TOBB ETÜ, BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BİL 331/531 Design and Analysis of Algorithms

HOMEWORK 1 (30 Points)

Due Date: January 16, 2018

1 [10 POINTS] PROPERLY ORDERING A COUNTABLE SET

We can state the *Fermat's conjecture* as a decision problem as follows:

Is there a triple of positive integers $(a,b,c) \in Z_+^3$ such that $\forall k \geq 3$ we have $a^k + b^k = c^k$?

As you all know, this is a computationally trivial problem that can be solved by a one line algorithm. Even though, there isn't any triple of positive integers that satisfy the desired equality for every integer k; it may be the case that for some specific values of k, the equality is satisfied. If that is so, can you find which triple of positive integers do that?

More precisely, you are asked to give an algorithm that takes an integer k as a parameter. You can assume that there exists positive integers a, b, and c such that $a^k + b^k = c^k$ for the integer k presented to your algorithm as input. Your algorithm should return such a triple (a,b,c).

You should first give a pseudocode. Then explain your pseudocode. Then you should prove that your algorithm terminates with the correct output.

2 [5 POINTS] ANALYZING AN ALGORITHM REQUIRES FORMAL TRAINING

One of your colleagues came up with a recursive algorithm to solve a computational problem and he wants to analyze the time complexity of his algorithm. He defines $T(n)$ as the running time of his algorithm on an input of size n, and tries to obtain a recurrence equation

for $T(n)$. Your professor at college was very obsessed with the formality of the arguments made, and you were often getting angry at him because of that. However, it seems that you have somehow inherited this undesirable habit from him and can't stand when people make vague arguments.

What would you say to your colleague?

3 [15 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 (which will be defined later), 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)**.

[3 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 an $O(n^i)$-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 running 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 4 points.