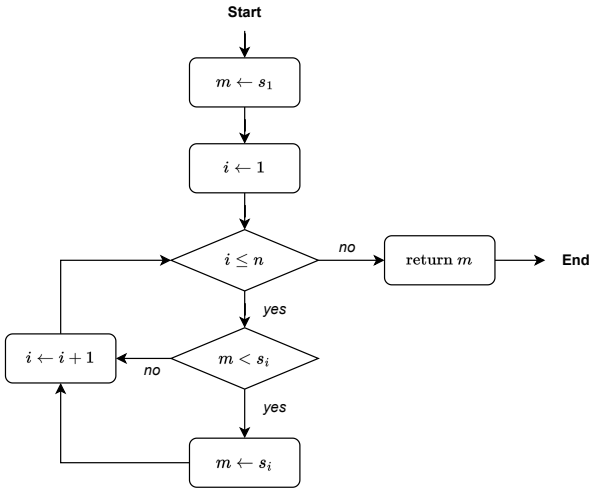
Final Examination

Algorithms & Data structures
NTNU IDATA 2302

December 2023

1 Basic Knowledge (5 points)

Question 1.1 (1 pt.)**.** *Consider the flowchart shown below. It describes an algorithm that accepts a sequence $s = (s_1, \ldots, s_n)$ as an argument and yields one single item. Write the equivalent pseudo-code.*



Question 1.2 (1 pt.)**.** *Using the Big-O notation, is it valid to write the following: $O(n) + O(2n) \in O(n)$? Explain your reasoning.*

Question 1.3 (1 pt.)**.** *Consider the graph G shown below and the tree T aside. Is T a "spanning tree" of G? Explain your reasoning.*

Question 1.4 (1 pt.)**.** *Consider the minimum-heap shown below in Figure 1.4. What is the configuration after one extracts the minimum value 12? Explain your reasoning.*

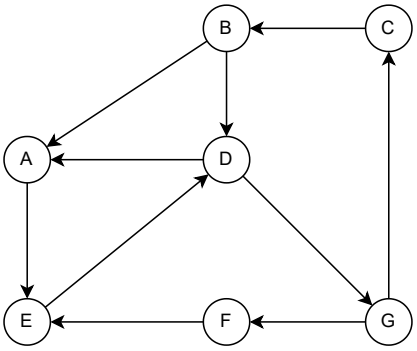


Figure 1: A sample graph G with 7 vertices

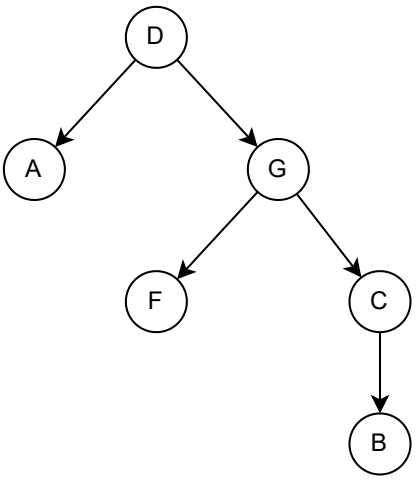


Figure 2: A tree built on G vertices

Figure 3: Graph and spanning Trees

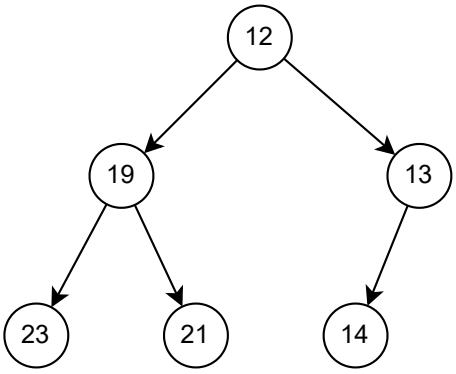


Figure 4: A sample minimum-heap with 6 values

Question 1.5 (1 pt.)**.** *In statement "P = NP" is one of the famous open problem in Computer Science. In this context, what do P and NP stand for?*

2 Reversing a String (5 points)

Let's look at the problem of reversing a given string, that is building the string that contains the same characters but in the opposite order. Here are a few examples:

- "hello" becomes "olleh" ;
- "car" becomes "rac";
- "madam" stays "madam";
- "evil" becomes "live";
- "desserts" becomes "stressed" ;
- etc.

In this exercise, we look at a procedure that accepts a string as input, and returns its reserve (as a new string). Here is what it would look like in Java:

```
1  String reverse(String input) {
2    String reversed = "";
3    int index = input.length - 1;
4    while (index >=0) {
5      reversed += input.charAt(index);
6      index--;
7    }
8    return reversed;
9  }
```

Question 2.1 (2 pt.)**.** *How many operations are performed by this algorithm? Express this number as a function of the length ℓ of the input string.*

Your count of operation should include:

- *comparisons and other logical tests*
- *arithmetic operations such as addition, subtraction, division, multiplication. We consider that adding a character to a string is a valid addition.*
- *variable assignments*

Question 2.2 (2 pt.)**.** *Propose an equivalent recursive algorithm that reverses an input string*

Question 2.3 (1 pt.)**.** *Considering runtime and memory consumption, which of these two approaches (iterative and recursive) would you favor? Why?*

3 Finding the Majority Element

We now turn our attention to the *majority item problem*. Given a sequence $s = (s_1, s_2, \ldots, s_n)$, find the item that occurs more than 50 %.

