TOBB ETÜ, BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BİL 133 Combinatorics and Graph Theory

HOMEWORK 7 (30 Points)

Due Date: July 25, 2017

1 [4 Points] Number of Square Integer Matrices

We say that a matrix is a **square matrix** if it has equal number of rows and columns. We say that a matrix is an **integer matrix** if all its entries are integers.
What can you say about the cardinality of the set of square integer matrices? Is it countable or uncountable?

2 [4 Points] An Equivalence Relation

We define the $\backsim$ relation over the collection of sets as follows. For any two sets S_1 and S_2, we say that $S_1 \backsim S_2$ if and only if there is a bijection $f : S_1 \to S_2$. Prove that $\backsim$ is an equivalence relation.
(Recall that a relation is an equivalence relation if it is reflexive, symmetric and transitive.)

3 [4 Points] Space Considerations for Merge Sort

We have seen several sorting algorithms in class. Merge Sort algorithm, which uses the **divide and conquer** approach, is not an **in place** algorithm, i.e., the amount of extra space it needs is not bounded by a constant.

[4 **Points**] In this problem, you are asked to calculate the amount of extra space used by Merge Sort.

4 [14 Points] Efficient Algorithm Design for Special Cases of Subset Sum

Subset Sum is a classical problem in computer science. In this problem, you are given a multiset (array) A of integers, and an integer k. The problem is to determine whether there exists a subset S of the integers in A such that the sum of the integers in S is exactly k.

[0 **Points**] Try to come up with an efficient (polynomial-time) algorithm for the **Subset Sum** problem. Never mind, if you could not, since nobody achieved to do that yet! This problem is **NP**-complete, and does not admit an efficient algorithm unless **P** = **NP**.

In this question, we will examine special cases of the **Subset Sum** problem. Specifically we will consider the cases where $|S|$ is bounded.

Consider the special case of the **Subset Sum** problem, where $|S| = 1$. In this special case, we are given an array A of integers, and an integer k, and asked to decide whether there exists a subset $|S| = 1$ of integers in A such that the sum of the integers in S is exactly k. Since $|S| = 1$, S is composed of just one integer. Thus, the problem is equivalent to checking whether k is one of the integers in A or not. This can be done by using the **linear search** algorithm in time **O(n)**.

[2 **Points**] Give an $O(n^2)$-algorithm for the special case of the Subset Sum problem, where $|S| = 2$, i.e., your algorithm should determine whether there exist integers i and j in A such that $i + j = k$.

Notice that you can immediately obtain an algorithm for the special case of the Subset Sum problem, where $|S| \leq 2$, that runs in $O(n^2)$-time. The algorithm should first make a call to the linear search algorithm, and then a call to the algorithm you designed. If any of them returns true, it should return true. This algorithm will be correct only if the algorithm you designed is correct. Thus, you need to prove your algorithm before we proceed.

[3 **Points**] Prove your $O(n^2)$-algorithm for the special case of the Subset Sum, where $|S| = 2$. You need to show that your algorithm terminates and returns a correct answer for all possible instances of the problem.

[3 **Points**] Let us now generalize your algorithm to handle the cases, where $|S|$ is larger. Show that for any integer i, you can give an $O(n^i)$-algorithm for the special case of the Subset Sum problem, where $|S| = i$.

Since, for any i, you can give a $O(n^i)$ algorithm for the special case of the Subset Sum problem with $|S| = i$, you can immediately give an $O(n^i)$-algorithm for the special case of the Subset Sum problem with $|S| \leq i$. This algorithm will first run the $O(n)$ linear search algorithm, then $O(n^2)$ algorithm you designed for the case $|S| = 2$, then the $O(n^3)$ algorithm for the case $|S| = 3$, ... , and the $O(n^i)$-algorithm for the special case $|S| = i$, and return true, if any of these algorithms returns true. Notice that the run-time of this algorithm is $\sum_{j=1}^{i} O(n^j) = O(n^i)$. Thus, you have proven the Theorem stated below.

Theorem 1. *For any integer i, there exists an $O(n^i)$-algorithm that decides the special case of the Subset Sum problem, where $|S|$ is bounded above by i.*

Let us now elaborate the special case, where $|S| = 2$. An electrical engineer trying to get a job in computer industry could as well come up with your $O(n^2)$-algorithm. Competitive computer scientists should be able to do a lot better than that.

[3 **Points**] Give an $O(n \cdot \lg n)$-algorithm for the special case of the Subset Sum problem, where $|S| = 2$.

[3 **Points**] I believe that you have already proven the termination and correctness of your $O(n \cdot \lg n)$-algorithm. If you haven't, it is time to do it for another 3 points.

5 [4 Points] O vs o

You are given two *similar looking* propositions below. However, one of them is true, and the other one is false. You are asked to prove the true one, and disprove the false one.
Let $f(n)$ and $g(n)$ be two functions defined over the set of positive integers such that $\lg f(n)$ and $\lg g(n)$ are strictly increasing.

- [2 **Points**] Prove that if $f(n) = O(g(n))$, then $\lg(f(n)) = O(\lg(g(n)))$.
- [2 **Points**] Disprove that if $f(n) = o(g(n))$, then $\lg(f(n)) = o(\lg(g(n)))$.