CIT CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

LINEAR DATA STRUCTURES AND ALGORITHMS.

ASSIGNMENT 2: ALGORITHMS

BACKGROUND.

In this assignment we are going to implement **divide&conquer** and **greedy**-based algorithms for solving different problems.

Note: The exercises proposed in this assignment are related to the exercises seen in the lectures. Thus, we strongly recommend to download, get to understand, run and debug the code examples of the lectures before start attempting the exercises of the assignment.

ASSIGNMENT 2 – HINT 1 **(Week 8)**

Divide and Conquer: First set of exercises.

BACKGROUND.

The folder **/src** contains the following files:

- **(MyList.java, MyStaticList.java, MyNode.java, MyDynamicList.java):** These classes stand for the package MyList<T> we have seen previously in the lectures of the Block II: Data Structures.
- **DivideAndConquerAlgorithms.java:** This class contains the proposed divide&Conquer functions you have to implement.
- **MyMain.java:** This class tests the functionality of the divide&Conquer functions.

The folder **/doc** contains the documentation of the project. In particular:

- **(MyList.html, MyStaticList.html, MyNode.html, MyDynamicList.html):** Contains the description of the package MyList<T> classes.
- **DivideAndConquerAlgorithms.html:** Contains the description of the class DivideAndConquerAlgorithms.java.
- **MyMain.html:** Contains the description of the class MyMain.java.

EXERCISE.

Implement the following functions of the class DivideAndConquerAlgorithms.java.

1. `public int maxInt(MyList<Integer> m);`
   The function returns the maximum item of m (-1 if m is empty).

2. `public boolean isReverse(MyList<Integer> m);`
   The function returns whether *m* is sorted in decreasing order or not.

3. `public int getNumAppearances(MyList<Integer> m, int n);`
   The function returns the amount of times that the integer *n* appears in *m*.

4. `public int power(int n, int m);`
   The function returns n^m.

5. `public int lucas(int n);`

   Mathematically, the Lucas series is defined as:

$$L_n := \begin{cases} 2 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ L_{n-1} + L_{n-2} & \text{if } n > 1. \end{cases}$$

   Thus, the Lucas series is as follows:

   2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123 …………..

The function returns the n-est item of the lucas series.
Examples: *lucas(0) → 2, lucas(4) → 7*

6. `public void drawImage(int n);`

The function prints prints a pattern of a given length.

```
*
**
***
...
```

ASSIGNMENT 2 – HINT 2 **(Week 9)**
Divide and Conquer: Second set of exercises.

BACKGROUND.

The folder **/src** contains the following files:

- **(MyList.java, MyStaticList.java, MyNode.java, MyDynamicList.java):** These classes stand for the package MyList<T> we have seen previously in the lectures of the Block II: Data Structures.
- **DivideAndConquerAlgorithms.java:** This class contains the proposed divide&Conquer functions you have to implement.
- **MyMain.java:** This class tests the functionality of the divide&Conquer functions.

The folder **/doc** contains the documentation of the project. In particular:

- **(MyList.html, MyStaticList.html, MyNode.html, MyDynamicList.html):** Contains the description of the package MyList<T> classes.
- **DivideAndConquerAlgorithms.html:** Contains the description of the class DivideAndConquerAlgorithms.java.
- **MyMain.html:** Contains the description of the class MyMain.java.

EXERCISE.

Implement the following functions of the class DivideAndConquerAlgorithms.java.

7. `public void recursiveDisplayElements(MyList<Integer> m);`
   Given a MyList, this recursive algorithm displays its elements by screen (if any).

8. `public MyList<Integer> smallerMyList(MyList<Integer> m, int e);`
   The function filters all elements of MyList being smaller than 'e'.

9. `public MyList<Integer> biggerEqualMyList(MyList<Integer> m, int e);`
   The function filters all elements of MyList being bigger or equal than 'e'.

