal up to and including s_i***, but differ in the next position. The algorithm fills the next position with*** s_{i+1}***, so O must fill the next position with*** s_x ***where x > i+1. This implies*** $s_x \geq s_{i+1}$***, so we can remove*** s_x ***from O and replace it with*** s_{i+1} ***without pushing the total over k. This new solution has the same cardinality as O, so it is also optimal, and it has fewer differences from A.***

We can repeat this sequence until we arrive at an optimal solution that has 0 differences from A – so A is optimal.

TL;DNR version of this proof:

Let O be any optimal solution that does not contain the smallest value in the set. Swap the smallest value for any value in O. The result is still optimal. Continue until all the smallest values have been swapped in. This matches the algorithm's solution.

Alternative Proof: Induction on the size of the set of values.

Base case: If |S| = 0, then the empty set is the only solution (and thus it is the optimal solution.

Inductive Hypothesis: Assume the algorithm always finds an optimal solution when the size of the set is $\leq n$*, for some* $n \geq 0$*.*

Let |S| = n+1, and assume the set has been sorted into ascending order. If $s_1 > k$*, there is no nonempty subset that sums to* $\leq k$*, and the algorithm correctly solves this case.*

Assuming there is a non-empty solution, let A be the algorithm's solution and let O be any optimal solution that does not contain s_1*. Replace any element of O with* s_1*. The result is still an optimal solution (call it O'), so the algorithm's first action is correct. This reduces the problem to a set of size n with a target value of* $k - s_1$*. By the inductive hypothesis, the algorithm finds an optimal solution to this reduced problem.*

O' also contains a solution to this same subproblem. This implies |A| = |O'| so A is optimal.

Marking:

The marking method here should be similar to Question 2, but it will depend on the proof type chosen by the student.

For the "eliminate differences" approach the essential concept is summarized in the TL;DNR version. If they express this idea clearly they should get at least 10/14

Example of an answer which is insufficiently clear:

"We should never take a larger value when a smaller one is available". I would grade this at 7/14. The idea is there but it is not fully developed.

For the inductive approach, use this grading scheme

Base case:	4 marks
Inductive Hypothesis:	3 marks
Inductive Step:	7 marks

In each part, give partial marks for proofs that have the right ideas but don't express them clearly.

Note that the base case can be set up with sets of size 1 rather than with the empty set.

QUESTION 4 (2 Marks)

True or false:

David Huffman was a pioneer in the field of mathematical origami.

TRUE FALSE

Solution: True

Marking:

True	**2 marks**
False	**2 marks**
No answer	**2 marks**

Yes, everyone gets 2 marks for this question. Apparently some people think I am trying to trick them with the different font sizes for TRUE and FALSE.