Greedy and Randomized Algorithms

1. Suppose you are running a business and, on day one, you get a set of n jobs $1, 2, \ldots, n$. For each job, you get paid \$100 minus \$1 per day it takes you to complete the job. You can only work on one job at a time, and once you start a job, you must complete it. You know how many days each job will take. You must complete all the jobs. You want to work on the jobs in the order that will maximize your profit.

   Formally, we say job j takes t_j days to complete. An ordering of the jobs will define a completion time C_j for each job j. For example, if the first job in the ordering is j, then the completion time of job j will be $C_j = t_j$. The completion time of any other job j' is then the completion time of the job before it in the ordering plus $t_{j'}$. You want to maximize the profit $100n - \sum_{j=1}^{n} C_j$, which is equivalent to minimizing $\sum_{j=1}^{n} C_j$.

   (a) Design a greedy algorithm to find an ordering (and so define completion times) that minimizes $\sum_{j=1}^{n} C_j$. Give pseudocode for this algorithm.

   (b) Prove that your algorithm correctly minimizes $\sum_{j=1}^{n} C_j$.

   (c) What is the running time of your algorithm?

2. You're walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest of k gold bricks of different (integral) weight. Your knapsack can only carry up to weight n before it breaks apart. You want to put as much in it as possible without going over, but you cannot break the gold bricks up.

   (a) Suppose that the k gold bricks have the weights $1, 2, 4, 8, \ldots, 2^{k-1}$, for $k \geq 1$. Describe a greedy algorithm that fills the knapsack as much as possible without going over.

   (b) Prove that your algorithm is correct.

   (c) Give an example, including a multiset of different weight values of bricks and total weight n that the knapsack can carry up, for which the greedy algorithm does not yield an optimal solution.

3. Consider the following randomized algorithm for generating biased random bits. The subroutine FAIRCOIN returns either 0 or 1 with equal probability; the random bits returned by FAIRCOIN are mutually independent.

```
ONEINTHREE():
    if FAIRCOIN() = 0
        return 0
    else
        return 1− ONEINTHREE()
```

   (a) Prove that ONEINTHREE returns 1 with probability 1/3.

   (b) What is the exact expected number of times that this algorithm calls FAIRCOIN?

   (c) Now suppose you are given a subroutine ONEINTHREE that generates a random bit that is equal to 1 with probability 1/3. Describe a FAIRCOIN algorithm that returns either 0 or 1 with equal probability, using ONEINTHREE as your only source of randomness.

4. Every year, the IRS receives n forms with personal tax returns. The IRS, of course, can not verify all n forms, but they can check some of them. Describe an algorithm that decides whether the number of incorrect tax forms is larger than $2\epsilon n$ or smaller than ϵn, where ϵ is a prespecified constant between 0 and 1. Your algorithm should either return the string "smaller" or "larger".

The algorithm is considered to be incorrect if it declares that the number of incorrect forms is smaller than ϵn, but it is in fact larger than $2\epsilon n$. Similarly, the algorithm is considered to be incorrect if it claims that the number of incorrect forms is larger than $2\epsilon n$, but it is in fact smaller than ϵn. If the number of incorrect forms is between ϵn and $2\epsilon n$, then returning either "smaller" or "larger" is acceptable.

Your algorithm should output a correct result with probability $\geq 1 - 1/n^{10}$. Assuming that verifying the correctness of a single tax form takes $O(1)$ time, what is the running time of your algorithm? (Hint: Use the Chernoff inequality.)