CSE 3500 – Algorithms and Complexity
Fall 2023 - Homework 2

Heap

(20 points) Question 1

You are given an array of integers. Write an algorithm with **linear** time complexity to check if this array satisfies the max heap property. The max heap property states that for any given node ***i***, the value of ***array[i]*** should be greater than or equal to the values of its children. Your algorithm should return True if it's a valid max heap and False otherwise. Analyze the time complexity of your algorithm.

(30 points) Question 2

Bringing back what we discussed in class, we explored the technique of merging two sorted lists into a single sorted list. Now, let's elevate the complexity a bit. Imagine you have k sorted lists with n elements in total. Your task is to merge these k sorted lists into one single sorted list efficiently.

Here's the challenge: Your algorithm should accomplish this in O($nlogk$) time complexity. I'll offer a hint: use a heap for k-way merging.

Run-time analysis

(50 points) Question 3

In this segment of the assignment, your objective is to put into practice two algorithms designed to determine whether a given number can be expressed as the sum of two other numbers within a provided list of integers. To simplify matters, we'll allow the same number to be used twice when composing the sum. To illustrate, if the list contains the number 10, then 20 = 10 + 10 and thus 20 is considered to be a sum of two numbers from the list.

You need to implement two algorithms:
Brute-force: You simply try all possible pairs of numbers to see if a given number matches each sum.
Binary-search: You would first sort the list of numbers and then perform binary search on the numbers. You can use existing functions to perform sorting that is provided by the language. If you write sorting yourself, make sure to write an efficient sorting algorithm (O(nlogn) time).

Now implement both algorithms using your favorite language (perhaps Python). Then run your program on the provided data to compare how the two different algorithms perform on small to large data files. Here are more detailed instructions:

You have two sets of files:
1. files listNumbers-n (listNumbers-10, listNumbers-100, listNumbers-1000, listNumbers-10000, listNumbers-100000, listNumbers-1000000) each one of these files contain different data size 10, 100, 1000, 10000, 100000, 1000000 random numbers respectively.
2. files listNumbers-n-nsol, each one of these files contain 10 numbers.
What you need to do:
1. For each number x in listNumbers-n-nsol file:
2. look for two numbers in listNumbers-n file, where the sum of these two numbers = x , or one number if you use it twice will also sum to x (ex: x = 30 , and you found 15 in listNumbers-n list)
3. repeat that for all pair of files (listNumbers-n and listNumbers-n-nsol)

Note: you don't need to find all pairs that sum to x, if one found, then stop and proceed to the next number in list listNumbers-n-nsol

File listNumbers-10-wsol is an example of that.

What to submit?

Submit a short report containing the following.

Settings:
(10 points) Write down the language you use, the machine (its CPU frequency and memory size) you use for testing your program.

Results:
(10 points) Provide a table comparing the two algorithms by listing the average running time for each data size.
(***10 points)*** Then plot the running time in a diagram (with the x-axis being the data size and y-axis being the running time); choose the proper scale to best demonstrate the results.
(10 point) Conclusion: Draw conclusion on why choice of algorithm matters.
(10 points) Code: Attach the source code of your implementation. If it is short enough, you may simply include your code as part of your report.