FIT2004: Assessment questions for week 6

THIS PRAC IS ASSESSED! (5 Marks)

DEADLINE: Sunday, 08-Apr-2018 23:55:00
LATE PENALTY: 25% per day.
MARKING: You will be interviewed during your lab or at another time as arranged by your demonstrator who will ask you a series of questions to assess your understanding of this exercise, and gauge how you implemented it. It is required that you implement this exercise strictly using **Python programming language**. Practical work is marked on the time and space complexity of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! "Forgetting" is not an acceptable explanation for lack of understanding. Demonstrators are not obliged to mark programs that do not run or that crash.
You will lose all marks for a given task if your algorithm does not produce correct results for the task. Also, heavy penalties are applied if your complexity is worse than the required complexity and you may lose all marks if the complexity is similar to that of the naive algorithm. This is because our focus is on developing efficient algorithms and data structures. Also, do not assume that each task carries equal marks.
SUBMISSION REQUIREMENT: You will need to submit a zipped file containing your Python programs (named salesman.py and playlist.py) as well as a PDF file briefly describing your solution and its space and time complexity. The PDF file must give an outline of your solution (e.g., a high level idea of how did you solve it) and the **worst-case** space and time complexity of your solution. Penalties will be applied if you fail to submit the PDF file. The zipped file is to be submitted on Moodle before the deadline.
Important: The assignments will be checked for plagiarism using an advanced plagiarism detector and the students will be interviewed by tutors to demonstrate the understanding of their code. Last year, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, many failed the unit. "Helping" others is NOT okay. Please do not share your solutions with others. If someone asks you for help, ask them to visit us during consultation hours for help.

1 Task 1: Salesman's Dilemma

A door-to-door salesman is visiting a strange town where people really hate their neighbors. He visited every house in the town and everyone told him that they are willing to buy some items from him but only if he does not sell anything to any of his neighbors. He noted down the items the residents of each house are willing to buy. Specifically, for each house i, he has recorded the total price of the items, p_i, residents of the house i will buy if he does not sell

anything to any of the neighboring houses of i. He told everyone that he will come back after thinking about it.

He came to you asking for help in maximizing his sale. Specifically, he wants you to write a Python program that determines a strategy to maximize his sale. The program must print the house numbers of all the houses he must sell to (in ascending order of house numbers). Your program must also output his sale if he follows your advice.

But you ask him, "How do I know the neighboring houses of a house". He tells you that there are N houses along the street and are sequentially numbered 1 to N. A house numbered i is called the neighboring house of a house j if $|i - j| \leq k$ where k is a value that the user enters at the start of your program and $|i - j|$ denotes the absolute difference between i and j. For example, if $k = 3$, the house number 50 is called the neighboring house of all the houses numbered 47 to 53 and the house number 2 is the neighboring house of all the houses numbered 1 to 5. If $k = 1$, the house number 50 is the neighboring house of 49 and 51, and the house number 2 is the neighboring house of 1 and 3.

Your goal is to write an efficient Python program to help him. Your program should solve the problem using $O(N)$ space and $O(N)$ time in the worst-case. Your program must be named `salesman.py`.

1.1 Input

The input file `houses.txt` consists of 2 lines. The first line of the input is the value of N corresponding to the total number of houses in the street. The next line contains N space separated numbers where $i - th$ of the numbers is p_i denoting the total price of the items the residents of house i are willing to buy if he does not sell to any neighboring house of i. Below is a sample input.

```
10
50 10 12 65 40 95 100 12 20 30
```

The above input shows that there are 10 houses. The first house will buy items worth of 50 and so on.

1.2 Output

Your program must ask the user to enter a value of k which you can assume will always be a positive integer smaller than or equal to N. Depending on the value of k, your program must generate output in two lines. The first line must give the house numbers he must sell to in order to maximize his sale. The house numbers must be printed in ascending order. The second line must print the total sale if he sells to these houses. Below are some sample outputs for the above input file for different values of k.

```
Enter value of k: 1
Houses: 1 4 6 8 10
Total Sale: 252
```

```
Enter value of k: 2
Houses: 1 4 7 10
Total Sale: 245
```

```
Enter value of k: 3
Houses: 1 6 10
Total Sale: 175
```

```
Enter value of k: 10
Houses: 7
Total Sale: 100
```

If there are more than one possible answers (e.g., multiple ways to achieve the maximum sale), you can print any of the answers.

