CSCI 2300 — Algo
Homework 7
Hayden Fuller

- **Q1** Given a undirected, weighted graph $G = (V, E)$. Assume that no two edges have the same weight. Prove that there is a unique minimum spanning tree (MST) for such a graph.
  proof by contradiction
  assume there are two different MST's
  since they contain the same nodes, they must differ by an edge.
  choose the minimum weight edge $e \in E_1 - E_2$
  E2 makes a tree, so adding edge e will create a cycle
  this cycle must contain $f \in E_1$, or else this same cyclel would exist in E_1 and it wouldn't be a tree
  since e was the smallest edge in E_1, and the weights can not be equal, $e < f$
  we can then get $E_3 = E_2 + e - f$, which will have a smaller weight than the origional E_2,
  a tree existing with smaller weight contradicts that E_2 is an MST
  Therefore, there is only one unoique MST for a graphs with unique edge weights.

- **Q2** Consider the following algorithm for finding a MST. Given a graph $G = (V, E)$, sort the edges in decreasing order of their weights. Pick each edge e in sorted order, and if e creates a cycle in G, remove it from G. Note that each time you remove an edge, you are modifying the graph, so that the next check for a cycle will be on the reduced graph. Prove that this method is correct, and analyze its running time.
  Reverse of Kruskal's algorithm
  Let T be the final set of edges, must show that T is an MST.
  T must be a spanning tree because we remove only and all cycles
  Via the cut property, if the weight of an edge is greater than the weights of all other edges crossing the cut, then this edge can npt belong to the MST
  Since the algorithm deletes edges in decreasing order of weights, it will always be the largest edge being checked
  Since we remove only edges that create a cycle, the graph will remain connected
  Since we check if all edges make a cycle and remove them if they do, there will be no cycles remaining at the end
  Therefore, this produces an MST

  Sorting edges takes O(E log E) time, checking for cycles takes O(E+V), giving us O((E log E) + V)

- **Q3** Given an alphabet with n characters. Let the characters be numbered from 1 to n, and let the frequency of character i be $f_i = \frac{1}{2^i}$ for $i = 1, 2, ..., n-1$, with the last character n having frequency $f_n = \frac{1}{2^{n-1}}$. For example, if $n = 6$, then the frequencies of the first five characters are 1/2, 1/4, 1/8, 1/16, 1/32, respectively, and the frequency of the last character is 1/32. Answer the following questions:

  (a) Show that the frequencies of all characters sum to 1 (as they should) for any n.
  $\sum_i^n f_i = \sum_i^{n-1} \frac{1}{2^i} + \frac{1}{2^{n-1}}$
  proof via induction:

base: $f_0 = 1$
induction: assume $\sum_i^n f_i$=1
$\sum_i^{n+1} f_i = \sum_i^n \frac{1}{2^i} + \frac{1}{2^n}$
$\frac{1}{2^n} + \frac{1}{2^n} = \frac{1}{2^{n-1}}$
$\sum_i^{n+1} f_i = \sum_i^{n-1} \frac{1}{2^i} + \frac{1}{2^{n-1}}$
$\sum_i^{n-1} \frac{1}{2^i} + \frac{1}{2^{n-1}} = \sum_i^n f_i$
$\sum_i^{n+1} f_i = \sum_i^n f_i = 1$
Therefore, $\sum_i^{n+1} f_i = 1$

(b) Show what the Huffman encoding is for each character. In building the encoding tree, the larger frequency child should be on the left, and if there are two groups of characters with the same frequency, the one with the smaller index should be on the left.
$f_i = \frac{1}{2^i}$ except for the last one being doubled. The tree is a long line of internal nodes with two leaves on the bottom and a leaf from each internal node. All sets of children will alawys be of equal frequency, so the order is determined by index. Assuming new internal nodes are added with the highest index, they will always be on the right, and this tree will go down and right. This means ever node n has code "1" $* (n-1) \cdot 0$ (n-1 1's and a 0 added at the end.

(c) What is the expected number of bits per character? Let the encoding length of character i be denoted as l_i, then the expected or average number of bits for the encoding is computed as $\sum l_i f_i$. Use this formula to derive a closed form expression (for any n).
number of bits per character is determined by it's index in the alphabet, $l_i = i$, and $f_i = \frac{1}{2^i}$, so $\sum \frac{i}{2^i} =$ yeah I'm out of time and need to turn this in...