1 Problem

The answers to the following should fit in the white space below the question.

1. Write down Kruskal's algorithm. It is sufficient to write down the main while loop and the rule describing how the algorithm proceeds. [1 point]

```
let $T = \emptyset$, $R = E$
while $R \neq \emptyset$
    find the minimal weight edge e from R
    if adding e does not create a cycle in T:
        add e to T
    remove e from R
T is the minimal spanning tree
```

2. State the *cut property* we used in class to analyze Kruskal's and Prim's algorithms. [1 point]

let $S \subseteq V$ $S \neq \emptyset, S \neq V$ be the cut,
the minimal weight edge $e = \{u, v\}$ crossing the cut
$(u \in S, v \notin S)$
must be in the minimal spanning tree

3. Suppose we are given an instance of the Minimum Spanning Tree Problem on a graph G, with edge costs that are all positive and distinct. Let T be a minimum spanning tree for this instance. Now suppose we replace each edge cost c_e by its square, c_e^2, thereby creating a new instance of the problem with the same graph but different costs.

   True or false: T must still be a minimum spanning tree for this new instance. [1 point]

True

$c_{e_i} < c_{e_j} \Rightarrow c_{e_i}^2 < c_{e_j}^2$

by kruskal's algorithm, the same spanning tree T

4. Consider the weighted interval scheduling problem where we are given n jobs as input with the i'th job having start time s_i, finish time f_i, and value v_i. (Thus, the input to the problem is n triples $(s_1, f_1, v_1), \ldots, (s_n, f_n, v_n)$.) Recall that our goal is to find the set of non-conflicting jobs with the highest possible total value. Consider the following greedy algorithm for the question:

   (a) Set $A = \emptyset$, $R = \{1, 2, \ldots, n\}$.
   (b) While $R \neq \emptyset$:
      i. Pick job $i \in R$ with highest v_i/f_i (value to finish time ratio) and add i to A.
      ii. Remove i and all jobs that conflict with i from R.
   (c) Return A.

   True or false: A achieves the highest possible total value. If true, provide a brief explanation. If false, provide a counterexample. [2 points]

False

$$\begin{cases} 1 \quad (0,1,5) & \frac{v_i}{f_i} = 5 \\ 2 \quad (0,3,10) & \frac{v_i}{f_i} = \frac{10}{3} \end{cases}$$

the greedy will pick job 1, not job 2

5. You have n items with the i'th item having weight w_i. You also have a knapsack with total weight capacity W (i.e., it can safely hold items whose total weight is at most W). Describe an algorithm for picking a *largest* possible subset of items that can be placed safely in the knapsack. That is, describe an algorithm to find a subset $S \subseteq \{1, 2, \ldots, n\}$ of maximum possible size such that $\sum_{i \in S} w_i \leq W$. For full-credit, your algorithm should run in time $O(n \log n)$. You don't have to prove correctness or analyze the time complexity of the algoritm. [2 points]

   [Hint: One approach is to give a greedy algorithm.]

```
first sort all the items by its weight in increasing order
let $w = 0$   $S = \emptyset$
for each item in increasing weight:
    if $w + W_i > W$ : break
    $w += W_i$
    add $i$ to $S$
```

S is the subset of maximum size.

2 Problem

Suppose you have a weighted undirected graph $G = (V, E)$ where all the weights are distinct. Prove that if an edge e is part of a cycle C and has weight more than every other edge in the cycle, then e cannot be part of the minimum spanning tree in G. [3 points]

[Hint: Assume that the statement is false for the sake of contradiction and let T being a MST that contains the edge e. Arrive at a contradiction by a swapping argument as we did in class for proving the cut property.]

Suppose such an edge $e = \{u, v\}$ is part of the minimal spanning tree T

let C be the cycle containing e, and e has more weight than every other edge in the cycle,

and $f = \{u', v'\} \in C$, $f < e$, f not contained in T

since C is a cycle, there must exist an edge $\in C$ not in T, therefore f exists

let $S \subseteq V$ be a cut, s.t. $v \in S$, $v' \in S$, $u \notin S$, $u' \notin S$

exchange e with f in T to obtain $T' = T \cup \{f\} \setminus \{e\}$

T' has smaller total weight than T because weight of e is larger than f

T' is connected because for every path containing e, replace e with $C \setminus \{e\}$, which contains f, and therefore all vertices are still connected. Thus, T' is also a spanning tree

Contradiction, because there exists spanning tree T' with smaller total weight than minimal spanning tree T

Therefore, e must not exists in minimal spanning tree T

3 Problem

Give a dynamic programming algorithm for the following version of knapsack where you have three copies of each item. There are n types of items with weights $w_1, \ldots, w_n$ respectively and *value* $v_1, \ldots, v_n$ respectively and you have **three** copies of each item. Suppose you have a knapsack of total weight capacity W. We a say configuration $(a_1, \ldots, a_n)$ is *safe* if $0 \leq a_i \leq 3$ and $a_1 w_1 + a_2 w_2 + \ldots + a_n w_n \leq W$ (i.e., it is safe to pack a_1 copies of item 1, a_2 copies of item 2, …, a_n copies of item n into the knapsack). The value of a configuration is the total value of the items in the configuration: for a configuration $(a_1, \ldots, a_n)$, its value is $v_1 a_1 + v_2 a_2 + \cdots + v_n a_n$.

