CSE 6140 / CX 4140 Assignment 4
due Nov 5, 2018 at 11:59pm on Canvas

1. Please upload a *single* PDF named `assignment.pdf` for all of your answers/report containing:
   (a) a typed preamble that contains (-2.5 pts if not included):
      i. the list of people you worked with people for each question (if applicable),
      ii. the sources you used,
      iii. and if you wish, your impressions about the assignment (what was fun, what was difficult, why...);
   (b) your solutions for all problems, typed (not handwritten).
2. a single zip file named `code.zip` containing your code, README, and results for Programming problem. Please do *not* place your report in the zip file, upload it separately, i.e. one pdf, one zip file.
3. **Not abiding by the submission instructions will cause you to lose up to all of the points for a problem.**
4. If you do not understand the question, please ask on Piazza or come to office hours. Misunderstanding the question is not a valid excuse for losing points.
5. Remember the academic honor code: you are not allowed to copy from other students, you may not use material from prior classes, and googling questions on the Internet is not allowed.

1 NP-complete (12.5 pts)

Devan is the owner of Devan's Amazing Ice Cream. You and your best friend visit the store, and you decide to treat your friend. There are n unit measures of ice cream flavors (say a scoop, and they don't split a scoop into smaller units), with the i^{th} flavor having a likeability score of s_i (each s_i is unique). They offer to pack your ice cream into two buckets. Each bucket is capable of holding as many scoops as possible. You being a good friend want to produce two equally pleasing buckets, and wonder if there's a way to divide up the flavors such that the two buckets B_1 and B_2 have exactly the same score $L_1 = L_2 = \frac{L}{2}$ (you may assume L is even, and it is the sum of likeability of all flavors). You do not care that the quantities of the buckets may be different, only that their likeability scores are the same. Unfortunately for you, you cannot come up with a good algorithm to quickly find the best arrangement. Even worse, your best friend tells you that you may just have to settle for containers with lopsided likeability scores the next time this happens. Prove that your best friend is right by showing that this problem is NP-Complete. Remember to follow the steps from lecture to prove NP-completeness; lack of any of the steps will result in a suboptimal grade. *Hint*: The Subset Sum problem (which is NP-complete) is that, given a set of integers S, is there a subset of S that sums to a value k?

2 "In Hartford, Hereford, and Hampshire... (12.5 pts)

... hurricanes hardly *ever* happen." Unfortunately, though, hurricanes do happen elsewhere. As such, you've been formally requested by the British prime minister, Sir Henry Higgins, to lead the project on setting up emergency bunkers throughout England in hopes of reducing casualties during a hurricane. In particular, inhabitants of every town in England should be able to reach a bunker in a reasonable amount of time.

One proposal is to build a bunker in each and every town, thus eliminating travel time and ensuring safety for all. However, due to deployment and maintenance costs, this solution turns out to be outside of the allocated budget. A much more cost effective strategy is to build a single bunker at a reasonable distance from all the towns, but that will lead to long travel times and congestion along the roads, which is expected to increase casualties.

Instead, you propose a happy medium: build a set of bunkers such that a town either has a bunker built in it, or is directly connected to a town which has a bunker. This reduces the number of bunkers needed, while still allowing all inhabitants ample time to get to safety. Satisfied with your proposition, PM Higgins allocates funds for this project (code-named Pygmalion) and asks you to come up with a method to solve it.

You formally set up the problem, `PYGMALION`, as follows: Given an undirected graph G, where nodes represent towns and edges represent roads, and given a number k, is there a way to build k bunkers at k different towns so that every town either has its own bunker, or is connected by a (direct) road to a town that does have a bunker?

Thinking about a solution, you sadly realize what you've gotten yourself into: Show that `PYGMALION` is NP-Complete. Follow all the steps we have outlined in class for a complete proof.

3 Programming (25 pts)

Devan has recently earned $10,000 in cash by gambling (lucky him!). He wants to buy a brand new car. However, he doesn't have enough money yet and of course he is wise enough to not take the risk of gambling again. He usually takes the bus home while dreaming of his own car. "My dark gray car, you are the most beautiful car in the world", he imagines.

One day, while day-dreaming, his eyes fell on an advertisement which changed his life:

Do you want money to buy a car? or a home?
Join us today. Tomorrow is too late!
Pool-o-Pale Investment Co. Visit www.pool-o-pale.com.

He visits the website as soon as he gets home and reads the rules and regulations. He finds that he has to invest his money. They will pay his daily interest, a typical banking approach. He then finds a very appealing rule:

The interest rates are known in advance!

For example, tomorrow's interest rate is 3.5 %. This means that tomorrow, the bank will pay Devan 0.035×10000. The interest rate on the day after tomorrow is -2.1%, which means that they will claim 0.021×10000 of his money on that day. Devan gets excited about this: he can invest on the days with positive interest rates only! "That's great!", he thought, feeling that he was closer than ever to buying the car. But then, he goes on to read the next rule:

Every person can join the Pool-o-Pale once!

"What a bad rule!" he whispered disappointedly. This means that he has to join the plan on one given day, and remain so till some later day. Then, he will earn money at the rate equal to the summation of the interest rates on those days. "How can I earn as much money as possible? I wish I knew of an algorithm that finds the best investment period for me", he thinks. That night he slept while driving his dark gray car in his dreams.

