Solutions to example exam 1
1. Describe how binary search works. Describe its main idea, what kind of data it can be used on, its time complexity and its pseudo code. Give an example of how the algorithm works.
Question 1 - Solution

Idea:
- Look at the data in the middle of the collection
  - if it is the same as the data we are looking for
    → return the position

- if it is bigger than the data we are looking for
  → look in the left part of the collection

- if it is smaller than the data we are looking for
  → look in the right part of the collection

Use:
Can be used on sorted collections.

Time complexity:
$O(\log n)$
Pseudo code – recursive version

```plaintext
int binarySearch(array, target, low, high)
  //RECUSION STEP
  IF array[mid] > target THEN
    binarySearch(array, target, low, mid -1)
  //RECUSION STEP
  IF array[mid] <= target THEN
    binarySearch(array, target, mid, high)

  // BASE CASE
  IF array[low] = target THEN RETURN low
  ELSE RETURN -1
```
Pseudo code – iterative version

```plaintext
int binarySearch(array, target)
    low = 0
    high = array.length
    WHILE(high – low > 0)
        mid = (low + high + 1) / 2
        IF array[mid] > target THEN high = mid – 1
        ELSE low = mid

    IF array[low] = target THEN RETURN low
    ELSE RETURN -1
```
Binary search

Example:

Search for the number 15.

| 1 | 2 | 15 | 20 | 50 | 60 | 70 | 80 | 90 |
Binary search

Example:

Search for the number 15.

15 == 50?
Binary search

Example:

Search for the number 15.

15 == 50? No
Binary search

Example:

Search for the number 15.

15

1 2 15 20 50 60 70 80 90

15 == 50? No

50 > 15?
Binary search

Example:

Search for the number 15.

15

1  2  15  20  50  60  70  80  90

15 == 50? No

50 > 15? Yes
Binary search

Example:

Search for the number 15.

(array[(0 + 3) / 2] = array[1])  ==  15?
Binary search

**Example:**

Search for the number 15.

\[
\text{array}[(0 + 3) / 2] = \text{array}[1] = 15? \quad \text{No}
\]

\[
\text{array}[(0 + 3) / 2] = \text{array}[1] > 15? 
\]
Binary search

Example:

Search for the number 15.

15

1 2 15 20 50 60 70 80 90

(array[(0 + 3) / 2] = array[1]) == 15? No

(array[(0 + 3) / 2] = array[1]) > 15? No
Binary search

Example:

Search for the number 15.

\[\text{array}\left[\frac{2+3}{2}\right] = \text{array}[2]\] == 15? Yes

return index 2
2. Write the code for a class that implements the interface \texttt{(Interface2)} below and contains at least two own methods.

\begin{verbatim}
public interface Interface2 {
    public String getDescription();
}
\end{verbatim}
public class Uppgift2 implements interface22 {
    private String name = "";
    private String address = "";

    // constructor
    public Uppgift2(String n, String a) {
        name = n;
        address = a;
    }
}
public class Uppgift2 implements interface22 {
    private String name = "";
    private String address = "";

    //setter
    public void setName(String n) {
        name = n;
    }

    //setter
    public void setAddress(String a) {
        address = a;
    }
}
Question 2 – Suggested solution

public class Uppgift2 implements interface2 {
    private String name = "";
    private String address = "";

    //getter
    public String getName() {
        return name;
    }

    //getter
    public void getAddress() {
        return address;
    }
}
public class Uppgift2 implements interface22 {
    private String name = "";
    private String address = "";

    // method from the interface
    public String getDescription() {
        return "name: " + name +
            "address: " + address;
    }
}
Question 2 – Suggested solution

```java
public class Uppgift2 implements interface22 {
    private String name = "";
    private String address = "";

    //method
    public String addName(String n) {
        Name += " " + n;
    }

    //method
    public String addCity(String city) {
        address += " " + city;
    }
}
```
3. Explain the concept of **inheritance** and provide a code example.

