Final Exam 2022

NTNU IDATA 2302

Friday, December 9, 2022

A few tips before to start. Some questions ask you to argue or explain, other may ask your to prove. When arguing/explaining, we expect a few sentences to justify, but when proving, we expect a detailed step-by-step reasoning.

Add at least a sentence to motivate the answer to every question. No point will be given for a result alone.

You are **not** required to write a program that would actually "compiles". You can—if you feel it helps—but you can also use pseudo-code, or bullet points if you prefer, or any combination there of.

There is a bonus question, that is, you can still have a full score without answering it. If you answer it correctly, the points are however part of your total score.

Good luck!

1 Basic Knowledge

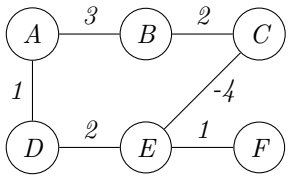
Question 1.1 (1 pt.)**.** *An algorithm runs in $O(n^2)$. Is it correct to say it also runs in $O(n^3)$?*

Question 1.2 (1 pt.)**.** *Consider the algorithm below that returns the smallest of its two arguments. Is it correct? Provide a proof.*

```
int minimumOf(int left, int right) {
    if (left < right) {
        return left;
    } else {
        return left;
    }
}
```

Question 1.3 (1 pt.)**.** *Consider a sorted sequence, say $(4, 9, 12, 17, 23)$ for instance. Which algorithm(s) can help check whether such a sequence contains a given value x, in less than linear time with respect to the length of the array? What time complexity does the algorithm take?*

Question 1.4 (1 pt.). *Consider the following graph, where edges are labelled with distances between nodes. What is the shortest path between vertices A and F? What is the associated distance?*



Question 1.5 (1 pt.). *What algorithm can we use to find the shortest path automatically in this graph? Explain your choice.*

2 Finding Duplicates

Consider a sequence of integers s, such as $s = (4, 61, 3, 10, 17)$. This exercise focuses on finding whether s contains duplicates, that is, the same number showing up several times. Here are a few examples:

- $(1, 3, 4, 3, 2)$ contains twice the value 3
- $(1, 3, 4, 6, 2)$ does not contain duplicate.
- $(1, 3, 4, 3, 4)$ contains duplicates: twice 3, and twice 4.

Question 2.1 (3 pts.). *Propose an algorithm "hasDuplicate" that detects whether a given sequence contains duplicates and runs in quadratic time in the worst case (i.e., $O(n^2)$).*

Question 2.2 (1. pt). *What is the worst case for this algorithm? What is the best case? Give an example for both.*

Question 2.3 (2 pts.). *How many comparisons does your algorithm perform in the worst case? Express this number as a function of the length ℓ of the given sequence. Detail your calculation.*

Question 2.4 (1. pt). *Where does your algorithm "waste" time performing unnecessary comparisons? Which data structure could you use to avoid that? Why?*

Question 2.5 (3 pts.). *Propose another algorithm that detects such duplicates in linear time (i.e., $O(n)$). Why does it run in linear time?*

3 Recursive Sums

Consider the following Java program. It computes the sum of the given array of integers values, for the range delimited by the two indices `start` and `end`. Note the recursive call. You can assume that `start` $\leq$ `end`.

```
int sum(int[]array, int start, int end) {
    if (start >= end) {
        return array[start];
    }
    return array[start] + sum(array, start+1, end);
}
```

Question 3.1 (2 pts.). *How many operations (i.e, arithmetic and logical operations) would the* `sum` *algorithm perform for a range of size n? (Hint: how would you model this number of operations as a recurrence relationship).*

Question 3.2 (2 pts.). *How much memory would the* `sum` *algorithm require for a problem size n? You can assume that each parameter occupies a single memory cell.*

Question 3.3 (2 pt.). *Sketch an alternative algorithm, using recursion, which computes the same sum. (Hint: try to add terms in a different way).*

4 Coin Change

Consider now the problem that cashiers have when they give change to customers who pay cash. Say for instance the customer must pay 57 kr but gives 70 kr. The cashier must return 13 kr, but using what coins? 1 coin of 10 kr plus 3 coins of 1 kr or maybe another combination thereof. In this exercise, your task is to create an algorithm that finds the minimum number of coins (no note) that adds up to a given amount. We will consider the following four types of coins (so-called denominations): 1 kr, 2 kr, 5 kr, 10 kr.

Question 4.1 (2 pts.). *Consider we have to give back 13 kr to the customer, and that our cash drawer contains 3 coins for each of the four possible types of coin. One way is to pick one coin of 10 kr and 3 coins of 1 kr. List all the other possible ways to return 13 kr?*

Question 4.2 (2 pts). *Provided our drawer contains 3 coins for each denomination, and that we have to return 13 kr. A friend tells us that the best solution to return 3 kr is to use 1 coin of 1 kr and 1 coin of 2 kr. What (similar) problem do we still have to solve in order to use this information for solving the original problem of returning 13 kr?*

Question 4.3 (1 pt). *How would you combine the solution given by our friend with the solution of this remaining problem to solve our initial problem, which is to return 13 kr?*

Let us model the solution of this coin change problem as a "coin set", that is a set of ordered pairs (d, n), where d is the denomination and n the number of coins of that type. For instance, a set of coins with only two 2 kr coins and one 5 kr coin would be $\{(0, 1), (2, 2), (1, 5), (0, 10)\}$. Should we be programming in Java, our algorithm would have the following signature:

```
CoinSet minimumCoins(int value, CoinSet drawer) {
    // ...
}
```

We assume as well the existence of the following operations to manipulate these `CoinSets`, which we shows here as a Java interface for the sake of simplicity

```
class CoinSet {
    // Empty coin set
    public CoinSet() {}

    // The denomination whose count is greater than 0
    int[] availableDenominations() { ... }

    // Add n coins of the denomination d
    put(int d, int n) { ... }

    // Remove n coins of the denomination d
    take(int d, int n) { ... }

    // Get the set with least coins
    static CoinSet smallest(CoinSet candidates[]) { ... }

}
```

Question 4.4 (3 pts.)**.** *Sketch a recursive algorithm, which, given a value v (in kroner) and the number of coins available for each denomination (i.e., the state of the cash drawer), finds the minimum number of coins that sum up to this given value v.*

Question 4.5 (3 pts.)**.** *What drawback do you see with this recursive algorithm? What technique(s) would you do to improve your solution?*

Question 4.6 (2 pts. (Bonus))**.** *Consider now that we have infinitely many coins for each denomination. Would your algorithm still terminate? Explain why?*

End of examination