1. (5 Points) Given T(n) = T(n-1) + 1/n, show that The Master Theorem is not applicable to this recurrence equation, hence show that T(n)=O(log n) using algebraic substitution given T(1) =1

T(n-1) is not T(n/b) thus don’t fit Master Theorem condition.

If we expand the T(n-1) using the given equation, we can get:

T(n) = T(1) + (1/n + 1/(n-1) + 1/(n-2) + … + 1/2) = 1 + ½ + … + 1/n

<= 1 + log(n) ≈ O(logn).

1. (5 Points) Using the very definition of Big-omega notation, prove that n2 +10n is Ω(n2). You must use the definition and find the constants in the definition to receive credit.

Formal definition is: For some constant c>0 and all large enough n, f(n)≥c g(n).

In this case for f(n) = n^2 + 10 \* n, if c = 1 then n^2 + 10n > n^2, then it is big omega n^2.

1. (5 Points) Is it possible to modify any sorting algorithm to become stable? If so How? If not, give reason?

It is possible by when it comes to key comparison, we can always add position as a property. For example if we are comparing a set of s1, s2,… sn, we can always pair them with index like (s1, 0) , (s2, 1) , … (sn, n-1). Thus, when it comes to the same key, we can compare the unique index to make sure the result would be stable.

1. (13 Points) Recall the **Extendable Array** Implementation from Sections 1.4.1 and 1.4.2. Now cconsider an extendable table that supports both add and remove methods. Suppose we grow the underlying array implementing the table by doubling its capacity any time we need to increase the size of this array, || and we shrink the underlying array by half any time the number of (actual) elements in the table dips below N/4, where N is the current capacity of the array. Use amortization method with cyber dollars to show that a sequence of n adds and remove methods, starting from an array with capacity N = 1, takes O (n) time. You must use amortization Technique with cyber dollars to receive any credit.

By considering the future coping we say it cost 3 cyber dollars for adding an element to the array in which 1 is adding the current element and 2 for future coping. Since the removal’s shrinking policy is exactly the opposite of adding’s double policy, we can apply the same theorem to removal. We charge 1 for adding the element and two for future removal. Thus, the worst case is simply n add operations and n remove operations, which is 3n + 3n cyber dollars in total which comes to f(n) = 6n, which is O(n).

1. (13 Points) Design an O(n lg n)-time algorithm that, given an array A of n integers and another integer x, determines whether or not there exist two (not necessarily distinct) elements in A whose sum is exactly x.

We can sort array A first using quick-sort or merge sort which takes O(n log n). Then we can start scan throw array A. For element A[i] we find if there exist A[j] such that A[j] = x – A[i]. We can use binary search with this, and each takes O (log n) for one search. We need to do a total of n search for n elements in A thus it would take O (n log n). The total running time is O(n log n) + O(n log n) which is still O(n log n).

1. (13 Points) Develop an algorithm that computes the **kth smallest** element of a set of n distinct integers in O (n + k log n) time.

We can use heapsort for this. Firstly, we construct a min-heap which takes O(n). Then each removal at heap takes O (log n) to re-self-construct the heap. For the kth smallest we need to do removal at min-heap for k times which is O (k log n). Thus the total running time is O (n + k log n).

1. (10 Points) Suppose a social network, N, contains n people, m edges, and c connected components. What is the exact number of times that each of the methods, **makeSet**, **union**, and **find,** are called in computing the connected components for N using Algorithm 7.2?

The makeSet method needs to be called for each person one time, which sums up to be n calls.

The find method needs to be called two times for each edge to return the two sets containing the two connected people for comparing and making union, and another n times for n people to output if person x belongs to connected component. As a result of (2m + n) calls.

The union method is called (n – c) times for un-connected people.

1. (13 Points) 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?

We can use quick-sort or merge-sort to sort the collection first, which takes O (n log n). Then we only need to scan through once to eliminate any duplicates like if A[i] = A [i-1], then remove A[i-1]. Thus the total running time is O(n log n) + O(n) which is O(n log n).

1. (13 Points). Suppose you have one machine and a set of jobs a1, a2,….an to process on that machine. Each job ai has a processing time ti, a profit pi, and a deadline di. The machine can process only one job at a time, and a job ai must run uninterruptedly for ti consecutive time units. If job ai completes by its deadline di, you receive a profit pi, but if it is completed after its deadline, you receive a profit of 0. Give an algorithm to find the schedule that obtains the maximum amount of profit, assuming that all processing times are integers between 1 and n. What is the running time of your algorithm?
2. (10 Points) Suppose you are given an instance of the fractional knapsack problem in which all the items have the same weight. Design an algorithm that solves the fractional knapsack problem in this case in O(n) time.