Learning goals

Before the next day, you should have achieved the following learning goals:

- Strengthen your understanding of how recursion works
- Understand that storing intermediate values for reuse (memoization) can largely speed up some computations.
- Implement divide-and-conquer solutions to problems.

1 Memoized Fibonacci

Write a Java class with a with a static method that calculates the nth element of the Fibonacci sequence as seen in the notes.

Do it with and without memoization. Compare the time that is needed in each case to find F(40) or F(45).

2 Anagrams

An anagram is a word created by recombination of the letters in another words. Some anagrams make sense ("silent", "listen") and some do not ("nietsl"). Some people only accepts "real" anagrams, but in this exercise we will accept them all, even if they do not exist as real words.

Write a class with a static method that takes a String as a parameter and returns a list (hint: you can use `List<String>` and `ArrayList<String>`) of strings with all the anagrams that can be made with it.

Is it easy to do this both iteratively and recursively? Is this similar to a former exercise?

3 Hailstone numbers

The sequence hailstone numbers is defined as follows:

- If the number n is even, the next number is $\frac{n}{2}$
- If the number is odd, the next number is $3n + 1$

It is thought that this sequence always converges to 1 (this is the Collatz conjecture, not demonstrated yet).

Write a method that returns a list of integers (hint: you can use `List<Integer>` and `ArrayList<Integer>`) with the sequence of hailstone numbers that starts at some given natural number provided by the user.

4 Binary search

The most basic example of divide-and-conquer strategies is the binary search. This is used to look for an element *in a sorted list.*

We can find an element in a list by traversing through the whole list and checking whether each element is the one we are looking for. The number of comparisons that we need by using this algorithm is proportional to the length of the list. If we know that the list is sorted, we can do better with a divide-and-conquer strategy, like the one defined by repeating these steps:

Initial action: If the list is empty, it does not contain the element and we have finished. If it is not empty, check the middle element, i.e. the element at `list.size()/2`. If it is the element we are looking for, we have finished.

Subproblem: If the element we are looking for is before the middle element, the next list to search into is the first half of the original list; otherwise, it is the second half.

Integration: No need for integration in this case. Just repeat looking into half-lists until the sublist is only one element long At that point, either the element is the one we are looking for or the list does not contain it.

Implement a binary search algorithm for a list of integer numbers. The method returns true if the list contains the element and false otherwise.

You can use the classes in the Java Collection Library. Search for numbers in lists with 10, 100, and 1000 elements; how many comparisons do you need in each case? (Hint: instead on entering manually 1000 elements in order, maybe you can implement one of the sorting algorithms in the following exercises and then use them to order a list of random numbers. Remember that you can create a random integer between 0 and N-1 with `Math.abs(N * Math.random())`).

5 Mergesort

Mergesort is a popular sorting algorithm that employs a divide-and-conquer strategy. You can implement a Mergesort for lists by following the following steps:

Subproblem: If the list contains 0 or 1 element, it is sorted and you can return it. If not, then divide the list into two lists of the same length (± 1). Then sort the lists (i.e. apply mergesort to each sublist).

Integration: When both sublists are returned sorted, check the first element of both sublists; remove the one that comes first (e.g. the lowest integer of the two) and add it to the result list. Repeat this process until all elements in both sublists have been added to the result list. Return the result list.

Example with five elements:

```
      [3, 7, 2, 9, 1]
   [3, 7, 2]      [9, 1]         (subproblem)
[3, 7]    [2]     [9] [1]        (subproblem)
[3] [7]   [2]     [9] [1]        (subproblem)
[3, 7]    [2]     [9] [1]        (integration of [3] and [7])
  [2, 3, 7]      [9] [1]         (integration of [3, 7] and [2])
  [2, 3, 7]      [1, 9]          (integration of [9] and [1])
      [1, 2, 3, 7, 9]            (integration of [2, 3, 7] and [1, 9])
```

6 Quicksort

Quicksort is another sorting algorithm that also employs a divide-and-conquer strategy. It works well in most usual computers because of the low-level interactions between registers and the main memory, which make it very popular and widely used.

You can implement a Mergesort for lists by following the following steps:

Initial action: If the list contains 0 or 1 element, it is sorted and you can return it. Otherwise, choose one element as "pivot" (usually the first one).

Subproblem: Divide the list into two lists: the first list contains the elements before the pivot (e.g. the integers lower than the pivot) and the second one contains the elements after the pivot. Then sort both lists (i.e. apply quicksort to each sublist, choosing a new pivot, etc).

Integration: When both sublists are returned sorted, simply concatenate them (first list, then pivot, then second list) and return the result.

Example with five elements:

```
      [3, 7, 2, 9, 1]
   [2, 1]   3   [7, 9]           (subproblem, pivot: 3)
[1] 2 []    3   [7, 9]           (subproblem, pivot: 2)
[1] 2 []    3   [] 7 [9]         (subproblem, pivot: 7)
  [1,2]     3   [] 7, [9]        (integration of the sublists of pivot 2)
  [1,2]     3    [7, 9]          (integration of the sublists of pivot 7)
     [1,2, 3, 7, 9]              (integration of the sublists of pivot 3)
```

7 Finding the roots of a polynomial (*)

Polynomials are a special type of mathematical function with a lot of applications in mathematics and computer science —from scientific calculations to games rendering— because they have a lot of nice properties. For instance, they are always continuous and infinitely derivable for any real number, and can be used to approximate other functions. A polynomial is a function of the form:

$$a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n$$

where the a_i are called *coeficients* and the x is a variable. A polynomial can easily be represented by an array containing the a_i coeficients. For example, the array `[5, 0, 3, -2]` represents the polynomial:

$$5 + 0 \cdot x + 3x^2 - 2x^3 = 5 + 3x^2 - 2x^3$$

When working with polynomials, it is common to search for their *roots*, i.e. the values of x that make them evaluate to zero. For instance, the polynomial $6 - 5x + x^2$ has roots at $x = 2$ and at $x = 3$:

$$6 - 5x + x^2|_{x=2} = 6 - 5 \cdot 2 + 2^2 = 6 - 10 + 4 = 0$$
$$6 - 5x + x^2|_{x=3} = 6 - 5 \cdot 3 + 3^2 = 6 - 15 + 9 = 0$$

Create a method that takes an array (minimum size 2) representing a polynomial and find whether it has a root in the range from -1000.0 to +1000.0. Use binary search to find the root knowing that:

- A polynomial of grade n (i.e. array length n+1) has at most n real numbers as roots (but they may be out of the [-1000, 1000] range).
- If a polynomial has a positive value at $x = A$, and a negative value at $x = B$ (or viceversa), then it has at least one root in the interval [A,B]. In other words, if we know that it is continuous, and it turns from positive to negative or viceversa, then we know that there is at least one point in which it is zero.

Find the root with a precision of six decimals. In other words, assuming that you have a method `eval(double)` that evaluates the polynomial for a certain value of x, and a `static final` variable called `PRECISION` with value `0.000001`; if:

```
Math.abs((polynomial(x) - 0) < PRECISION)
```

then `x` is a root of the polynomial. Remember that rounding errors make it unlikely that any number will evaluate to exactly 0.0, so we can only search for double numbers up to a certan precision

Different applications need different precisions. GPS positioning has a precision of a few meters, the design of a plane usually works with precision up to one milimeter, and the measurements for the latests scientific experiments in quantum physics require the best precision that is achievable by state-of-the-art technology.