CS362 Exam I

Last Name:

First Name:

Email:

Honesty Policy:

1. This is a close-book exam. You may only use the double-sided handwritten US letter sized "cheating sheet" and a dumb calculator that you have brought to the exam. Nothing else permitted.

2. You are not allowed to talk to anyone except the professor in charge during the exam. You will be **thrown out** if caught talking to other students or peeking at other students' exams.

3. It is your responsibility to guard your work. Don't let other people copy your solutions.

Exam Instructions:

1. Please silence your cell phones.

2. Write your answers legibly.

3. There are 5 problems in this exam.

4. For algorithmic problems, do not give detailed pseudo-code. Describe the key ideas and key steps of your solutions, correctness argument, and provide running time analysis. In the case, where you are using a well-known algorithm (e.g., merge sort or linear time selection) as a subroutine without modification, just cite the well-known algorithm and its running time. Do not repeat the steps of the well-known algorithms.

5. Don't spend too much time on any single problem. If you get stuck, move on to something else and get back later.

6. Don't leave multiple solutions for the same problem. You may be graded based on the one that's wrong.

7. Good luck and enjoy the exam!

1. Let V be an **unsorted** set of n points that are on the circumpherence of a circle. Design an $O(n \log n)$ algorithm that computes from V a linked list containing all vertices, sorted in counter clockwise order.

   Solution:

   Assume the center of the circle is at the origin. (If not, pick 3 arbitrary points, and caculate the center of the circle, which will take constant time.)

   Calculate the polar angle of each point in V with respect to the center. This process will take $O(n)$ time.

   Sort based on the polar angle. This process will take $O(n \log n)$ time.

2. Develop an efficient algorithm to reverse the words in the sentence For example, the sentence, “this is computer algorithms” shoud become “algorithms computer is this”. Optimize the time and space of your algorithm.

   Solution: Use a stack.

3. Suppose you are asked to design a data structure that support the following operations:

   (a) $Insert(x)$: insert a key x into the data structure only x is not already there.

   (b) $Delete(x)$: delete a key x from the data structure if it is there.

   (c) $Find\_Next(x, k)$: find the $k-$th smallest key $> x$ if it exists.

   All these operations should take $O(\log n)$ time in the **worst case**, where n is the number of keys in the data structure.

   Solution:

   For efficient searching, insertion and deletion, we have to use a BST. However, the BST doesn't support $Find\_Next(x, k)$. We can overcome this by maintaining for each node, the number of nodes in its sub-tree.

4. Let A be an array of n arbitrary and **distinct** numbers. A has the following property: If we imagine B as being sorted version of A, then any element that is at position i in array A would, in B, be at a position j such that $|i-j| \leq k$. In other words, each element in A is not farther than k positions away from where it belongs in the sorted version of A. Suppose you are given such an array A, and you are told that A has this property for a particular value $k << n$ (that value of k is also given to you). Design an $O\left(n \log k\right)$ time algorithm for sorting A.

   Solution: Assume that we want to sort A increasingly.

   Create a min-heap of size $k+1$. Insert the first $k+1$ numbers, i.e., $A[1..k+1]$ in to the heap. This will take $O(klogk)$ time.

   In each iteration, extract the min from the heap. Insert to the heap the next element from A that's not in the heap yet.

5. Answer the following questions regarding the integer paritition problems for a given set S of n positive integers $\{k_1, k_2, ..., k_n\}$.

   (a) The 2-partition problem asks if there is a partition of S into two **disjoint** subsets S_1 and S_2, $S_1 \cup S_2 = S$ and such that

   $$\sum_{k_i \in S_1} k_i = \sum_{k_j \in S_2} k_j$$

   Design an efficient algorithm for solving the 2-paritition problem.

   (b) The 3-paritition problem asks if there is a partition of S into three **disjoint** subsets S_1, S_2, and S_3, $S_1 \cup S_2 \cup S_3 = S$ and such that

   $$\sum_{k_i \in S_1} k_i = \sum_{k_j \in S_2} k_j = \sum_{k_l \in S_3} k_l$$

   Design an efficient algorithm for solving the 3-paritition problem.

Solution:

We associate each item a cost equal to its size. 2-parition can be viewed as the knapsack problem, where the size of the knapsack if half of the sum of the all the numbes.

Similarly, the 3-paritition problem is a 2 Knapsack problem (as in hw2), where each knapsack has a size a third of the sum of all the numbers.