Solution:

Inheritance means that we create new classes, **subclasses**, from existing classes, **superclasses**, and extends them with new methods and instance variables.
3. Explain the concept of inheritance and provide a code example.

Solution:

```java
public class MySuperClass {
    // instance variable
    int size = 1;

    // constructor
    public MySuperClass(int s) {
        size = s;
    }
}
```
3. Explain the concept of **inheritance** and provide a code example.

**Solution:**

```java
public class MySuperClass {
    // instance variable
    int size = 1;

    //setter
    public int setSize(int s) {
        Size = s;
    }

    //getter
    public int getSize() {
        return size;
    }
}
```
3. Explain the concept of \textit{inheritance} and provide a code example.

\textbf{Solution:}

```java
public class MySuperClass {
    // instance variable
    int size = 1;

    // method
    public void printSize() {
        System.out.println(size);
    }
}
```
3. Explain the concept of inheritance and provide a code example.

Solution:

```java
public class SubClass1 extends MySuperClass {
    // default constructor
    public SubClass1() {}

    // method
    public int increaseSize(int i) {
        size += i;
    }
}
```
4. What will the program below print? Explain the examples 5 and 10 in detail.

```java
class TestRegularExpression {
    private static void tryReg(String ex, String regexp, String text) {
        if (text.matches(regexp)) {
            System.out.println(ex + " matchar.");
        } else {
            System.out.println(ex + " matchar INTE.");
        }
    }
}
```
public static void main(String[] args) {
    tryReg("Exempel 1" , "a?b.*", "bb");
    tryReg("Exempel 2" , "(ac)+", "acac");
    tryReg("Exempel 3" , "ab*", "abbbbbbbbb");
    tryReg("Exempel 4" , "ab*", "abab");
    tryReg("Exempel 5" , "a?b*", "abab");
    tryReg("Exempel 6" , "a?b*", "bbbbbbbb");
    tryReg("Exempel 7" , "a+b*", "abbbbbbbb");
    tryReg("Exempel 8" , "((aa)|(bb)).*c.*", "aaaacc");
    tryReg("Exempel 9" , "((aa)|(bb)).*c.*", "bbaacc");
    tryReg("Exempel 10" , "((aa)|(bb)).*c.*", "baacc");
}
}
Question 4 – Solution

tryReg("Exempel 1", "a?b.*", "bb");

Printout:
Exempel 1 matchar.

Explanation:
a?    - option, a may exist
b     - b must exist
.*    - something might exist
Question 4 – Solution

```csharp
tryReg("Exempel 2" , "(ac)+", "acac");
```

**Printout:**
Exempel 2 matchar.

**Explanation:**
(ac)+ - The string must consist of 1 or more “ac”
Question 4 – Solution

```
tryReg("Exempel 3", "ab*", "abbbbbbbbb");
```

Printout:
Exempel 3 matchar.

Explanation:
- a - The string must begin with a
- b* - must exist 0 or more b
Question 4 – Solution

```javascript
tryReg("Exempel 3", "ab*", "abbbbbbbbbb");
tryReg("Exempel 4", "ab*", "abab");
```

Printout:
Exempel 3 matchar.
Exempel 4 matchar INTE.

Explanation:
a - The string must begin with a
b* - must exist 0 or more b
Question 4 – Solution

```java
tryReg("Exempel 5", "a?b*", "abab");
tryReg("Exempel 6", "a?b*", "bbbbbbbb");
```

Printout:
Exempel 5 matchar INTE.
Exempel 6 matchar.

Explanation:
a? - option, the string may begin with a
b* - the rest of the string must consist of 0 or more b

"abab" has an a between the bs
→ the string does not match the regular expression "a?b*"
Question 4 – Solution

tryReg("Exempel 7", "a+b*", "abbbbbbb");

Printout:
Exempel 7 matchar.

