



## **LINEAR DATA STRUCTURES AND ALGORITHMS.**

### **ASSIGNMENT 2: ALGORITHMS**

**IMPORTANT:** The exercises proposed in this assignment are related to the exercises studied in the lectures. Thus, it is strongly recommended to download, get to understand, run and debug the code examples of the lectures before start attempting the exercises of the assignment.

#### **BACKGROUND.**

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

## ASSIGNMENT 2 – PART 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-th item of the lucas series.  
Examples: *lucas(0)*  $\rightarrow$  2, *lucas(4)*  $\rightarrow$  7

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

The function prints a pattern of a given length.

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

## ASSIGNMENT 2 – PART 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 3 – PART 3





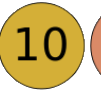




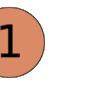
(Week 10)

### 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](#). Note that there is no limit on how many coins can be returned from each type of coin. These are the steps a human would take to emulate a greedy algorithm to represent 36 cents using only coins with values {1, 5, 10, 20}:

Accuracy:	Coins:	Number Coins:
$36 - 20 = 16$		1
$16 - 10 = 6$	 	2
$6 - 5 = 1$	  	3
$1 - 1 = 0$	   	4

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.java:** This class contains the proposed Greedy algorithm that you have to implement. The algorithm solves the problem of change making described above.
- **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.html:** Contains the description of the class `ChangeMaking.java`.
- **MyMain.html:** Contains the description of the class `MyMain.java`.

## EXERCISE.

Implement the functions of the class ChangeMakingJava.java according to the schema of greedy algorithms studied in the lectures.

1. `selectionFunctionFirstCandidate:`  
Basic policy: Just pick the first non-discarded type of coin.
2. `selectionFunctionBestCandidate:`  
More elaborated policy: Pick the biggest non-discarded type of coin.
3. `feasibilityTest`
4. `solutionTest`
5. `objectiveFunction`
6. `solve:`

This function calls to the above functions in order to solve the problem. The selection function selected is `selectionFunctionFirstCandidate` if *typeSelectFunc=1*. Instead, if *typeSelectFunc=2*, `selectionFunctionBestCandidate` is the selected function.

This function is the responsible of printing on the screen the output of the problem (see Figure above as example of output):

- Accuracy, which is the difference of amount of change provided change provided minus the target amount of money (input of the function).
- Coins that are provided as change
- Number of coins provided as change.

## MARK BREAKDOWN.

Assignment 2: 25 marks.

- Part 1: 9 marks (1.5 marks each function)
- Part 2: 6 marks (1.5 marks each function)
- Part 3: 10 marks

To evaluate each function I will run it over some tests (do not forget to run the main files and check that the outputs are correct!). The results are based on the performance of your code over these tests. Also, remember to print the error messages.

For each function, there are 4 possible scenarios:

- A. The function passes all the test, it is efficient and prints error messages → 100% of marks.
- B. The function does not pass all tests by small/medium mistakes → 90% of marks – 10% marks (depending how small is the mistake).
- C. The function does not pass all tests by big mistakes, it does not compile or it generates and exception (makes the program crash) or the function was not attempted → 0% of marks.
- D. The function works well but you do not pass the demo due to you do not answer my questions about your code or your answers are wrong → 5% of marks.

**IMPORTANT:** I will use a source code plagiarism detection tool. In case of detecting a copy (for at least one method) between some students, all these students get 0% marks for the whole assignment. Including the student that originally coded it.

## SUBMISSION DETAILS.

### Deadline.

2<sup>nd</sup> December, 11:59pm.

### Submission Details.

Please submit to Blackboard (Assignments) **ONLY** the following files (do not ZIP the files) :

- Part 1: File [DivideAndConquerAlgorithms.java](#)
- Part 2: File [DivideAndConquerAlgorithms.java](#)
- Part 3: File [ChangeMaking.java](#)

### Lab Demo.

A brief individual interview about the assignment will take place on our lab session on the lab of week 12. **The demo is mandatory for the assignment to be evaluated.**

Please, let me know in the lab if you are willing to do the demo earlier. (With the corresponding uploading of the full assignment to the blackboard). Once the demo is done, there will be no possibility of re-evaluating later. The evaluation will be done with the exact code presented at this moment. Therefore, my advice is to take this option only if you are 100% sure that your assignment is already completed.

If you cannot attend (for important reasons) to the demo, you must talk with me and we will do another appointment to the demo no later than scheduled demo.

