COMP 446 / 546 Final Exam Sample 1 Name: _____

- By submitting this assignment, you agree to fully comply with Koç University Student Code of Conduct, and accept any punishment as directed on the course syllabus in case of failure to comply. If you do not submit this on time, you will receive no credit.
- Closed notes, closed book. If you are stuck with a question, skip it and try again later.
- Show your work clearly, in order, and with proper **explanation to get full points**.
- You have 3 hours to take this 110 pts final. Good luck!

1. (15pts) **True/False/Other (explanation necessary)**

a) (5 pts) Every problem in NP can be solved in exponential time.

b) (5 pts) No adversary can force randomized quicksort to run in quadratic time (n^2).

c) (5 pts) Given a graph with edge capacities 3k for some positive integer k (a different k may be used for each edge). The maximum flow must also be of the form 3t for some positive integer t.

2. (10 pts) **Divide and Conquer**
Assume you are solving a problem with input size n using divide-and-conquer strategy. By performing n^2 operations, you can create 10 subproblems of size n/3, and with n^2 **more** operations, you can combine the answers to those subproblems into a solution to your original problem of size n.

a) (5 pts) Write down the recursion for the runtime of your algorithm, and solve it.

b) (5 pts) Suppose you can reduce the number of subproblems to 9 with only n^2 more operations in the divide and merge steps. How does this improve your overall complexity? (Give the new bound).

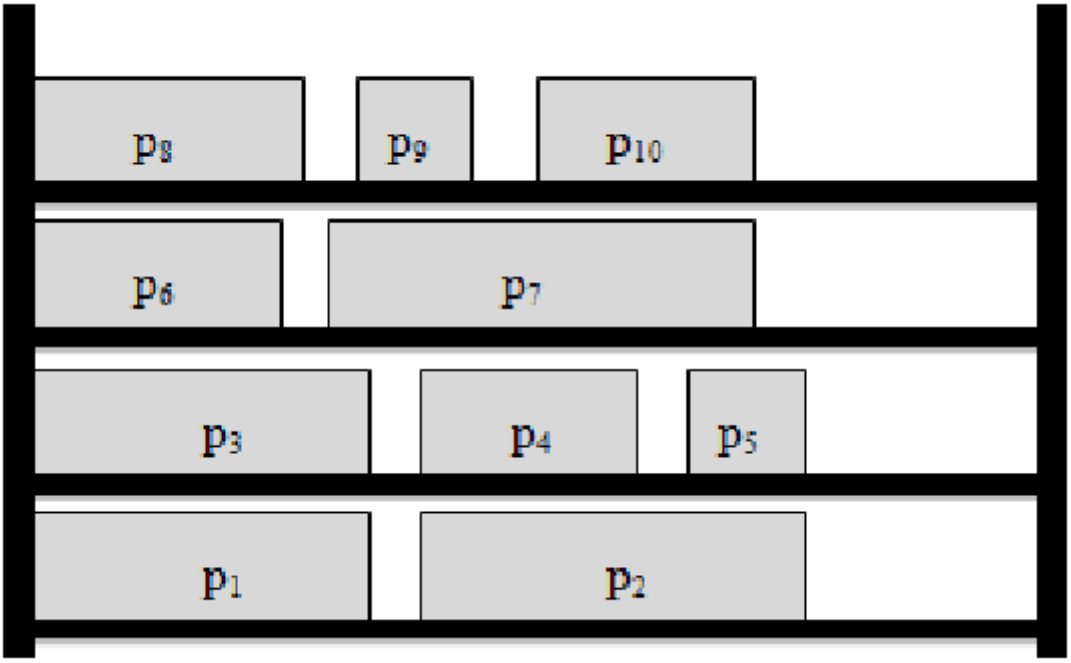
3. (15 pts) **NP-Hardness**

Given a function $f : [0, 1]^n \rightarrow R$ which can be computed in polynomial time, show that the problem of determining whether $\exists x_1 \ldots x_n$ such that $f(x_1, \ldots, x_n) > 0$ is NP-complete. (You must first prove that this problem is in NP by showing a polynomial-size certificate for YES instances, and a polynomial time algorithm that verifies the YES instances using the given certificate. Your verification algorithm must return false for NO instances regardless of the certificate. Then, you must provide a proper reduction from a known NP-complete problem. You must show that your reduction runs in polynomial time in its input length.)

