**CSE 421**  
*Introduction to Algorithms*  
**Assignment #3 (rev. a)**  
Due: Friday, 4/21/17

Review the instructions in HW #1.

Caution: In my experience, “solutions” to problems about greedy algorithms given without correctness proofs are very likely to be incorrect. Proofs matter!

1. [10 points] Prof. Flashbulb proposed the following greedy algorithm for solving the interval partitioning problem (aka interval coloring, aka scheduling all intervals; section 4.1, p 122): apply the greedy algorithm for interval scheduling from the earlier part of section 4.1 (the algorithm on page 118); give all of those intervals one color, remove them from consideration, and recursively apply the same method to the resulting reduced set of intervals (using new colors). Does this method does give an optimal solution to the problem? Either prove it or give a simple counterexample and explain why your counter example is a counter example.

2. [10 points] A quick look at the algorithm on page 124 for the interval partitioning problem suggests an \( O(n^2) \) time bound: an outer loop for \( j = 1, \ldots, n \), and an implicit inner loop “for each interval \( I_i \)” that also seems to need order \( n \) time.

   (a) The algorithm on pg 124 appears to assume that \( d \) (the depth of the problem instance; pg 123) is known in advance. However, the very simple variant of it given on slide 29 (approximately) avoids this assumption, and determines \( d \) while computing the partition. (How? Basically, assume the depth is 1 until you are faced with 2 overlapping intervals; assume it is 2 until you see 3, etc., and partition accordingly.) Give a slightly more detailed description of an implementation of the later algorithm that will take only \( O(dn) \) time. (For your timing analysis, ignore the \( \Theta(n \log n) \) time for the initial step that sorts the intervals by start time.)

   (b) If \( d \) is small, the above method typically will be much faster than the simple \( (n^2) \) analysis suggests. Show, however, that it is not faster in the worst case. I.e., show that there are instances where it takes time \( \Omega(n^2) \). [Note that to show a lower bound like this, it’s not good enough to show one bad example; you really need to show that there are infinitely many bad examples, for larger and larger values of \( n \).]

   (c) Give a more clever implementation of the algorithm whose running time is \( O(n \log n) \), (even if depth is \( \Omega(\log n) \)) and justify this time bound. [Hint: use a better data structure.]

3. [10 points] Suppose you are designing a new Computer Science building. It has one very long corridor, along which all faculty offices are located, say at positions \( x_1 < x_2 < \cdots < x_n \) (arbitrary real numbers, for simplicity). Of course, faculty will die if their offices are not located within 100.0 feet of a combined espresso cart/WiFi hotspot.

   (a) Briefly describe a simple greedy algorithm which, given the \( x_i \), will find locations \( g_1 < g_2 < \cdots < g_m \) satisfying the above constraint (i.e., for each \( 1 \leq i \leq n \) there is a \( 1 \leq j \leq m \) such that \( |x_i - g_j| \leq 100 \)) while minimizing the total number \( m \) of carts/hotspots.

   (b) Explain why your algorithm is correct. In particular, argue that any solution that differs from the one your algorithm specifies can be replaced by one with no additional carts/hotspots that also satisfies the constraints and is more similar to your algorithm’s solution.

4. [10 points] The proof in the text and slides for the optimality of the earliest finish first algorithm for interval scheduling (roughly slides 7-23) is an example of the “greedy stays ahead” paradigm. It is also possible to prove optimality via an “exchange” argument. Complete the following proof sketch illustrating this approach. A *schedule* is just an ordered list of jobs. Let \( G \) be the greedy schedule (ordered by increasing finish time) and let \( H \), among all optimal schedules, be the one that has the longest prefix in common with \( G \). (A *prefix* is just some number of consecutive elements starting at the front of the list.) Argue that if \( G \) is not equal to \( H \), then an exchange can be made, creating another optimal schedule \( H' \) that has a longer prefix in common with \( G \), contradicting the choice of \( H \).

5. [10 points]
(a) Simulate the Huffman tree algorithm on the input alphabet a, b, c, d, e with frequencies .15, .30, .50, .02, .03, resp. Show the intermediate “trees” built by the algorithm, as well as the final tree.

(b) What will be the total length in bits of the compressed text built from a sequence of 100 letters having the above frequencies using the Huffman code you calculated?

(c) Suppose I tell you that the frequencies of a and b are .15, .30, resp, as above, but I don’t tell you the other frequencies. I can quickly tell (e.g., without running the full Huffman algorithm) that the Huffman code for this data is not the one shown in figure 4.16 (a) (pg 168). Explain how.

6. [10 points] A set of \( n \) records is to be stored on a linear magnetic tape (yes, old school). For each \( i \), the \( i \)th record has length \( L_i \) and will be accessed \( N_i \) times. The tape is kept rewound and, whenever a record is accessed, that record and all those preceding it on the tape must be traversed. The cost of accessing a record is defined as the total distance traversed in accessing the record. Thus, if the records are placed on the tape so that record 1 comes first followed by records 2, 3, \( \cdots \), \( n \) in numerical order, then the cost of each access to record \( i \) will be \( L_1 + L_2 + \cdots + L_i \) and the total cost of all accesses will be \( \sum_{i=1}^{n} \sum_{j=1}^{i} N_i L_j \). Give a simple greedy algorithm to determine the order in which the records should occur on the tape to minimize the total cost of all accesses.

7. [10 points]

(a) In a prefix code tree (as drawn in my Huffman slides), the frequency \( f(v) \) of a leaf \( v \) is the frequency of the corresponding letter in the corpus being compressed. The Cost of a tree, as defined on slide 11, is the sum over leaves of leaf frequency times leaf depth. This is the average number of bits per character attained by this code. Define the frequency \( f(v) \) of an internal node \( v \) (i.e., a non-leaf) in the tree to be the sum of the frequencies of the leaves in the subtree rooted at \( v \). Show that the cost of the tree is equal to the sum of the frequencies of all internal nodes in the tree, and also equal to the sum of the frequencies of all nodes other than the root. (Note: normally, “frequencies” are non-negative fractions summing to 1; for simplicity, my slides labeled nodes with frequencies times 100.)

(b) With the definition of “frequency” given above for internal nodes, the definition of inversion given on slide 13 can be extended from pairs of leaves to arbitrary pairs of nodes. Assuming that the tree is a “full” tree, i.e., that each internal node has exactly 2 children, and that no leaf has frequency zero, show that the claim on that slide remains true: exchanging inversions never increases cost. (Be careful that your argument includes “ties”—equal frequency and/or equal depth is still an inversion.)

8. [10 points] Suppose you break a file into 8 bit characters and find that all 256 combinations are present, with the most frequent occurring less that twice as often as the least frequent. Prove that compressing this file with the Huffman code for this data will not result in a shorter file.

Revision History: