COMP 3804 – Design and Analysis of Algorithms
Assignment 2

Due: February 17, 2017 at 23:55

- Your solutions should be submitted online on cuLearn in the form of a single PDF file.
- Your answers should be precise, concise and clear. All algorithms should be given in pseudocode.
- Every part of every theory question is worth 2 marks. The grading scheme is 2 points for a correct answer, 0 for a completely incorrect answer, and 1 point for something in-between. The implementation part (question 4i) is worth 10% of the grade.

1. Let $S3(X)$ be the number of sequences of 1s, 2s, and 3s that sum up to some given number X. For example, there are 7 sequences that add up to 4: 31, 22, 211, 13, 121, 112, and 1111, so $S3(4) = 7$.
   (a) Find the optimal substructure in $S3(X)$ and express it as a recurrence.
   (b) Give a dynamic programming algorithm that computes $S3(X)$, based on the recurrence from 1a.
   (c) Analyze the running time of your algorithm.

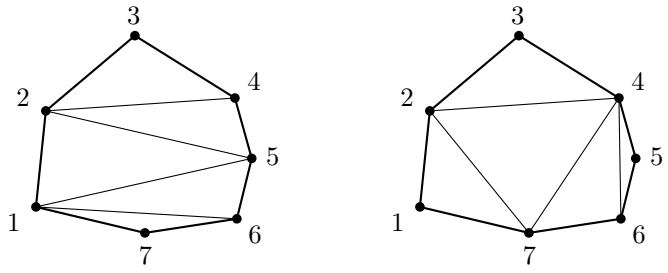

Figure 1: Two triangulations of the same convex polygon. The triangulation on the right has smaller weight than the triangulation on the left.

2. Given a convex polygon P with n vertices, a *triangulation* of P consists of $n-3$ diagonals that do not intersect each other, except possibly at their endpoints (see Figure 1 for an example). These diagonals partition P into a set of disjoint triangles, which is useful for many applications, in particular in computer graphics. The *weight* of a triangulation is the sum of the lengths of its diagonals. We are interested in finding the triangulation of minimum weight. For convenience, we number the vertices of P from 1 through n in clockwise order. Note that any subsequence of vertices forms a new, smaller, convex polygon.
   (a) Prove that the minimum weight triangulation of P consists of a triangle $(1, k, n)$ that includes the edge $(1, n)$, and the minimum weight triangulations of the two convex polygons formed by the vertices 1 through k and k through n.
   (b) Let $W(P)$ be the weight of the minimum weight triangulation of P. Express $W(P)$ as a recurrence, using the property you proved in 2a.
   (c) Give a dynamic programming algorithm that computes $W(P)$, based on your recurrence.
   (d) Analyze the running time of your algorithm.

3. When a new gene is discovered, a standard approach to understanding its function is to look through a database of known genes and find close matches. The closeness of two genes is measured by the extent to which they are aligned. To formalize this, think of a gene as being a sequence of characters from an alphabet $\Sigma = \{A, C, G, T\}$. Consider two genes (sequences) $X = ATGCC$ and $Y = TACGCA$. An alignment of X and Y is a way of matching up these two genes by writing them in columns, for instance:

$$\begin{array}{cccccccc} - & A & T & - & G & C & C & \\ T & A & - & C & G & C & A & \\ \hline -3 & +1 & -3 & -3 & +1 & +1 & -1 & = -7 \end{array}$$

Here the $-$ indicates a gap. The characters of each gene must appear in order, and each column must contain a character from at least one of the genes. The score of an alignment is calculated by assigning a score to each column and adding these up. A column with two matching characters gets score $+1$, a column with two mismatched characters gets score -1, and a column with a gap gets score -3. The score of the alignment above would be $-3 + 1 - 3 - 3 + 1 + 1 - 1 = -7$. Given two genes, we want to find the alignment with the highest score.

(a) Prove that if X and Y end with the same character, there is an optimal alignment whose final column contains those two characters. (*Hint:* try a proof by contradiction.)

(b) Suppose X ends with A and Y ends with C. What are the possibilities for the final column of an alignment of X and Y?

(c) Give a recurrence for the score $S(X, Y)$ of the optimal alignment of X and Y.

(d) Give a dynamic programming algorithm to compute $S(X, Y)$, based on your recurrence.

(e) Analyze the running time of your algorithm.

4. You're a voracious reader with a large collection of books $B = [b_1, b_2, \ldots, b_n]$ and you're going on vacation. You've already packed all your essentials and discovered that you have W grams left to fill up with books. Because you want to optimize your reading pleasure, you've rated every book b in your collection for enjoyment ($e(b)$) and then used a scale to find out how much each books weighs in grams ($w(b)$). You only have one copy of each book, and you are interested in maximizing the sum of the enjoyment values.

For example, suppose that $W = 1000$ and you are considering the following four books:

	b_1	b_2	b_3	b_4
w	600	300	400	200
e	30	14	16	9

Then the optimal book selection consists of books b_1 and b_3, which together weigh exactly 1 kilogram and have a total enjoyment of 46.

(a) Suppose that $S = \{b_1, b_4, b_{10}\}$ is the set of books with maximum total enjoyment among all sets of books that weigh at most 1000 grams in total. Show that $\{b_1, b_4\}$ is an optimal set of books among all sets of books that weigh at most $1000 - w(b_{10})$ grams in total and only use books b_1 through b_9.

(b) Use the observation from 4a to give a recurrence for $S(W, k)$: the maximum total enjoyment over all sets of books with total weight at most W that only use books b_1 through b_k.

(c) Give a dynamic programming algorithm that computes the maximum total enjoyment over all sets of books that weigh at most W grams.

(d) Analyze the running time of your algorithm.

(e) Is this running time polynomial in the *size* of the input? Why, or why not?

You have also classified each book in your collection into one of three genres: Classics, Fantasy, and Science Fiction. To make sure that your books aren't too similar, you want to pack at least G books from each genre.

(f) Modify your recurrence from 4b to account for this new condition. (*Hint:* introduce three new variables, representing the number of books from each genre, up to G.)

(g) Modify your algorithm from 4c to account for this new condition.

(h) Analyze the running time of your new algorithm.

(i) (10%) Implement your algorithm as `computeMaximumEnjoyment` in `BookPacker.java` in the accompanying zip file. Submit your source code to the submission server (not cuLearn).

(j) **Bonus:** Implement the `computeMaximumEnjoymentBooks` method that returns the set of books that gives the maximum enjoyment, instead of just the enjoyment value.