**WIA2005 Algorithm Design &Analysis**
**Semester 2, 2016/17**
**Lab 1**

---

**PART A: Program the solution and try to estimate the time complexity of each code**

1. Given an array of ints length 3, return the sum of all the elements.

   sum3([1, 2, 3]) → 6
   sum3([5, 11, 2]) → 18
   sum3([7, 0, 0]) → 7

2. Given an array of ints, return true if 6 appears as either the first or last element in the array. The array will be length 1 or more.

   firstLast6([1, 2, 6]) → true
   firstLast6([6, 1, 2, 3]) → true
   firstLast6([13, 6, 1, 2, 3]) → false

3. Create a program, print out all even numbers between 1-100. Calculate the time complexity of the program.

4. Using a linear search, write a program that takes in an integer and search an array of size 100 that contain random numbers between 1- 1000. If the search is successful (number is in the generated array) the output will be the array index number containing the integer; else display "Number is not in the array". Calculate the time complexity of the program.

5. Create an iterative and recursive method to compute the first n numbers in the Fibonacci sequence. Calculate the time complexity of the program.

6. Given an array length, return an array with the elements "rotated left". Example of array of length 3, {1, 2, 3} yields {2, 3, 1}.
   rotateLeft3([1, 2, 3]) → [2, 3, 1]
   rotateLeft3([5, 11, 9]) → [11, 9, 5]
   rotateLeft3([7, 0, 0]) → [0, 0, 7]

7. Start with 2 int arrays, a and b. Consider the sum of the values in each array. Return the array which has the largest sum. In event of a tie, return a. Example below for array of length 2.

   biggerTwo([1, 2], [3, 4]) → [3, 4]
   biggerTwo([3, 4], [1, 2]) → [3, 4]
   biggerTwo([1, 1], [1, 2]) → [1, 2]

**PART B**

**Introducing Java Map API**

```
// Make a new empty map
Map<String, String> map = new HashMap<String, String>();
```

map.get(*key*) -- retrieves the stored value for a key, or null if that key is not present in the map.map.put(*key*, *value*) -- stores a new key/value pair in the map. Overwrites any existing value for that key.
map.containsKey(*key*) -- returns true if the key is in the map, false otherwise.
map.remove(*key*) -- removes the key/value pair for this key if present. Does nothing if the key is not present.

1. Modify and return the given map as follows: if the key "a" has a value, set the key "b" to have that value, and set the key "a" to have the value "". Basically "b" is a bully, taking the value and replacing it with the empty string.

   mapBully({"a": "candy", "b": "dirt"}) → {"a": "", "b": "candy"}
   mapBully({"a": "candy"}) → {"a": "", "b": "candy"}
   mapBully({"a": "candy", "b": "carrot", "c": "meh"}) → {"a": "", "b": "candy", "c": "meh"}

2. Given an array of strings, return a Map<String, Integer> containing a key for every different string in the array, always with the value 0. For example the string "hello" makes the pair "hello":0. We'll do more complicated counting later, but for this problem the value is simply 0.

   word0(["a", "b", "a", "b"]) → {"a": 0, "b": 0}
   word0(["a", "b", "a", "c", "b"]) → {"a": 0, "b": 0, "c": 0}
   word0(["c", "b", "a"]) → {"a": 0, "b": 0, "c": 0}