EECS 233 Homework 2

General requirements:

- Due at 11:00 PM on the posted due date.
- Include your name and network ID as a comment at the top of all of your programs.
- Create a typed document (.txt or .pdf) with answers to the questions.
- Upload your document and all .java files as a .zip file to Blackboard. Do not use other formats such as .rar.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus.

Instructions: This assignment deals with the problem of computing the maximum subsequence sum, as described in Chapter 2 of the Weiss book and discussed in class. Problems #1 – #3 require typed answers. Only problem #4 requires you to write a program. Note that sample code for the algorithms in the textbook was posted with the September 9 lecture notes, but it is not required for this assignment.

1. For an array A of size N = 3, there are six possible subsequences: A_1, A_2, A_3, A_1A_2, A_2A_3, $A_1A_2A_3$. How many subsequences are possible for N = 4? Show your work.

2. Derive a formula for the total number of possible subsequences for an arbitrary size N. You may use any combination of diagrams, equations, and written logic. Hint: visualize a 2-D grid where one dimension represents the starting index of each subsequence (0 to N-1) and the other dimension represents the length of the subsequence (1 to N).

3. Algorithm #1 (Figure 2.5) in the Weiss book, which was discussed in class, is very inefficient. One reason is that it redundantly computes sums for some subsequences. For the case N = 3, the sum of A_1 is computed three times (A_1, A_1A_2, $A_1A_2A_3$). Likewise, the sum of A_2 is computed twice (A_2, A_2A_3), and the sum of A_1A_2 is computed twice (A_1A_2, $A_1A_2A_3$). The following analysis table shows the number of summations performed for each subsequence:

Subsequence:	A_1	A_2	A_3	A_1A_2	A_2A_3	$A_1A_2A_3$
# of summations:	3	2	1	2	1	1

   Any instance of repeating a summation is redundant. By this definition, there are a total of 4 redundant summations in the table above. Repeat the above analysis for N = 4. Show your work.

4. Write a program that computes the number of redundancies for a range of N values. Display the values of N, the number of subsequences using your formula from problem #2, and the number of redundancies. Below is an example of how your output might look. Exact formatting is not important. Hint: one approach is to use the original triple-loop algorithm and maintain an array of counters for each subsequence, similar to the 2-D grid that was suggested for problem #2 above. Other approaches may be possible too.

```
N = 10. Subsequences = 55. Redundancies = 165.
N = 15. Subsequences = 120. Redundancies = 560.
N = 20. Subsequences = 210. Redundancies = 1330.
N = 25. Subsequences = 325. Redundancies = 2600.
N = 30. Subsequences = 465. Redundancies = 4495.
N = 35. Subsequences = 630. Redundancies = 7140.
N = 40. Subsequences = 820. Redundancies = 10660
```

Grading rubric coming soon!