Consider the following examples:

- The empty sequence $s = ()$ has no majority item.

- The sequence $s = (A, B, C, C, C, B, A)$ does not admit a majority item. The item 'C' occurs only three times (it should occurs at least 4 times to be a majority item).
- The sequence $s = (A, A, A, A, A, B, B)$ has majority item 'A', which occurs 5 times over 7 items (70 %).
- The sequence $s = (A, A, B, B)$ has no majority item. Both A and B occurs exactly 50 %.

This exercise focuses on building an algorithm, which, given a sequence of characters, returns the majority item if any, or null otherwise. In Java, that would look like

```
char findMajorityItem(char[] input) {
  return ...;
}
```

Question 3.1 (2 pt.)**.** *Propose an algorithm to find the majority item.*

Question 3.2 (2 pt.)**.** *What is the runtime complexity of your solution, in the worst case? Explain your reasoning.*

Question 3.3 (1 pt.)**.** *What is the memory complexity of your solution, in the worst case? Explain your reasoning.*

4 The 8-puzzle

The *8-puzzle* is a classic sliding puzzle consisting of a 3×3 grid with eight numbered tiles and one empty space. The goal is to rearrange the tiles from a given initial configuration to a specified goal configuration by sliding the tiles into the empty space. Each tile can move to an adjacent empty space (either vertically or horizontally) in a single move. The puzzle is typically initialized with a random arrangement of the tiles, and the challenge lies in finding a sequence of moves that leads to the desired configuration. Figure 5 illustrates a few possible moves from a given configuration.

Players aim to reach the goal state, usually a specific numerical order, by strategically sliding tiles one at a time until the entire grid is ordered accordingly. The puzzle's complexity arises from the vast number of possible configurations and the limited movements allowed.

Question 4.1 (1 pt.)**.** *How many configurations are possible for a 8-puzzle? Explain your reasoning.*

Question 4.2 (1 pt.)**.** *Consider the initial and target configurations shown below on Figure 6. What is the sequence of move to reach the "target" configuration?*

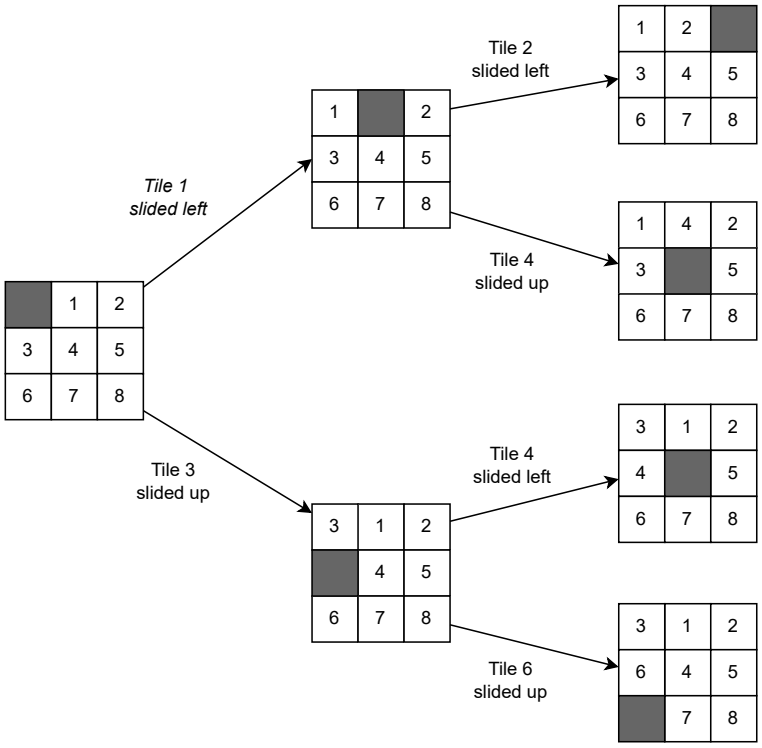


Figure 5: Example of 8-puzzle configurations with transitions from one configuration to the next

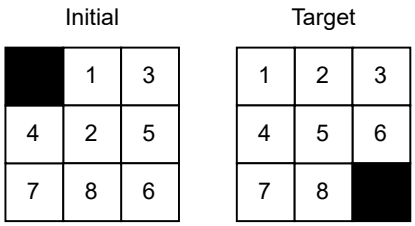


Figure 6: Two 8 puzzle configurations

Question 4.3 (3 pt.)**.** *Outline an algorithm to check whether a given "target" position is reachable from a given "initial" configuration.*

Assume for instance the following interface for a configuration, with methods to compare configuration and compute what configurations can be reached (in one move) from the current one.

```
interface Configuration {

  /**
   * @return true if the configuration are the same
   */
  boolean equals(Configuration other);

  /**
   * @return the set of configuration that can be reached by moving any
   * tile into the free position.
   */
  Set<Configuration> neighbors();

}
```