Explanation:
a?    - option, the string may begin with a
b*     - the rest of the string must consist of 0 or more b
Question 4 – Solution

tryReg("Exempel 8", "((aa)|(bb)).*c.*", "aaaacc");
tryReg("Exempel 9", "((aa)|(bb)).*c.*", "bbaacc");
tryReg("Exempel 10", "((aa)|(bb)).*c.*", "baacc");

Printout:
Exempel 8 matchar.
Exempel 9 matchar.
Exempel 10 matchar INTE.

Explanation:
(aa|bb)  - the string must begin with aa or bb
.*  - the string continues with something
    c  - the string continues with c
 .*  - the string ends with something

The string “baacc” does not begin with aa or bb
→ the string does not match the regular expression
"((aa)|(bb)).*c.*"
5a. Which strings with length less than four match this regular expression: "ab?c*".
5b. Which strings with length less than five match this regular expression: "a*+b"
5c. Which strings with length less than four match this regular expression: "(a*bc|c*d?e)+"
5a. Which strings with length less than four match this regular expression: "ab?c*".

Solution:

Length 1: a
Length 2: ab, ac
Length 3: abc, acc
5b. Which strings with length less than five match this regular expression: "a*+b"

Solution:

Length 1: b
Length 2: ab
Length 3: aab
5c. Which strings with length less than four match this regular expression: 
"(a*bc|c*d?e)+"

Solution:

Length 1: e
Length 2: bc, de, ee
Length 3: abc, bce, cde, cce, dee, eee, ebc, ece, ede
6. Create a class for simple linked lists of strings (String) without using the Java standard classes for lists. In the class, there should be a class for the nodes that are linked together in the list. Explain why you are creating the class as you do!
A simple linked list consists of nodes. Each node contains a string and a pointer to the next node.
Question 6 - Solution

A simple linked list consists of nodes. Each node contains a string and a pointer to the next node.

```java
private class Node {
    String data;
    Node next;

    Node(int v, Node n) {
        data = v;
        next = n;
    }
}
```

![Diagram of a linked list with nodes containing strings and pointers](image)
Question 6 - Solution

public class MyLinkedList{
    // The private class Node keeps track of
    // the nodes in the list. The classes
    // that uses MyLinkedList does not have
    // to know how MyLinkedList is constructed.

    private class Node {
        String data;
        Node next;

        public Node(E d, Node n) {
            data = d;
            next = n;
        }
    }

    // This instance variable points to the first
    // node in the list.
    Node start;
}
public class MyLinkedList{
    // This method adds new strings last in the
    // list. This method should be called
    // when we want to add a string to the list.

    public void add(String s){
        if(start == null) {
            start = new Node(s, start);
        } else {
            Node currentNode = start;
            while (currentNode.next != null) {
                currentNode = currentNode.next;
            }
            currentNode.next = new Node(s, start);
        }
    }
}
public class MyLinkedList{
    // This method returns at which position
    // the string s occurs in the list.
    // This method is useful if we want to know
    // if a string s already exists in the list
    // before we add it.
    public int search(String s){
        // code in lab 1
    }
}

// This method removes all occurences of the
// string s from the list. This method is
// useful if s exists in the least and we
// want to remove it.
public void delete(String s){
    // code in lab 1
}
7. Write the code for a class that implements the interface `Interface7`. The method `decOrder()` examines whether the integers (int) in an array is ordered from the larger number to the smaller number and returns true if that is the case and false otherwise.

```java
public interface Interface7 {
    public boolean decOrder(int[] array);
}
```
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if (array.length == 0 || array.length == 1) return true;
        for (int i = 0; i < array.length - 1; i++)
            if (array[i + 1] > array[i]) return false;
        return true;
    }
}

array: 4 4 3 1
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: 4 4 3 1
public class Uppgift7 implements Inteface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: 4 4 3 1
Question 7– Solution – Execution1

```java
public class Uppgift7 implements Inteface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}
```

