This assignment is due on April 26. Submit your assignment via Gradescope. All submitted work must be done individually without consulting someone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

As a first step, go to the last page and read the section: Advice on how to do the homework.

Problem 1 (10 points)

We are given a multiset M of integers, i.e., a set in which every integer can occur multiple times. We want to know if we can assign these integers to two multisets M_1 and M_2 such that the total sum of all integers in M_1 equals the total sum of all integers in M_2.

Example:
When we have $M = \{4, 1, 1, 3, 5, 4, 2\}$, we can construct $M_1 = \{1, 2, 3, 4\}$ and $M_2 = \{1, 4, 5\}$. The sum of the integers in M_1 equals $1 + 2 + 3 + 4 = 10$, which is the same as the sum of the integers in M_2.

We are given three algorithms for this problem:

WRONGALGORITHM starts by removing two duplicates from M (if any exist) and adding one to each of M_1 and M_2. Next, it repeatedly takes the largest integer x from M and tries to find a multiset M' of integers in $M \setminus \{x\}$ such that the sum of the integers in M' is equal to x. If such an M' is found, it adds x to M_1 and M' to M_2. If it can continue this process until M is empty, it returns true. Otherwise it returns false.

PERMUTATIONALGORITHM generates a random permutation of the input and considers the integers in that order. It adds integers to M_1 until the sum is at least half of the sum of the integers in M. On equality, it returns true. Otherwise, it considers a permutation it hasn't tried yet and repeats the above process. This is repeated until either it returns true in one of its iterations, or there are no permutations left to try.

BALANCINGALGORITHM sorts the integers in non-increasing order and adds each integer to the multiset whose sum is smallest when that integer is considered. Once all integers are added to one of the two multisets, it returns whether their sums are equal.

```
1: function WRONGALGORITHM(M)
2:     $M_1, M_2 \leftarrow \emptyset, \emptyset$
3:     while $M$ contains some duplicate integer $x$ do
4:         $M \leftarrow M \setminus \{x, x\}$
5:         $M_1, M_2 \leftarrow M_1 \cup \{x\}, M_2 \cup \{x\}$
6:     while $M \neq \emptyset$ do
7:         $x \leftarrow$ largest integer in $M$
8:         $M \leftarrow M \setminus \{x\}$
```

```
9:          if there exist $M' \subseteq M$ s.t. $x =$ sum of integers in $M'$ then
10:             $M \leftarrow M \setminus M'$
11:             $M_1, M_2 \leftarrow M_1 \cup \{x\}, M_2 \cup M'$
12:         else return false
13:     return true
```

```
1: function PERMUTATIONALGORITHM(M)
2:     while there exists a permutation of $M$ we haven't tried do
3:         $M_1 \leftarrow \emptyset$
4:         while sum of integers in $M_1 <$ sum of integers in $M/2$ do
5:             Add next integer of the permutation to $M_1$
6:         if sum of integers in $M_1 =$ sum of integers in $M/2$ then
7:             return true
8:     return false
```

```
1: function BALANCINGALGORITHM(M)
2:     $M_1, M_2 \leftarrow \emptyset, \emptyset$
3:     Sort $M$ in non-increasing order
4:     for each integer $x$ in $M$ do
5:         if sum of integers in $M_1 \leq$ sum of integers in $M_2$ then
6:             $M_1 \leftarrow M_1 \cup x$
7:         else
8:             $M_2 \leftarrow M_2 \cup x$
9:     return sum of integers in $M_1 =$ sum of integers in $M_2$
```

a) Show that WRONGALGORITHM doesn't always return the correct answer by giving a counterexample.

b) Argue whether PERMUTATIONALGORITHM always returns the correct answer by either arguing its correctness (if you think it's correct) or by providing a counterexample (if you think it's incorrect).

c) Argue whether BALANCINGALGORITHM always returns the correct answer by either arguing its correctness (if you think it's correct) or by providing a counterexample (if you think it's incorrect).

Problem 2 (25 points)

Suppose you have k non-empty *sorted* doubly-linked lists $L_1, \ldots, L_k$, where $n = |L_1| + \ldots + |L_k|$. You are to design a data structure that supports both REMOVE-MIN and REMOVE-MAX in $O(\log k)$ time on the union of the lists; i.e., these operations act on the elements that have not been deleted so far from the data structure. We will only be removing elements, so you don't have to handle insertions. You are allowed $O(k)$ time to initialize the data structure and you are not allowed to modify the lists (you can traverse them but you are not allowed to add or remove elements from them). For full marks you need to meet *all* these requirements.

a) Describe your data structure, i.e., the structure and its operations.

b) Briefly argue the correctness of your operations.

c) Analyze the running time of your operations.

Problem 3 (25 points)

We are running a restaurant and as a new promotion management has decided to advertise how many lunch specials we have that cost at most \$15. Every lunch special consists of one main and one side dish. Our chefs like to try out new dishes on a regular basis, so it'd be great if we could efficiently compute the number of such combinations when needed.

Your task is to design a data structure that supports the following operations, where *name* is the name of a dish and *price* is its price (a positive integer):

- addNewMainDish(*name*, *price*): Adds a new main dish called *name* costing *price*.
- addNewSideDish(*name*, *price*): Adds a new side dish called *name* costing *price*.
- removeMainDish(*name*): Removes the main dish called *name* and returns its price, if *name* exists.
- removeSideDish(*name*): Removes the side dish called *name* and returns its price, if *name* exists.
- countCombinations(): Returns the number of combinations of a main dish and a side dish with total price at most \$15.

Each operation should run in $O(\log n)$ time where n is the number of dishes (i.e., mains plus side dishes), except countCombinations which should run in $O(1)$ time. Your data structure should take $O(n)$ space.

a) Design a data structure that supports the above operations in the required time.

b) Briefly argue the correctness of your data structure and operations.

c) Analyse the running time of your operations and the total space of the data structure.

Advice on how to do the home work

- Assignments should be typed and submitted as pdf (no handwriting)
- When designing an algorithm or data structure, it might help you (and us) if you briefly describe your general idea, and after that you might want to develop and elaborate on details. If we don't see/understand your general idea, we cannot give you points for it.
- Be careful with giving multiple or alternative answers. If you give multiple answers, then we will give you marks only for "your worst answer", as this indicates how well you understood the question.
- Some of the questions are very easy (with the help of the lecture notes or book). You can use the material presented in the lecture or book without proving it. You do not need to write more than necessary.
- When giving answers to questions, always prove/explain/motivate your answers.
- When giving an algorithm as an answer, the algorithm does not have to be given as (pseudo-)code.
- If you do give (pseudo-)code, then you still have to explain your code and your ideas in plain English.
- Unless otherwise stated, we always ask about worst-case analysis, worst case running times etc.
- As in the lecture, and as is typical for an algorithms course, we are interested in the most efficient algorithms and data structures.
- If you use additional resources (books, scientific papers, the internet, etc.) to formulate your answers, then add references to your sources.
- If you refer to a result in a scientific paper or on the web, you need to explain the results to show that you understand the results and how it was proven.