Anthony Rusignuolo

CS 600WS – Advanced Algorithms

Reza Peyrovian

Homework 5

I pledge my honor that I have abided by the Stevens Honor System.

1. R-10.6 Give an example set of 8 characters and their associated frequencies so that the Huffman tree for this set is a complete binary tree.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g | H |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

This would result in a complete binary tree whose root would have a frequency of 8

1. C-10.5 Describe an efﬁcient greedy algorithm for making change for a speciﬁed value using a minimum number of coins, assuming there are four denominations of coins (called quarters, dimes, nickels, and pennies), with values 25, 10, 5, and 1, respectively. Argue why your algorithm is correct.
   1. Starting with the largest coin denomination, dispense coins and subtract their value from the specified value until the denomination is greater than the remainder of the specified value, then repeat this process with the next largest coin denomination until the appropriate amount of change has been made. This algorithm ensures that the larger denominations will be used as much as possible, reducing the necessity for the smaller denominations to make up the same dollar amount, but with more coins.
2. A-10.1 In the art gallery guarding problem we are given a line L that represents a long hallway in an art gallery. We are also given a set X = {x0,x1,...,xn−1} of real numbers that specify the positions of paintings in this hallway. Suppose that a single guard can protect all the paintings within distance at most 1 of his or her position (on both sides). Design an algorithm for ﬁnding a placement of guards that uses the minimum number of guards to guard all the paintings with positions in X.
   1. First, sort X in increasing order. Then, iterate through X and place a guard after every unprotected painting.
3. R-11.1 Characterize each of the following recurrence equations using the master theorem (assuming that T(n)=c for n<d, for constants c>0 and d ≥ 1).
4. T(n)=9T(n/3) + n3log n
   1. a.  
      This reaches the base case when or .  
      Therefore, the equation is .
   2. b.  
      This reaches the base case when or .  
      Therefore, the equation is .
   3. c.  
      Therefore, the equation is .
   4. d.  
      Therefore, the equation is .
   5. e.  
      Therefore, the equation is .
5. C-11.3 There is a sorting algorithm, “Stooge-sort,” which is named after the comedy team, “The Three Stooges.” if the input size, n, is 1 or 2, then the algorithm sorts the input immediately. Otherwise, it recursively sorts the ﬁrst 2n/3 elements, then the last 2n/3 elements, and then the ﬁrst 2n/3 elements again. The details are shown in Algorithm 11.5. Show that Stooge-sort is correct and characterize the running time, T(n), for Stooge-sort, using a recurrence equation, and use the master theorem to determine an asymptotic bound for T(n).  
     
   Algorithm StoogeSort(A, i, j):

Input: An array, A, and two indices, i and j, such that 1 ≤ i ≤ j ≤ n

Output: Subarray, A[i..j], sorted in nondecreasing order

n ← j − i +1 // The size of the subarray we are sorting

if n =2 then

if A[i] >A[j] then

Swap A[i] and A[j]

else if n>2 then

m ←n/3

StoogeSort(A, i, j − m) // Sort the ﬁrst part

StoogeSort(A, i + m, j) // Sort the last part

StoogeSort(A, i, j − m) // Sort the ﬁrst part again

return A

Algorithm 11.5: Stooge-sort.

* 1. If the subarray size is less than 3, the sorted subarray is returned instantly. If it is greater than 3 the first 2/3 of the subarray are sorted, then the last two thirds are sorted relying on the correct order from the first sort, then the first two thirds are sorted again to polish out any issues with some smaller values being in the last one third moving to the front of the second third.  
     Therefore, the equation is .

1. A-11.1 A very common problem in computer graphics is to approximate a complex shape with a bounding box. For a set, S, of n points in 2-dimensional space, the idea is to ﬁnd the smallest rectangle, R, with sides parallel to the coordinate axes that contains all the points in S. Once S is approximated by such a bounding box, we can often speed up lots of computations that involve S. For example, if R is completely obscured some object in the foreground, then we don’t need to render any of S. Likewise, if we shoot a virtual ray and it completely misses R, then it is guarantee to completely miss S. So doing comparisons with R instead of S can often save time. But this savings is wasted if we spend a lot of time constructing R; hence, it would be ideal to have a fast way of computing a bounding box, R, for a set, S, of n points in the plane. Note that the construction of R can be reduced to two instances of the problem of simultaneously ﬁnding the minimum and the maximum in a set of n numbers; namely, we need only do this for the x-coordinates in S and then for the y-coordinates in S. Therefore, design a divide-and-conquer algorithm for ﬁnding both the minimum and the maximum element of n numbers using no more than 3n/2 comparisons.
   1. Taking groups of two and placing the smaller value into a minimumSet and the larger value in a maximumSet then recursively calling the function on both minimumSet and maximum set until they are both down to one value. Both sweeps will make n/2 comparisons. Therefore, there will be n total comparisons.
2. R-12.9 Sally is hosting an Internet auction to sell n widgets. She receives m bids, each of the form “I want ki widgets for di dollars,” for i =1, 2, ..., m. Characterize her optimization problem as a knapsack problem. Under what conditions is this a 0-1 versus fractional problem?
   1. It is a fractional knapsack problem if the bids can be satisfied by a partial fulfillment. Otherwise, it is a 0-1 problem.
3. A-12.3 An American spy is deep undercover in the hostile country of Phonemia. In order not to waste scarce resources, any time he wants to send a message back home, he removes all the punctuation from his message and converts all the letters to uppercase. So, for example, to send the message,

“Abort the plan! Meet at the Dark Cabin.”

he would transmit  
ABORTTHEPLANMEETATTHEDARKCABIN   
Given such a string, S, of n uppercase letters, describe an efﬁcient way of breaking it into a sequence of valid English words. You may assume that you have a function, valid(s), which can take a character string, s, and return true if and only if s is a valid English word. What is the running time of your algorithm, assuming each call to the function, valid, runs in O(1) time?

* 1. Iterate from 0 to n using a variable, j. Then, iterate from 0 to j using a variable, i, running valid(s) for s=S[i:J]. This would result in n2 function calls and, therefore, be O(n2)