Anthony Rusignuolo

CS 600WS – Advanced Algorithms

Reza Peyrovian

Homework 4

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

1. C-7.4 Let A be a collection of objects. Describe an efﬁcient method for converting A into a set. That is, remove all duplicates from A. What is the running time of this method?
   1. Using a hash table would be one could simply use a loop to check if a value has not been added to the hash table then add it if that’s the case. It would run in O(n)
2. A-7.2 One of the tasks for an operating system is the job of scheduling computations to be performed by the processor(s) that are part of that system. A subtask that comes up in some processor scheduling problems is to solve a sequence σ of O(n) priority queue operations, where each operation in σ is either an insert(i) or removeMin(), such that i is a distinct integer in the range from 1 to n. This problem is known as the ofﬂine-min problem, since the entire sequence, σ, is given in advance. Interestingly, the ofﬂine nature of this problem gives us a faster way of answering the operations in σ than using a heap. Namely, if k is the smallest integer inserted somewhere in σ, then we know the very next removeMin() in σ will return k. Likewise, after we have matched up k and this removeMin(), and deleted both from σ, then we can repeat this argument on the operations that remain. Use this observation to design an algorithm for answering all the operations in σ in O(nα(n)) time, and thereby solve the ofﬂine-min problem.
   1. Something to do with the Ackermann function. It’s far too complicated and confusing for me to understand. If that is part of the solution, hopefully the Professor will cover that more.
3. C-8.3 Suppose we are given two n-element sorted sequences A and B that should not be viewed as sets (that is, A and B may contain duplicate entries). Describe an O(n)-time method for computing a sequence representing the set A ∪ B (with no duplicates).
   1. Create a loop that continues until both A and B are empty. Inside the loop the first values of the list will be compared and the lower one will be removed from its respective parent list and appended to the end of the union set only if its value isn’t already in the union set, this can be checked for by comparing to the last value in the set (since the two lists are sorted) instead of checking the whole set.  
        
      For example:  
      While A != [] and B != []:  
       if A[0] <= B[0]  
       x = A.dequeue()  
       else  
       x = B.dequeue()  
       if x != AUB[-1]  
       AUB.enqueue(x)
4. C-8.7 Suppose we are given a sequence S of n elements, on which a total order relation is deﬁned. Describe an efﬁcient method for determining whether there are two equal elements in S. What is the running time of your method?
   1. The list can be traversed and its values added to a hash table. If any of the values already exist in the table, then there is at least one duplicate value. This algorithm runs in O(n)
5. A-8.4 Bob has a set, A, of n nuts and a set, B, of n bolts, such that each nut has a unique matching bolt. Unfortunately, the nuts in A all look the same, and the bolts in B all look the same as well. The only comparison that Bob can make is to take a nut-bolt pair (a, b), such that a ∈ A and b ∈ B, and test if the threads of a are larger, smaller, or a perfect match with the threads of b. Describe an efﬁcient algorithm for Bob to match up all of his nuts and bolts. What is the running time of this algorithm?
   1. Start by picking a random nut. Then compare all the other nuts to the random nut and put them into two groups: one group for larger threads and one group for smaller threads. Then match the random nut to a bolt. Once the matching bolt is found compare the other bolts to it and put them into two groups, a group with smaller threads and a group with larger threads. Redo this comparison for both the larger thread and smaller thread nuts and bolts. The comparison to match the nut and bolt runs in O(n) time, the sorting runs in O(n-1) time, and the recursive comparison runs in O(2logn). Therefore the function runs in O(2logn \* (n + (n-1))) = O(nlogn)
6. C-9.5 Suppose we are given a sequence, S, of n integers in the range from 1 to n3. Give an O(n)-time method for determining whether there are two equal numbers in S.
   1. This is a similar solution to what was done for C-8.7. Simply traverse S and add its values to a hash table. If the value already exists in the table in can be concluded that there are duplicates.
7. A-9.5 Consider the election problem from the previous exercise, but now describe an algorithm running in O(n) time to determine the student numbers of every candidate that received more than n/3 votes.
   1. Use a hash table to increment the values as they repeat and checking if they’re greater than n/3. O(n)