CS 513: Design and Analysis of Algorithms

Midterm Exam

December 3, 2025

1. **Short questions** Answer the following questions. Simply give the **final** answer.

   (a) G is an undirected weighted graph with all weights to be positive integers. If all edges have distinct weights then the minimum spanning tree is unique. If otherwise, i.e., some edges have the same weight, the minimum spanning tree is not unique. Answer True or False. (4pt)
   False. The second part is wrong.

   (b) Given an array $A[1..n]$ of n distinct integers, find an algorithm to compute a local minimum. That is, $A[i-1] > A[i]$, $A[i+1] > A[i]$. Write in one sentence the algorithm you use and its running time. (4pt)
   Binary search. $O(\log n)$

   (c) In a graph with distinctive weights (no two edges have the same weight), the edge of the minimum weight on a cycle is on the minimum spanning tree. True or false? (4pt)
   False

   (d) Solve recurrences $T(n) = T(n-2) + n\sqrt{n}$ with big Θ notation. (4pt)
   $\Theta(n^2\sqrt{n})$.

   (e) Solve recurrences $T(n) = T(n/3) + \sqrt{n}$ with big Θ notation. (4pt)
   $\Theta(\sqrt{n})$.

2. **Finding Median:** Given $A[n]$ and $B[n]$ as two sorted array in increasing order. Suppose all numbers are distinct. Find the median of the $2n$ numbers in $O(\log n)$ time. Write down your algorithm, show correctness and prove the running time. (20pts)

   Compare $A[n/2]$ with $B[n/2]$. If $A[n/2] \leq B[n/2]$, then we argue that any element in $A[1..n/2]$ or $B[n/2+1..n]$ cannot be the median. Thus we can safely throw them away. Since we are throwing away equal number of points on both side of the true median this does not change the value of the true median. Thus we are left with two arrays each of size only $n/2$. Continue and in $O(\log n)$ steps we can find the true median.

3. Consider the following problem. The input is a collection $A = \{a_1, \cdots, a_n\}$ of n points on the real line. The problem is to find a minimum cardinality collection S of unit intervals that cover every point in A. Another way to think about this same problem is the following. You know a collection of times (A) that trains will arrive at a station. When a train arrives there must be someone manning the station. Due to union rules, each employee can work at most one hour at the station. The problem is to find a scheduling of employees that covers all the times in A and uses the fewest number of employees.

(a) Prove or disprove that the following algorithm correctly solves this problem. Let I be the interval that covers the most number of points in A. Add I to the solution set S. Then recursively continue on the points in A not covered by I. (10pts)

(b) Prove or disprove that the following algorithm correctly solves this problem. Let a_j be the smallest (leftmost) point in A. Add the interval $I = (a_j, a_j + 1)$ to the solution set S. Then recursively continue on the points in A not covered by I. (10pts)

Answer:

(a) Not correct. A counterexample is shown below:

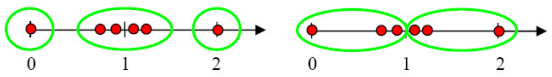

Figure 1: A counterexample

(b) Correct.
If we use I_i to represent the ith interval of this algorithm from left to right and we use I'_i to represent the ith interval of other algorithms. We use mathematical induction to prove one claim: the right endpoint of I_i is greater or equal to that of I'_i. That is:

$$right(I_i) \geqslant right(I'_i)$$

The basis step is trivial because of the property of the greedy algorithm. The inductive hypothesis is: when $i = k$, we have $right(I_k) \geqslant right(I'_k)$. Suppose I_{k+1} covers point a_j which is the leftmost uncovered point. If I'_{k+1} doesn't cover a_j, it is obvious that $right(I_{k+1}) \geqslant right(I'_{k+1})$. Otherwise if I'_{k+1} covers a_j, because the left endpoint of I_{k+1} is at a_j, it is in fact the rightmost interval that covers a_j, and we can also conclude that $right(I_{k+1}) \geqslant right(I'_{k+1})$. We have proved that when $i = k + 1$, $right(I_{k+1}) \geqslant right(I'_{k+1})$, therefore, the claim holds.
Based on the claim above, we can prove that this greedy algorithm uses fewest intervals to cover the whole set of points. If there exits one algorithm that uses fewer intervals, for example m intervals, but this greedy algorithm uses n intervals ($m < n$). This means $right(I_m) < right(I'_m)$, and it is a contradiction. So this algorithm is optimal.