2 Task 2: Playlist

Alice drives to work and listens to her favorite songs on her way. She hates it when she is in middle of a song and reaches her destination. This is because if she does not listen to a song till the end then it keeps playing in her mind for the whole day which affects her performance. She does not want to sit in the car waiting for the song to finish. Also, she does not want to stop listening to the song before she has arrived her destination. In other words, she wants the songs to play such that a song ends exactly when she has arrived her destination.

She is writing a phone app to help scheduling the songs. The app communicates with Google Maps app and determines the estimated time T (in minutes) to reach the destination. We will assume that the estimate T is perfect and it will take her exactly T minutes to reach the destination. Her app also has obtained the duration (in minutes) of each song in her phone. However, she does not know how to proceed further. Well, you have guessed correctly. She has come to seek your help once again.

Your goal is to write an algorithm that determines a playlist of the songs such that, if the songs are continuously played during her journey, a song finishes exactly when she reaches the destination. In other words, if the estimated duration of her journey is T minutes, the total duration of the songs in the playlist must also be exactly T. Furthermore, she does not want any song to be repeated. Therefore, your playlist should not have duplicate songs.

You are given a list of songs numbered 1 to N (called ID of the song). A song with ID i has a duration d_i (in minutes). You can assume that no two songs have the same duration. The output must print the playlist meeting the above requirement (in ascending order of the IDs of the songs). If it is not possible to find such playlist, you must report a message stating "Bad luck Alice!".

The worst-case time complexity of your algorithm must be $O(NT)$ and the worst-case space complexity must be $O(NT)$ where N is the total number of songs and T is the estimated journey time. Your program must be named `playlist.py`.

2.1 Input

The input file consists of 2 lines. The first line of the input contains a single integer N that represents the total number of songs in the phone. The next line contains N space separated numbers where $i-th$ of the numbers is d_i denoting the duration of the song with ID i. Below is a sample input.

```
5
10 3 5 7 2
```

In the above input, there are 5 songs: duration of song with ID 1 is 10 minutes, and the song with ID 2 is 3 minutes and so on.

2.2 Output

Your program must ask the user to enter the journey time T (which you can assume will always be a positive integer). Your output must print the playlist that meets the above requirements, in ascending order of IDs, with the duration of each song displayed next to the song ID (see below). If there are more than one correct answers (multiple playlists with the total duration T), you are free to print any of the playlists. If it is not possible to find such a playlist, you must display "Bad luck Alice!"

Below are some sample outputs for different values of trip length T.

```
Enter trip length: 14
Playlist
ID: 3 Duration: 5
ID: 4 Duration: 7
ID: 5 Duration: 2
```

```
Enter trip length: 17
Playlist
ID: 1 Duration: 10
ID: 4 Duration: 7
```

Note that the following answer for $T = 17$ is also correct.

```
Enter trip length: 17
Playlist
ID: 1 Duration: 10
ID: 3 Duration: 5
ID: 5 Duration: 2
```

In the case, where there are multiple correct answers (more than one valid playlists), you are free to print any of the playlists. If the trip length is 16, no playlist can be found that has the total duration equal to 16 (see the output below).

```
Enter trip length: 16
Bad Luck Alice!
```

Things to note

The provided input files are quite small. You may want to write a bruteforce algorithm (with exponential time complexity) to test the correctness of your dynamic program algorithm. You are also encouraged to try different (and probably larger) input files to test your algorithms. Your programs will be tested on different (and larger) input files. It is your responsibility to ensure that your algorithm is correct for all cases and meets the complexity requirements.

```
-=o0o=-
  END
-=o0o=-
```