Let's help Devan buy a car! Tomorrow morning, he is going to a branch of Pool-o-Pale. The manager will give him a spreadsheet containing the fixed interest rates from now until some days later. He has less than ten seconds to decide the interval he is going to invest within. You are going to help him find an efficient algorithm to accomplish this. You can, because you know how to design and analyze efficient algorithms!

Suppose the manager will give him a text file containing on its first line, n, the total number of days in the plan. Then, at line i he will receive a (positive or negative) real number indicating the daily interest rate, say a_i. If you really want to help him, you have to find the indices j and k such that $\sum_{j \leq i \leq k} a_i$ is maximum. Assume that there exists at least one day where the interest rate is positive (otherwise, there is no reason why Devan should invest). Remember you have only ten seconds.

You, as an algorithm expert, should try and analyze the following proposals:

- A brute-force approach for this problem seems very naive. You can design a faster algorithm. Believe in yourself! Implement a *divide-and-conquer* approach by splitting the array into two halves. The best solution will either be:
  - fully contained in the first half
  - fully contained in the second half
  - such that its start point is in the first half and its end point in the second half

  The first two cases are handled recursively. The third one is a linear search.
- Secondly, you will implement a more clever solution by *dynamic programming*: Assume that the days are indexed by the set $I = \{1, ..., n\}$. Let $B(j)$ denote the maximum sum of interest rates that can be obtained if $j \in I$ is Devan's last day of investment. Then, it is easy to see that $B(j)$ can be computed using the following recurrence relation:

$$B(j) = \begin{cases} 0 & j = 0 \\ \max\{B(j-1) + a_j, 0\} & \text{otherwise} \end{cases}$$

  In words, if j is Devan's last day of investment, the maximum interest rate he can obtain up to j is either: the maximum interest rate he can obtain up to $j-1$, plus that of the j-th day (if $B(j-1) + a_j \geq 0$); or zero (if $B(j-1) + a_j < 0$, i.e. there is no investment interval ending on day j that is profitable). You can easily (and efficiently) compute $B(j), \forall j \in I$ using a bottom-up approach. Once you have computed all the necessary $B(j)$ values, you can solve Devan's problem by returning the the best i and j values (Hint: the best i and j correspond to the $B(j)$ that is maximum among all $j \in I$).

You will help Devan by implementing the two aforementioned algorithms (divide-and-conquer and dynamic programming). Your algorithms should take inputs and generate outputs as follows.

Input: The first line contains two numbers. The first one, n, is the number of days. The second one, k, is the number of instances of the problem you should solve. Then, the next k lines contain n comma separated values. The input files are named `<n>.txt`, e.g. 7.txt and have the form:

```
7,3
-1.5,3.4,-3.1,1.7,2.7,-4.8,3.4
-1.2,3.8,-6.1,9.7,2.8,-5.8,1.4
-3.5,6.4,-3.1,1.7,1.7,-1.8,3.2
```

Output: For each algorithm and each input file there should be an output file with k lines. Each line has four numbers. The first one is the maximum value of interest rate Devan will receive. The next two numbers are the indices of the the start and end days of optimal investment, and are in the range $[1, n]$. The last value is the running time of the corresponding algorithm in milliseconds. Values are separated by commas, and non-integer values are output with two decimal digits.

```
4.78,2,6,1554.21
...
...
```

Your should submit the output for both algorithms corresponding to the input files provided. Your outputs should be named as

`<GTusername>_output_<algorithm>_<n>.txt`

where `<algorithm>` should be either `dc` (for divide and conquer) or `dp` (for dynamic programming). Put all output files in a folder named `output`.

Sample data for debugging: We provide a sample input file with $(n = 10, k = 10)$ in `10.txt`, and the corresponding sample output file in `drobinson67_output_dc_10.txt`. If your algorithms are correct, they will have the same values for the first three columns of the sample output file. Note that you should not submit your output file for this sample dataset.

Deliverables: You should create a zip file for the programming portion of this assignment that includes the following:

- Source for the two algorithms, well structured and commented. Similar to the previous assignments, you are allowed to use Python, Java, or C++.
- A `README` file explaining how to run your codes. If you use Java or C++, please also include the command(s) you use to compile your code (we should be able to compile (if necessary) and run your code and see the output files generated). Your code should contain no dependencies and should compile or run on a vanilla installation of Ubuntu 18.04 with only `build-essential`, `default-jdk`, `python3`, and `python2.7` installed. If you have any doubt about whether it will compile or run, please reach out to the TAs before the deadline.
- A folder named `output` containing the corresponding output of the input files provided in. The output should be in the format described in the "Output" section above.

Your report should be part of your `assignment.pdf`. The report should include:

- A description of your divide and conquer algorithm.
- A description of your dynamic programming algorithm.
- Time and space complexity analysis of both algorithms.
- A single graph with two lines that shows how the average running time of your algorithms grows with n. For each input size, n, average the running time over the k instances in the input file for that value of n.
- Observations about your empirical results that tie back into your time and space complexity analysis.
- Discussion on how the two algorithms compare with each other in terms of the complexity and empirical performance.