Ben-Gurion University of the Negev

Data Structures

202.1.1031

# Assignment No. 2

*Responsible staff members:*
Dr. Ben daniel Sebastian(sebastia@post.bgu.ac.il)
Naser Rabah (rabahn@post.bgu.ac.il)

Publish date: 26.11.2025
Submission date: 12.12.2026

# Contents

# 0 Integrity Statement

To sign and submit the integrity statement (presented below), fill in your name and ID in the method signature in the class IntegrityStatement. See the comment in the method signature for example.

If you have relied on or used an external source, **you must cite** that source at the end of your integrity statement. External sources include shared file drivers, Large Language Models (LLMs) including ChatGPT, forums, websites, books, etc.

> "I certify that the work I have submitted is entirely my own. I have not received any part of it from any other person, nor have I given any part of it to others for their use. I have not copied any part of my answers from any other source, and what I submit is my own creation. I understand that formal proceedings will be taken against me before the BGU Disciplinary Committee if there is any suspicion that my work contains code/answers that is not my own."

**Assignments without a signed integrity statement will get 0 grade!**

# 1 Preface

## 1.1 Assignment Structure

In this assignment, we discuss the Abstrct Data Type of Dynamic Set, and three main data structures seen in class: Array (sorted and unsorted), Linked List (sorted and unsorted), and AVL Tree. The assignment is to be implemented using Java using the included skeleton files. The assignment consists of three main parts:

- Warm-Up and Familiarization.
- Implementing Dynamic Set.
- Designing a Data Structure according to specifications.

You **must** read the entire assignment **at least once** before starting work on it!

## 1.2 General Instructions

- The skeleton files for the assignment are available in the VPL system of the assignment. You should download the skeleton files to your computer, implement your code, and upload your submission to the VPL system.
- You are expected to test your code locally and on the VPL system and sign your name and ID in the IntegretyStatement file as mentioned above.
- In this assignment, you **must not add any additional `.java` files** to the assignment. Your code should be submitted in the already existing files.
- You **must not edit the given implementation** in the skeleton files. You are **permitted** (and actually should) to add fields and/or auxiliary methods to some of the classes as you wish for your implementations.
- You **may not** use any built-in Java data structure or collection. You **can** use:
  - Primitive variables.
  - Basic arrays.
  - The classes given in the skeleton files.
- You **can assume that the input is legal** for all the functions in the assignment. There is **no need to check and throw exceptions** for illegal arguments.
- Your code will be tested automatically for correctness, and manually for **efficiency and clarity**.
- **It is strongly recommended that you submit an updated version of your assignment after each section you solve. Avoid submissions in the last hour and preferably avoid submissions in the last evening.**

# 2 (15 points) Warm-Up and Familiarization

## 2.1 Given Implementations

In the skeleton files you are given five classes, of the following data structures:

- **MyArray** (unsorted)
- **MySortedArray**
- **MyLinkedList** (unsorted, doubly linked list)
- **MySortedLinkedList** (doubly linked list)
- **MyAVLTree**

In addition, you are given a class **Element** representing a general element in a general data structure with key and satellite data, and three extend classes representing specific elements in specific data structures:

- **Link** (a link in a Linked-List).
- **TreeNode** (a node in an AVL-Tree).
- **ArrayElement**[1] (an element in an Array).

**Read those classes carefully and make sure you understand the given implementations!**

## 2.2 Implementation of specific methods

**Task 2.1:** In the class **MyArray**, implement the method **reverse()**. The method should reverse the order of the elements in the array.

**Task 2.2:** In the class **MyLinkedList**, implement the method **reverse()**. The method should reverse the order of the elements in the linked-list.

**Task 2.3:** In the class **MyAVLTree**, implement the method **depthOfMax()**. The method should return the depth of the element with the largest key in the tree (return $(-1)$ if the tree is empty).

[1] As you will see, in the class **ArrayElement** there is a field *index* that should be maintained. One might ask, why to complicate such a simple and direct implementation such an array with extra fields. The reason is to be consistent with the operations of Dynamic-Set. Some of them receives pointer to an element as input, and should have the information about the element's position in the data-structure.

# 3 (25 points) Implementing Dynamic Set

As learned in class, Dynamic Set is an abstract data type supporting the following operations: **search**, **insert**, **delete**, **successor**, **predecessor**, **minimum**, **maximum**.
In this section, you are required to implement a Dynamic Set with the given time complexities for each operation:

- **search(k)** - Returns the element in the DS whose key is $k$ if exists (return *null* if no element in the DS has key $k$) - $O(\log n)$.
- **insert(x)** - Add the element $x$ to the DS (Assumes that insert is always called when there is a place in the array to insert) - $O(n)$.
- **delete(x)** - Removes the element $x$ from the DS (the operation receives a pointer to $x$) - $O(n)$.
- **minimum()** - Returns the element in the DS with smallest key (return *null* if no such exists) - $O(1)$.
- **maximum()** - Returns the element in the DS with largest key (return *null* if no such exists) - $O(1)$.
- **successor(x)** - Returns the element in the DS with smallest key that is larger than $x.key$ (return *null* if no such exists) - $O(1)$.
- **predecessor(x)** - Returns the element in the DS with largest key that is smaller than $x.key$ (return *null* if no such exists) - $O(1)$.

**Task 3.1:** In the class **MyDynamicSet**, implement a dynamic set that supports the requirements mentioned above.

## 4 (60 points) Designing a Data Structure according to specifications

In this section, we will design a new data structure according to a specified ADT and its time complexity requirements. We need to Implement, for a client who owns a products factory, a data structure to store his products. Each product is characterized by a unique identification number and quality, and is represented by the class **Product**. The optional qualities for a product are: 0/1/2/3/4/5. The data structure should support the following operations in the given time complexities:

| Operation | Description | Time complexity |
|---|---|---|
| $Init()$[2] | Initiate an empty data structure. | $O(1)$ |
| $Insert(id, quality)$ | Insert a product with a unique identification number $id$ (natural number), and its quality $quality$ (integer between 0 to 5), to the data structure. | $O(\log n)$ |
| $Delete(id)$ | Delete the product with the unique identification number $id$ from the data structure (if there exists such an element in the data structure). | $O(\log n)$ |
| $MedianQuality()$ | Return the median quality of the products currently stored in the data structure. The median is defined as the $\lceil \frac{n}{2} \rceil$-th smallest element. (Return $-1$ if the data structure is empty) | $O(1)$ |
| $AvgQuality()$ | Return the average quality of the products currently stored in the data structure. (Return $-1$ if the data structure is empty) | $O(1)$ |
| $JunkWorst()$ | Delete all the elements with quality 0 from the data structure. This operation should return a linked list with all the deleted elements. | $O(1)$ |

**Task 4.1:** In the class **MyDataStructure**, implement a data structure that supports the requirements mentioned above.

# Good Luck!

[2]The initiation in the code is by implementing the constructor $MyDataStructure()$.