Algorithms and Data Structures

NTNU IDATA 2302

Session II, May 2023

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.)**.** *What is the runtime complexity of the Bubble sort algorithm? Why?*

Question 1.2 (1 pt.)**.** *Consider the algorithm below, which counts the number of time a given letter occurs in the given word. As for the runtime, what is the best case scenario?´*

```
int countOccurrences(String word, char letter) {
  int count = 0;
  for (int i=0 ; i<word.length() ; i++) {
    if (word.charAt(i) == letter) {
      count++;
    }
  }
  return count;
}
```

Question 1.3 (1 pt). *Consider an algorithm A whose runtime is described by the function* $f(x) = \frac{3x-1}{x}$. *Is this correct to say that "A runs in* $O(1)$*"? Explain your reasoning?*

Question 1.4 (1 pt.). *Consider the hash-table shown below. It resolves collisions using "separate chaining". Suppose we insert a piece of data with key "Lisa", whose "hash" is 123. Explain where would it be inserted?*

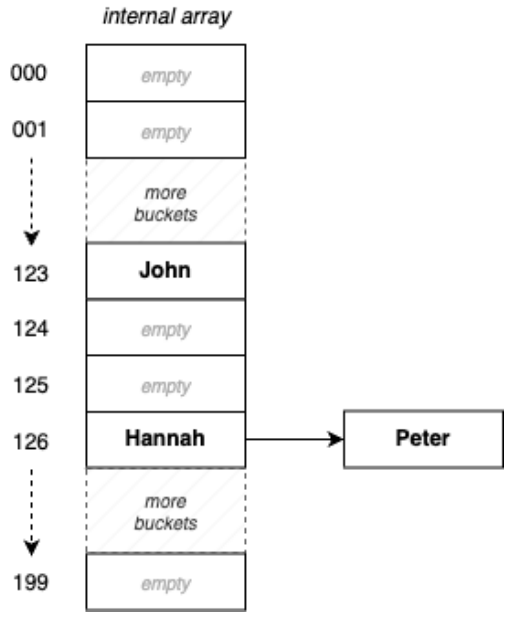

Figure 1: Internal state of the hashtable

Question 1.5 (1 pt.). *What are the two main techniques that "dynamic programming" relies on to help improve recursive algorithms?*

2 Linked-Lists (5 pts.)

Consider the following sketch of a Java implementation of linked-lists.

```
class List<T> {
  private Node<T> head;

  public void delete(T item) { ... }
}

class Node<T> {
```

```
  List next;
  T item;
}
```

Question 2.1 (1.5 pt.)**.** *Provide an iterative algorithm for the* `delete` *operation below. This operation should delete the given item from the list if it exists or left the list unchanged otherwise.*

```
void delete(T item);
```

Question 2.2 (2 pts.)**.** *In the worst case, how many arithmetic and logical operations does your deletion algorithm perform for a linked list containing n nodes? Detail your calculation. Arithmetic and logic operations include:*

- *assignments such as* `x = 23 + y ;`
- *comparisons, such as* $=$, $<$, $\leq$, *ldots ;*
- *logical operations such as* `&&`, `||`, *or* `!` *;*
- *arithmetic operations such as* $+$, $-$, $\times$, *and so on.*

Question 2.3. *Give a* recursive *alternative implementation of this* `delete` *operation.*

3 Emergency Room Triage

In this exercise, we look at hospitals, and emergency departments in particular. When a new patient arrives, a doctor first quickly decides how urgent is the situation, and then records her in the system, including for instance her name, age, address, and priority (a value from 1 to 10, where 1 is the highest priority). The emergency department can thus retrieve the patient with the highest priority and help them as soon as possible.

Your task is to design a data structure to hold this list of patients.

Question 3.1 (1. pt)**.** *Which data structure would you use to maintain the patient by priority and retrieve them by priority efficiently? Justify your choice.*

Question 3.2 (2 pt.)**.** *Explain how the extraction of the patient with the highest priority would work, with the data structure you have selected. Detail the algorithm you would use.*

Question 3.3 (2 pt.)**.** *Explain how the insertion of a patient would work, with your data structure. Detail the algorithm you would use.*

4 Dependency Management

In this exercise, we look at package management systems, such as Maven in Java, NPM for JavaScript, PIP in Python, Cargo in Rust, NuGet in C#, etc.

Sometimes package installation fails because of so-called cyclic dependencies. Here is an example inspired by the Python ecosystem. The libraries "matplotlib", "numpy", and "scipy" are commonly used together in data science projects. "scipy 1.7.0" depends on "numpy 1.20.3" for array processing, and numpy 1.20.3 depends on "matplotlib 3.4.2", which in turn depends on "numpy". This creates a cycle dependency between the four libraries, which can cause issues when installing or updating them, and is nicknamed the "dependency hell".

For the sake of simplicity, we imagine the following API for our package management system:

```
1   public class Package {
2     public String getName() { }
3     public String getVersion() { }
4     public List<Package> getDependencies() { }
5     public boolean equals(Object other) { }
6   }
7
8   public class PackageManager {
9     boolean hasCycle(Package package) {}
10  }
```

The interface `Package` represents the packages a developer install in her environment. Each package is identified by a unique name and a version (here a String for the sake of simplicity). Besides, each package may depend on an arbitrary number of other packages that are its "direct" dependencies.

Your task is to propose an algorithm that detects such cycles in the dependencies.

Question 4.1 (1 pt.)**.** *What data structure does the dependencies form?*

Question 4.2 (2 pts.)**.** *Give an algorithm to detect cycles in the dependencies of a given package. Your algorithm should detect cycles of any length.*

Question 4.3 (2 pts.)**.** *In the worst case, what is the runtime complexity of your algorithm? Explain your reasoning.*