array: 4 4 3 1
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: 4 4 3 1
public class Uppgift7 implements Inteface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: 4 4 3 1

i  i+1
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: 4 4 3 1
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array:  

|   4   |   4   |   3   |   1   |
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;
        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;
        return true;
    }
}

array: [4, 4, 3, 1]
Question 7– Solution – Execution1

```java
public class Uppgift7 implements Inteface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}
```

array: 4 4 3 1
public class Uppgift7 implements Inteface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: 4 5 3 1
public class Uppgift7 implements Inteface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: 4 5 3 1
Question 7– Solution – Execution 2

```java
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}

array: [4, 5, 3, 1]
```
Question 7– Solution – Execution2

```java
public class Uppgift7 implements Interface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1)
            return true;

        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;

        return true;
    }
}
```

array: 4 5 3 1

\[ i \quad i + 1 \]
public class Uppgift7 implements Inteface7 {
    public boolean decOrder(int[] array) {
        if(array.length == 0 || array.length == 1) return true;
        for(int i = 0; i < array.length-1; i++)
            if(array[i + 1] > array[i])
                return false;
        return true;
    }
}

array: 4 5 3 1

i i + 1
8. Explain how a binary search tree works. Give an example of what you can use them for.
Binary search tree - Example

key

1 2 3 4 5 6 7 8 9
Binary search tree - Example

key

1 2 3 4 5 6 7 8 9

key == 5? Done
key < 5? Search in the left child
key > 5? Search in the right child
Binary search tree - Example

key

key == 5? Done
key < 5? Search in the left child
key > 5? Search in the right child
Binary search tree - Example

key == 3? Done
key < 3? Search in the left child
key > 3? Search in the right child
Binary search tree - Example

key == 3? Done
key < 3? Search in the left child
key > 3? Search in the right child
Binary search tree - Example

key == 2? Done
key < 2? Search in the left child
key > 2? Search in the right child
Binary search tree - Example

key

key == 8? Done
key < 8? Search in the left child
key > 8? Search in the right child
Binary search tree - Example

key == 8? Done
key < 8? Search in the left child
key > 8? Search in the right child
Binary search tree - Example

key

key == 7? Done
key < 7? Search in the left child
key > 7? Search in the right child
Binary search tree - Example

```
    5
   / \
  3   8
 /   /\n2   4   7
 / \   \ 
1   6   9
```
Binary search tree - ADT

- Binary search tree consists of nodes with this contents:
  - key
  - data
  - pointer to the left child
  - pointer to the right child

```
key
data
```

```
Left child
         Right child
```
Binary search tree - ADT
ROOT – The node at the top of the tree

Internal nodes – Nodes that has one parent and at least one child

LEAFS(TERMIAL) – The nodes at the bottom of the tree
ROOT – The node at the top of the tree

Internal nodes – Nodes that has one parent and at least one child

LEAFS(TERMINALS) – The nodes at the bottom of the tree
ROOT – The node at the top of the tree

Internal nodes – Nodes that has one parent and at least one child

LEAFS(TERMINAL) – The nodes at the bottom of the tree
Binary search tree - Theory

How many comparisons do we need until we have found the right key?

**Answer:** In the worst case we have to travel through the tree until we have arrived in a leaf.
Binary search tree - Theory

How many comparisons do we need until we have found the right key?

Answer: In the worst case we have to travel through the tree until we have arrived in a leaf.

Note:
- If the tree is **unbalanced** then we need $O(n)$ comparisons.
- If the tree is **balanced** then we need $O(log(n))$ comparisons.
Unbalanced tree - Example

Number of nodes: 9
Height: 9
Search time: $O(n)$
Balanced tree - Example

Number of nodes: 9
Height: 4
Search time: $O(\log(n))$
Binary search tree – Use

The keys in a binary tree are stored in a sorted order and if the tree is balanced, then we can improve the search time to $O(\log (n))$. 