4. (30 pts) **Greedy Algorithms and Dynamic Programming**

Buy-N-Large has to sell quite a lot of small useless goods, and to maximize throughput in their warehouses, they use giant rotating shelves. As item p_i arrives, they place it on the top shelf, exactly 1 inch to the right of item p_{i-1} , or, if the shelf is empty, aligned all the way to left side of the shelf. Each item p_i is exactly w_i inches wide and each shelf is W inches wide. When they are done putting items on a shelf (which may happen at any time), they rotate the shelf down and begin stocking the shelf above. No item may hang of the edges or the shelves will jam when they reach the un-stocking robots at the bottom of the shelf device (see diagram). At some point in time, the top shelf is empty. Call the next item to arrive p_1 . In order to minimize the motion by the robots that stock and unstock the shelves, we want to stock items p_1 . . . p_P in such a way that **the sum of squares of the distance from the rightmost element of a shelf to the right edge of the shelf is minimized**.

Give an algorithm that decides which items go on which shelves (which is equivalent to deciding between which items to start stocking the next shelf) and minimizes this value. Demonstrate that it is correct.



[**Hint:** The solution is either a greedy algorithm or a dynamic programming approach. In both cases, prove that optimal substructure property exists. If you think a greedy approach should be used, prove also that the greedy choice property holds. If you think a dynamic programming approach should be used, explain the order of computing the subproblems.

In both cases, explain your algorithm clearly, although you do **not** need to give an explicit pseudocode. Analyze the runtime of your approach (ideally, it should not be more than $O(n^3)$ with preprocessing if necessary).]

5. (20 pts) **Randomized Algorithms**
Given an array A of n numbers, we call a number x **the frequent element** of A if it occurs at least 0.7n times in A (i.e., for at least 70% of values for $1 \leq i \leq n$ we have A[i] = x). Note that not all arrays have frequent elements and each array can have at most one frequent element. **The mode** of A is the value that occurs most frequently in A. Also recall the terminology of two approaches to using randomness in developing algorithms, both using random bits rather than relying on any distribution of the data. A Monte Carlo method is one that runs in a given time, but has some probability of failure. A Las Vegas algorithm always returns the correct answer, but has some probability of taking "longer than expected". For these algorithms, you do not need to provide a fully-working pseudocode. Instead, you can coarsely describe the algorithm.

a) (5 pts) Give an algorithm that finds the mode of A in time O(n log n).

b) (5 pts) Given a number x, describe a very efficient O(n) time algorithm that checks whether x is the frequent element of A.

c) (5 pts) Give an O(n) time deterministic algorithm that finds the frequent element of A or reports that none exists. Alternatively, you can give a Las Vegas algorithm that runs in expected O(n) time.

d) (5 pts) Give a very efficient Monte Carlo algorithm that finds the frequent element of A (or reports there is none). State the probability of error. The probability of error should be at most 0.25 (and you must show this).
[**Hint:** Start your algorithm by taking 3 random elements.]

6. (10 pts) **Linear Programming**

Suppose there are two warehouses, from which supplies are to be shipped to several destinations. We wish to find the best way to satisfy all the demands for the warehouse goods at least shipping cost. Specifically, the 2 Warehouse Multi-Demand Problem (2-WDP) is defined as follows.

Input: (I_W, I_Y, d_1,…, d_n, CW_1, …, CW_n, CY_1, …, CY_n)
A positive integer I_W, representing the inventory of warehouse W.
A positive integer I_Y , representing the inventory of warehouse Y.
A sequence (d_1, d_2, ... , d_n) of positive integers, where d_i is the demand at destination i. The demands can be met: we have Sum(i from1 to n) $d_i = I_W + I_Y$.
A list of positive integers CW_i representing the cost of shipping one unit from W to destination i, for $1 \le i \le n$;
A list of positive integer CY_i representing the cost of shipping one unit from Y to destination i, for $1 \le i \le n$.

Output: (w_1, …, w_n, y_1, …, y_n)
A sequence of nonnegative integers w_i and y_i , for $1 \le i \le n$, where w_i is the amount to be shipped from W to destination i, and y_i is the amount to be shipped from Y to destination i.

Create a Linear Program that minimizes the total shipping cost such that each destination receives the amount requested and each warehouse serves exactly its inventory.

7. (10 pts) **Amortized Analysis**

```
Spiral(n)
        if n <= 1 then return
        drive north n miles
        drive west n miles
        drive south n miles
        drive east n-1 miles
        drive north 1 mile
        run Spiral(n-2)
```

Let M(n) be the total number of miles driven when `Spiral(n)` is called.

a) (2 pts) M(1) = ?

b) (2 pts) M(2) = ?

c) (4 pts) M(n) = ? recursively

d) (2 pts) M(n) = O(?)