10.
```
public MyList<Integer> concatenate(MyList<Integer> m1,
                                   MyList<Integer> m2);
```
   The function computes a new lists whose content is the concatenation of m1 and m2.

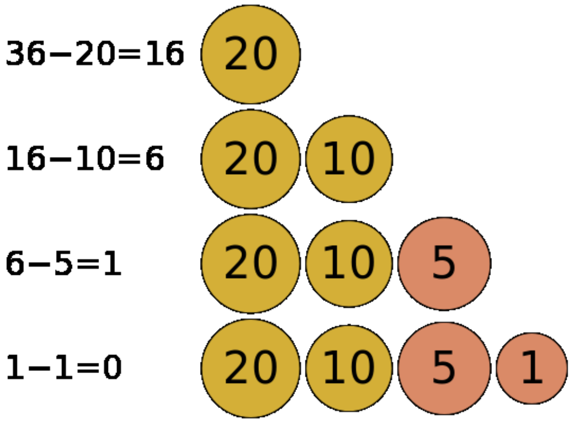
11. `public MyList<Integer> quickSort(MyList<Integer> m);`
   Given a concrete MyList, it computes a new sorted list using the method Quick Sort.

ASSIGNMENT 2 – HINT 3 **(Weeks 10 and 11)**
Greedy Algorithms

BACKGROUND.

The *change-making problem* addresses the question of finding the minimum number of coins (of certain denominations) that add up to a given amount of money. It is a knapsack type problem, and has applications wider than just currency.

Greedy algorithms determine minimum number of coins to give while making change. These are the steps a human would take to emulate a greedy algorithm to represent 36 cents using only coins with values {5, 20, 1, 10}:



The folder **/src** contains the following files:

- **(MyList.java, MyStaticList.java, MyNode.java, MyDynamicList.java):** These classes stand for the package MyList<T> we have seen previously in the lectures of the Block II: Data Structures.
  - **ChangeMaking_1.java:** This class contains the proposed problem you have to implement with a greedy algorithm.
    The selection function uses a naïve policy: Just pick the first non-discarded type of coin.
  - **ChangeMaking_2.java:** This class contains the improved solution to the proposed problem you have to implement with a greedy algorithm.
    The selection function uses a more elaborated policy: Pick the biggest non-discarded type of coin.
- **MyMain.java:** This class tests the functionality of the greedy algorithms.

The folder **/doc** contains the documentation of the project. In particular:

- **(MyList.html, MyStaticList.html, MyNode.html, MyDynamicList.html):** Contains the description of the package MyList<T> classes.
- **ChangeMaking_1.html,:** Contains the description of the class DivideAndConquerAlgorithms.java.
- **ChangeMaking_2.html,:** Contains the description of the class DivideAndConquerAlgorithms.java.
- **MyMain.html:** Contains the description of the class MyMain.java.

EXERCISE.

Implement the following functions of the class ChangeMaking_1.java.

```
1. public int getCandidate(int changeGenerated,
                           MyList<Integer> discarded,
                           MyList<Integer> coinValues);
```

Basic policy: Just pick the first non-discarded type of coin.

```
2. public boolean isValid(MyList<Integer> coinValues,
                          int amount,
                          int changeGenerated,
                          int itemSelected);
```

```
3. public boolean isFinal(int changeGenerated,
                          MyList<Integer> discarded,
                          MyList<Integer> coinValues,
                          int amount);
```

```
4. public MyList<Integer> getQuality(MyList<Integer> sol,
                                     int changeGenerated,
                                     int amount);
```

```
5. public MyList<Integer> solve(MyList<Integer> coinValues,
                                int amount);
```

Implement the following functions of the class ChangeMaking_2.java.

```
6. public int getCandidate(int changeGenerated,
                           MyList<Integer> discarded,
                           MyList<Integer> coinValues);
```

More elaborated policy: Pick the biggest non-discarded type of coin.

The rest of the functions can be directly re-used from the class ChangeMaking_1.java.