Worth: 5%

1. [20 marks]
We are given a list of $n > 0$ objects, say $X = (x_1, x_2, ..., x_n)$. The only operation we can do on these objects is equality testing (therefore we can not sort it). We must find all objects which occur in X strictly more than $n/3$ times. (We define $|X|$ to be this n.)

   (a) [3 points]
   Define M to be the maximum number of different objects (i.e., x_i and x_j with $x_i \neq x_j$) that can each appear in X strictly more often than $|X|/3$ times. (Note the minimum number of such frequent objects is zero.) Give the value of M and prove (carefully) that it is correct.

   (b) [12 points]
   Design a divide-and-conquer algorithm to solve the above problem. Present your algorithm as a pseudocode, and prove its correctness.

   (c) [5 points]
   Compute the complexity of your algorithm.

2. [20 marks]
Consider the problem of choosing cell phone tower locations on a long straight road. We are given a list of distances along the road at which there are customers' houses, say $\{x_k\}_{k=1}^{K}$. No towers have yet been built (so no customers currently have service). However, a survey of the road has provided a set of J possible tower locations, $\{t_j\}_{j=1}^{J}$, where these potential tower locations are also measured in terms of the distance along the road. (These indexed lists of house and tower locations might be in any order. You can assume that all these distances are integers.)

   Each customer will get service (at home) if and only if they are within a range $R > 0$ of at least one cell phone tower that gets built, that is, the house at x_k will have service iff there exists a tower that has been built at some t_j with $|x_k - t_j| \leq R$.

   You can assume that if all the towers were built then every home would get service. However, such a solution could be overly expensive for the phone company. How can we minimize the number of towers that need to be built but still provide service at every house? Note that a suitable solution may still leave some parts of the road without cell service, even though each customers' house will have service.

   (a) [7 points]
   Write a greedy algorithm for choosing the minimum number M of cell phone towers required, along with a suitable set of locations, say $T = \{t_{j(m)}\}_{m=1}^{M}$, such that every house has service. Present your algorithm as a pseudocode.

   (b) [10 points]
   Prove the correctness of your algorithm.

   (c) [3 points]
   Compute the complexity of your algorithm.

3. [20 marks]
A waiter has n tables waiting to be served. The service time required by each table is known in advance: it is $t(i)$ minutes for table i. The waiter must serve each table one at a time, and once he starts serving a table he must completely finish serving it before starting to serve the next one.

The waiter wishes to serve the customers in an order that minimizes the total waiting time:

$$T = \sum_{i=1}^{n} (\text{time spent waiting by table } i)$$

His idea is to serve the tables in increasing order of $t(i)$. We can assume that the tables are pre-sorted in this order.

(a) [5 points]
Explain why the total waiting time can be written as $T = \sum_{i=1}^{n} (n-i)t(i)$.

(b) [10 points]
Give a proof that the waiter's algorithm is always optimal. You may assume the above formula is true, even if you weren't able to explain it.

(c) [5 points]
Compute the complexity of the waiter's algorithm.

4. [20 marks]
Suppose we are given a list of n integers, say $X = (x_1, \ldots, x_n)$. The problem is to find a longest subsequence of X, say $S = (x_{j_1}, x_{j_2}, \ldots, x_{j_K})$, where $|x_{j_{k+1}} - x_{j_k}| = 3$ for each $k = 1, 2, \ldots, K-1$, and also the length of such a sequence. Note that for S to be a ***subsequence*** of X, we must have $1 \le j_1 < j_2 < \cdots < j_K \le n$, i.e., the elements of S must be chosen in the same order as they appear in X.

For example, if $X = (1, 5, 3, 4, 2, 5, -1, 2)$, then $S = (1, 4)$ and $S = (5, 2, 5)$ are possible subsequences, but $S = (5, 2, -1, 2)$ is the longest subsequence with the desired property.

(a) [15 points]
Give a detailed dynamic programming algorithm to solve this problem. Follow the steps outlined in class, and include a brief (but convincing) argument that your algorithm is correct.

(b) [5 points]
What is the worst-case running time of your algorithm? Justify briefly. (For full credit, your algorithm must use at most $O(n)$ space and run in $O(n^2)$ time.)

5. [20 marks]
Suppose we are given a $n \times n$ checkerboard with an integer-valued score written on each square, say $w(i, j)$, for $1 \le i, j \le n$. We are also given n pebbles. We obtain the score of a square only when we place a pebble on it, and we wish to maximize the sum of these scores. The pebbles must be placed according to the following two rules:

(a) Any column can contain either zero or one pebble, not more, while rows can contain any number of pebbles (from zero up to n).

(b) If any two pebbles are placed on neighbouring columns, say columns j and $j+1$, with one pebble at (i, j), then the only possible locations for the other pebble is at either $(i-1, j+1)$ or $(i+1, j+1)$ (assuming, of course, that these later squares are still within the board).

A pebble placement that satisfies these rules is said to be feasible.

For example, given the board in the figure, an optimal feasible placement is to put pebbles on rows and columns $(2, 1)$, $(1, 2)$ and $(2, 4)$, for a score of $6 + 10 + 10 = 26$. Moreover, 26 is the maximum possible score for this board.

7	10	-5	4
6	2	3	10
5	-1	-5	-1
-1	7	-1	-7

(a) [15 points]
Give a detailed dynamic programming algorithm to solve this problem. Follow the steps outlined in class, and include a brief (but convincing) argument that your algorithm is correct.

(b) [5 points]
What is the worst-case running time of your algorithm? Justify briefly. (For full credit, your algorithm must use at most $O(n^2)$ space and run in $O(n^3)$ time.)