Give an algorithm which given the numbers $w_1, \ldots, w_n$, $v_1, \ldots, v_n$, W as input computes the maximum value achievable over all safe configurations. For full-credit it is sufficient to give a correct algorithm for the problem which runs in time $O(nW)$ and it is not required to prove correctness or analyze the time-complexity of the algorithm. You must provide full description of the algorithm. [4 points]

renumber all the items as follows

$W'_{3k-2} = W_k \qquad V'_{3k-2} = V_k$

$W'_{3k-1} = W_k \qquad V'_{3k-1} = V_k$

$W'_{3k} = W_k \qquad V'_{3k} = V_k$

and therefore a_k = number of items chosen from $3k-2, 3k-1, 3k$

let $V(0,w) = 0 \ \forall w$, $V(i,0) = 0 \ \forall i$

```
for i from 1 to 3n :
    for w from 1 to W :
        $V(i,w) = \begin{cases} V(i-1,w) & \text{if } w < W'_i \\ \max\begin{cases} V(i-1,w) \\ V(i-1, w-W'_i) + V'_i \end{cases} & \text{otherwise} \end{cases}$
```

$V(3n, W)$ is the maximum value achievable over all safe configurations

4 Problem

You are given two arrays of integers $X = [x[0], x[1], \ldots, x[m]]$ and $Y = [y[0], y[1], \ldots, y[n]]$ as input. For two subsequences of X, Y of the same length, i.e., sequences of indices $0 \leq i_1 < i_2 < \ldots < i_k \leq m$ and $0 \leq j_1 < j_2 < \ldots < j_k \leq n$, the value of the subsequences is defined as

$$\sum_{\ell=1}^{k} \frac{1}{1 + |x[i_\ell] - y[j_\ell]|}.$$

Give an algorithm that given X, Y as input computes the maximum possible value achievable over all subsequences. For full-credit, your algorithm should run in time $O(mn)$ (ignoring the cost of arithmetic, i.e., adding numbers). You don't have to prove correctness or analyze the time-complexity of the algorithm. [4 points]

Example: $X = [1, 4, 2, 5]$, $Y = [1, 2, 10, 4, 100]$. Here, if you look at subsequences $x[0], x[2], x[3]$ and $y[0], y[1], y[3]$ you get value $1/1+1/1+1/2 = 2.5$. Whereas, if look at subsequences $x[0], x[1], x[2], x[3]$ and $y[0], y[1], y[2], y[3]$, you get value $1/1 + 1/3 + 1/9 + 1/2 \sim 1.9444$. So the first subsequence has better value. Your goal is to find the best possible value achievable over all subsequences.

[Hint: Create subproblems like we did for edit-distance in class and develop the appropriate recurrence.]

$$M(i,j) = \begin{cases} 0 & \text{if } i<0 \text{ or } j<0 \\ \max\begin{cases} \frac{1}{1+|X[i]-y[j]|} + M(i-1, j-1) \\ M(i-1,j) \\ M(i,j-1) \end{cases} & \text{otherwise} \end{cases}$$

$M(i,j) = 0 \quad \forall\, i,j$

```
def recurrence(i,j):
    if i<0 or j<0 : return 0
    return max( $\frac{1}{1+|X[i]-y[j]|}$ + M(i-1,j-1) ,
                M(i-1, j),
                M(i, j-1))
```

⇒ continue on next page

```
for i from 0 to m
    for j from 0 to n
        M(i, j) = recurrence(i, j)
```

$M(m,n)$ is the maximum possible value

5 Problem

Consider the following variant of the RNA sequencing question. Given a sequence $X = (x_1, \ldots, x_n)$, a set of pairs $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$ is an *allowed* set of pairs if the following hold:

1. Each index appears in at most one pair in M (i.e., no repetitions).
2. Each pair is one of $\{G, C\}$ or $\{A, U\}$. That is, for all $1 \leq p \leq m$, $\{x_{i_p}, x_{j_p}\}$ is one of $\{G, C\}$ or $\{A, U\}$.
3. No sharp edges: For all pairs $(i, j) \in M$, $i < j - 4$.
4. No crossing edges: If pairs $(i, j), (k, \ell) \in M$, then we cannot have $i < k < j < \ell$.

(These are the same rules as we worked with in class.)

The *stability* of an allowed set of pairs M is given by the following formula:

$$stability(M) = \sum_{p=1}^{m} (j_p - i_p)^2.$$

That is, the stability of the collection of pairs is the sum of squares of the number of characters between each pair. Give an efficient algorithm that given a sequence $X = (x_1, \ldots, x_n)$ computes the maximum possible *stability*(M) over all feasible sets of pairs M. For full-credit, your algorithm should run in $O(n^3)$ time. You do not have to prove correctness or analyze the time complexity of the algorithm. [4 points]

$$P(i,j) = \begin{cases} 0 & \text{if } i < j-4 \\ \max \begin{cases} (j-i)^2 + P(i+1, j-1) & \text{if } X_i \text{ and } X_j \text{ is an allowed pair} \\ \max_{i<k<j} P(i,k) + P(k+1,j) \end{cases} \end{cases}$$

$P(i,j) = 0 \quad \forall i,j \qquad C(i,j) = \text{false} \quad \forall i,j$

```
def recurrence(i,j) :
    if $i < j-4$ : return 0
    if $C(i,j)$ is true : return $P(i,j)$
    maxP = 0
    if $\{X_i, X_j\}$ allowed pair :
        maxP = $(j-i)^2$ + recurrence(i+1, j-1)
    for k from i+1 to j-1 :
        maxP = max(maxP, recurrence(i,k) + recurrence(k+1,j))
```

$\Rightarrow$ next page

```
    P(i,j) = maxP
    C(i,j) = true
    return maxP
```

recurrence(1, n) gets the maximal possible stability