4. Alice wants to throw a party and she is trying to decide who to invite. She has n people to choose from, and she knows which pairs of these people know each other. She wants to invite as many people as possible, subject to two constraints:

(a) For each guest, there should be at least five other guests that they already know.

(b) For each guest, there should be at least five other guests that they don't already know.

Design and analyze an algorithm that computes the largest possible number of guests Alice can invite, given a list of n people and the list of pairs who know each other. (20pts)

Solution: *We first state two observations. We keep a set S as the set of possible candidates to invite. S initially includes all the n people.*

(a) if a guest p knows four or fewer other guests in S, p cannot be invited regardless.

(b) if a guest p knows all but four or fewer others in S, p again cannot be invited, as the optimal list is a subset of S and this has a contradiction with the constraints.

Therefore we remove a guest if its degree is either too high or too low. We repeat this process, until no more guests are removed. That is the final list.

To show correctness, we need to argue that all guests who were removed have to be removed. This can be done by an inductive proof. Details are skipped here.

For running time, we keep a graph and for each vertex keep the degree counter. Keep two lists, one with the "low degree" vertices called L – vertices with degree smaller than five; and one with the "high degree" ones called H – vertices with degree greater than k-5, with k as the current number of vertices. In each iteration, as long as $L \cup H$ is not empty, we take one vertex p from $L \cup H$, remove it from S, update the degree of the vertices that are neighbors of p, and possibly include them in L and H. In each iteration we remove one vertex and the update cost is at most $O(n)$. The running time is $O(n^2)$.

5. Suppose you have n final projects and a total of $H > n$ hours. You want to decide how to partition your time. Ideally you want to maximize the average grade on the n projects. Suppose there is a function f_i for project i: $f_i(h)$ is the grade you will get if you spend h hours on the project i. Suppose the maximum grade is g and f_i is nondecreasing. The problem is to decide how many hours to spend on each project (in integer values only) such that the average grade is the highest. Design an algorithm with running time being polynomial in n, g and H. (20pts)

   Solution: *Define $G(i, h)$ as the maximum total grade for project $1, 2, \cdots, i$ with a total of h hours. The recursive structure is*

$$G(i, h) = \max_{x=1}^{h} \{G(i-1, x) + f_i(h-x)\}$$

   . The original problem is $G(n, H)$. The running time is $O(nH)$.

6. (Optional problem for extra credits) **Smart Cashiers.** Suppose we have a country with k types of bills, with value $v_1, v_2, \cdots, v_k$, all v_i's are distinct integers, k is a constant. Given a total dollar amount N, a cashier will need to come up with a set of bills that sum up to N. Ideally the smart cashier would like to use a minimum number of bills.

   (a) If the values of the bills are power of two, i.e., $v_i = 2^{i-1}$, find a greedy algorithm for the smart cashier, for any dollar value N. Write down your algorithm, show correctness and prove the running time. (5pts)

   For the algorithm, we first choose the maximum number of bills of values v_k so that the sum does not exceed N. Then we choose the maximum number of bills of value v_{k-1} and so on.

   To show that this is correct, we order the bills in decreasing value and each solution can be represented by a vector. The algorithm we generate is lexicographically the largest. Now we prove the correctness by induction.

   (b) For general values of $v_1, v_2, \cdots, v_k$, show that the same greedy algorithm you developped for (a) does not always give the optimal answer. For this you need to show a counter example. (5pts)

   Suppose the bills are of value $1, 4, 5, 7$. Now $N = 9$. The optimal is to take a bill of value 4 and a bill of value 5. But the above algorithm gives three bills of $7, 1, 1$.

(c) For general values of $v_1, v_2, \cdots, v_k$, solve the problem by dynamic programming. Write down your algorithm, show correctness and prove the running time. (5pts)

Define by $S(x)$ the minimum number of bills needed to get a value of x.

$$S(x) = \min_i \{S(x - v_i) + 1\}$$

if $x > v_i$. $S(v_i) = 1$ for all i. The running time is $O(nk)$, and $O(n)$ since k is a constant.