1. (40 points) Consider the following problem. The input consists of \( n \) skiers with heights \( p_1, \ldots, p_n \), and \( n \) skies with heights \( s_1, \ldots, s_n \). The problem is to assign each skier a ski to minimize the average difference between the height of a skier and his/her assigned ski. That is, if the \( i^{th} \) skier is given the \( \alpha(i)^{th} \) ski, then you want to minimize:

\[
\frac{1}{n} \sum_{i=1}^{n} | p_i - s_{\alpha(i)} |
\]

For each of the following two algorithms, either prove that the algorithm is incorrect, or prove that it is correct using an exchange argument.

(a) **Greedy Algorithm A**: Consider the following greedy algorithm. Give the shortest skier the shortest ski, give the second shortest skier the second shortest ski, give the third shortest skier the third shortest ski, and in general give the \( k^{th} \) shortest ski the \( k^{th} \) shortest ski.

(b) **Greedy Algorithm B**: Find the skier and ski whose height difference is minimized. Assign this skier this ski. Repeat the process until every skier has a ski.

2. (20 points) We consider the following Minimum Edit Distance Problem. The input to this problem is a pair of strings \( A = a_1 \ldots a_m \) and \( B = b_1 \ldots b_n \). The goal is to convert \( A \) into \( B \) as cheaply as possible. The conversion rules are as follows. For a cost of 2 you can delete any letter. For a cost of 3 you can insert a letter in any position. For a cost of 4 you can replace any letter by any other letter. The output for the problem should be the cheapest cost that will allow \( A \) to be converted into \( B \).

For example, you can convert \( A = gatgag \) to \( B = gtcagc \) via the following sequence: \( gatgag \) at a cost of 3 can be converted to \( gatgagc \), which at cost of 2 can be converted to \( gtgagc \), which at cost of 4 can be converted to \( gtcagc \). Thus the total cost for this conversion would be 9. Under the assumption that this is the cheapest possible conversion, then the output for this problem would be the integer 9.

(a) Give pseudo-code for the most natural recursive algorithm for this problem.

(b) Explain why the running time of this recursive algorithm can be exponential in the input size.

(c) Give pseudo-code for an iterative, array based, dynamic programming algorithm whose running time is \( O(mn) \).

(d) Show the value of the first three rows of the array (not counting initialization rows) that this algorithm would produce for the above example input. To state this is a different way, show the table that the algorithm would produce if the input was \( A = gat \) and \( B = gtcagc \).
3. (20 points) Consider the following problem. The input to this problem is two sequences
$T = t_1, \ldots, t_n$ and $P = p_1, \ldots, p_k$ such that $k \leq n$, and a positive integer profit $c_i$ associated
with each $t_i$. The output is a subsequence of of $T$ that matches $P$ with maximum aggregate
profit. That is, the sequence $i_1 < \ldots < i_k$ such that for all $j$, $1 \leq j \leq k$, we have $t_{i_j} = p_j$ and
$\sum_{j=1}^{k} c_{i_j}$ is maximized.

So for example, if $n = 5$, $T = XYXXY$, $k = 2$, $P = XY$, $c_1 = c_2 = 2$, $c_3 = 7$, $c_4 = 1$ and
$c_5 = 1$, then the optimal solution is to pick the second $X$ in $T$ (so $i_1 = 3$) and the second $Y$
in $T$ (so $i_2 = 5$) for a profit of $7 + 1 = 8$. So the output for this instance would be 3, 5.

(a) Consider developing a polynomial-time dynamic programming algorithm to compute the
maximum profit (so it need not compute the actual matching) based on pruning a tree
where the optimal solutions are at the leaves. Define the structure of the tree, state how
the tree should be pruned, and state the width and height of the resulting pruned tree.

(b) Define the array that will be used by the dynamic programming algorithm that naturally
results from this pruning.

(c) Give pseudo-code for the resulting dynamic programming algorithm. Again note that
this algorithm need only compute the maximum profit, and not the actual matching.
State the worst-case running time as a function of $n$ and $k$.

(d) Explain how an algorithm could reconstruct the actual matching from the array com-
puted by this dynamic programming algorithm.

4. (20 points) Assume that you are given a collection $B_1, \ldots, B_n$ of boxes. You are told that
the weight in kilograms of each box is an integer between 1 and some constant $L$, inclusive.
However, you do not know the specific weight of any box, and you do not know the specific
value of $L$. You are also given a pan balance. A pan balance functions in the following
manner. You can give the pan balance any two disjoint sub-collections, say $S_1$ and $S_2$, of the
boxes. Let $|S_1|$ and $|S_2|$ be the cumulative weight of the boxes in $S_1$ and $S_2$, respectively.
The pan balance then determines whether $|S_1| < |S_2|$, $|S_1| = |S_2|$, or $|S_1| > |S_2|$. You have
nothing else at your disposal other than these $n$ boxes and the pan balance. The problem is
to determine if one can partition the boxes into two disjoint sub-collections $P_1$ and $P_2$ of equal
weight. So the output is just a single bit denoting whether such a partition is possible. Every
box must be in exactly one of $P_1$ and $P_2$. Give an algorithm for this problem that makes at
most $O(n^2 L)$ uses of the pan balance. For full credit you must explain why your algorithm
is correct, and state the worst-case number of uses of the pan balance by your algorithm, and
fully justify your claimed bound.

For some partial credit,you can give an algorithm where the number of uses is polynomial in
$n$ and $L$. 