CS325: Analysis of Algorithms, Winter 2024

Group Assignment 1

Due: Tue, 1/25/24

Homework Policy:

1. Students should work on group assignments in groups of preferably three people. Each group submits to CANVAS a zip file that includes their source code and their *typeset* report. The report must include the name of all group members. Specifically, for this assignment your zipped folder should contain two files named `assignment1.pdf`, and `assignment1.py`. One submission from each group is sufficient.
2. The goal of the homework assignments is for you to learn solving algorithmic problems. So, I recommend spending sufficient time thinking about problems individually before discussing them with your friends.
3. You are allowed to discuss the problems with other groups, and you are allowed to use other resources, but you *must* cite them. Also, you must write everything in your own words, copying verbatim is plagiarism.
4. *I don't know policy:* you may write "I don't know" *and nothing else* to answer a question and receive 25 percent of the total points for that problem whereas a completely wrong answer will receive zero.
5. Algorithms should be explained in plain english. You can use pseudocodes if it helps your explanation, but the grader will not try to understand a complicated pseudocode.

In the first group assignment of this quarter, we explore a set of related problems that can be efficiently solved using a divide-and-conquer approach. These problems share the characteristic of finding a subset of numbers in an array or multiple arrays that sum up to a constant within a given range. Perhaps, the most well-known problem of this kind is the subset sum, which is NP hard. But, the problems of this assignment are more constrained and admit polynomial time algorithms. In fact, our goal for this assignment is to devise algorithms with nearly linear running time.

(A) In the first problem, our input is consists of (i) two arrays $A[1,\ldots,m]$ and $B[1,\ldots,n]$ of integers, which can be negative, zero or positive, and (ii) a target value range denoted by $[t_{\min}, t_{\max}]$. Our objective is to find the number of possible pairs of items, one from A and one from B, that add up to a number within the range $[t_{\min}, t_{\max}]$. For example, if $A = [-2, 0, -1]$, $B = [-3, 10]$, and $t_{\min} = -3$ and $t_{\max} = 20$, the output must be 4 since $(-2, 10)$, $(0, -3)$, $(0, 10)$, $(-1, 10)$ are the pairs that add up to a number within $[-3, 20]$, while $(-2, -3)$, $(-1, -3)$ add up to numbers outside this range . Design an algorithm for this problem with running time $O((n + m) \cdot \log(n + m))$.

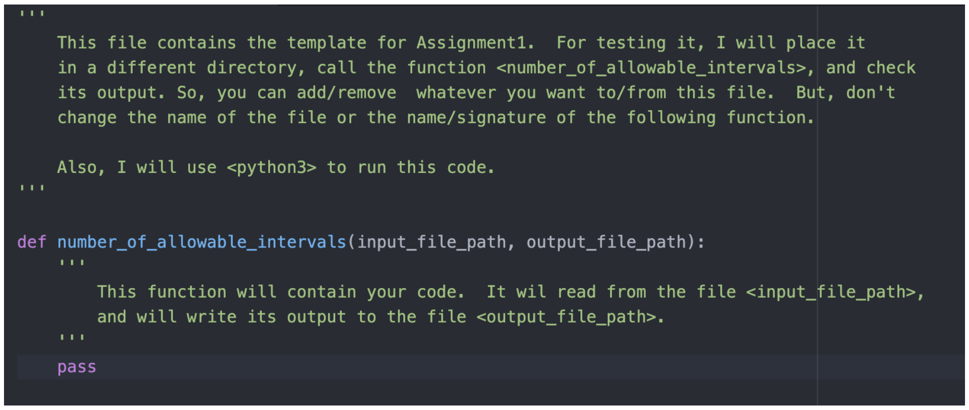
(B) In the second problem, we are given only one array $A[1,\ldots,n]$, and a target value range $[t_{\min}, t_{\max}]$. We would like to count the number of non-empty contiguous subarrays of A that add up to a value in the range $[t_{\min}, t_{\max}]$. For example, if $A = [-3, -4, 2, 0]$, $t_{\min} = -4$ and $t_{\max} = 3$, then the output must be 7, since $[-3]$, $[-4]$, $[2]$, $[0]$, $[-4, 2]$, $[-4, 2, 0]$, and

$[2, 0]$ are contiguous subarrays of A with sum in the range $[-4, 3]$, while $[-3, -4]$, $[-3, -4, 2]$, $[-3, -4, 2, 0]$ are the other contiguous subarrays of A with sum outside this range. Design an $O(n \cdot \text{polylog}(n))$ time algorithm for this problem. (Hint: Start with an $O(n^3)$ time algorithm, then reduce the running time to $O(n^2)$, and finally to $O(n \cdot \text{polylog}(n))$. For the last step, consider using divide and conquer and the algorithm in part (1)).

Report (60%). In your report, provide the description of your algorithms, running time analysis, and proof of correctness. Algorithms should be explained in plain english. You can use pseudocodes if it helps your explanation, but the grader will not try to understand a complicated pseudocode. Part (A) is worth 20 percent and part (B) 40 percent.

Code (40%). Submit a python program for the problem outlined in part (B). Your program will undergo testing against various test cases to assess both correctness and efficiency. For each test case, there's a time constraint—automatically stopping after 20 seconds if execution exceeds this limit. In such instances, the group will miss the points associated with that particular test case.

Note: it is important that your output is formatted as described below, since your codes will be tested automatically. You must implement the function "number_of_allowable_intervals" in the following code. The code you submit will be an implementation of this procedure in a file named "assignment1.py".



```
'''
    This file contains the template for Assignment1.  For testing it, I will place it
    in a different directory, call the function <number_of_allowable_intervals>, and check
    its output. So, you can add/remove  whatever you want to/from this file.  But, don't
    change the name of the file or the name/signature of the following function.

    Also, I will use <python3> to run this code.
'''

def number_of_allowable_intervals(input_file_path, output_file_path):
    '''
        This function will contain your code.  It wil read from the file <input_file_path>,
        and will write its output to the file <output_file_path>.
    '''
    pass
```

Input/Output The input file is composed two lines. The first line contains three integers $1 \leq n \leq 10^5$, which is the length of the array A as described in part (B) of the problem, and $-10^6 \leq t_{\min}, t_{\max} \leq 10^6$, which specify the allowable range. The second line contains n integers $g_1, g_2, \ldots, g_n$ that are separated by commas; you can assume $1 \leq i \leq n$, we have $-10^5 \leq g_i \leq 10^5$.

The output file must composed of one line that contains a single integer that is the number of contiguous subarrays of A that add up to a number within $[t_{\min}, t_{\max}]$.

Sample Input (1):
4, -4, 3
-3, -4, 2, 0

Sample Output (1):
5

Sample Input (2):
5, 2, 2
1,1,1,1,1

Sample Output (2):
9

Sample Input (3):
3, 0, 10
-1, 1

Sample Output (3):
2