

TRINITY COLLEGE
DEPARTMENT OF COMPUTER SCIENCE
CPSC 215: Data Structures and Algorithms
Instructor: Dr. Chandranil Chakrabortii
Spring 2025

Assignment 2

Academic Honesty Policy

In working on programming assignments, you may discuss broad issues of interpretation and understanding and general approaches to a solution. However, conversion to a specific solution or to program code must be your own work. Programming assignments are expected to be the work of the individual student, designed, and coded by him or her alone. Violations are easy to identify and will be dealt with promptly according to the Academic Integrity and Intellectual Dishonesty outlined in the Student Handbook.

- Copying another person's programs or encouraging or assisting another person to commit plagiarism is cheating. In particular, the following activities are strictly prohibited:
- Giving and receiving help in the actual development of code or writing of an assignment.
- Looking at another person's code or showing your code to another person.
- Sharing a copy of all or part of your code regardless of whether that copy is on paper or in a computer file.
- Turning in the work of any other person(s) (former students, friends, textbook authors, people on the Internet, etc.) and representing it as your own work.
- Fabricating compilation or execution results.
- Use of AI ChatBots for any help regarding the assignment is **strictly prohibited**.

The penalty for cheating is a failing grade (F) for the course, and the student is asked to appear at an Academic Dishonesty Hearing. In addition, the College also places a record of the incident in the student's permanent record. It is your responsibility to protect your work from unauthorized access. Do not discard copies of your programs in public places. Do not leave computers unattended and copies of output lying around.

Objectives

- Understanding Asymptotic Notations
 - Explain the concepts of O , Ω , and Θ notations.
 - To get familiar with an array-based List ADT implementation
 - To design and implement a Set ADT
-

Part I: Analyze code to indicate running time.

Given the code below using Big O notation what is the running time $T(n)$?

A.

```
int array_sum(int[] a, int n) {
    int i;
    int sum = 0;
    for (i = 0; i < n; i++) {
        sum = sum + a[i];
    }
    return sum;
}
```

B.

```
int sum = 0;
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        for (k = 0; k < n; k++) {
            if (i == j && j == k) {
                for (l = 0; l < n * n * n; l++) {
                    sum = i + j + k + l;
                }
            }
        }
    }
}
```

C.

```
int sum(int a, int b) {
    int c = a + b;
    return c;
}
```

D.

```
void fun(int n) {
    int i, j;
    for (i = 1; i <= n; i++)
        for (j = 1; j < log(i); j++)
            System.out.println("CPSC 215");
}
```

Assume that $\log(x)$ returns log value in base 2.

E.

```
int binarySearch(int[] arr, int target) {
    int left = 0;
    int right = arr.length - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            return mid; // Found the target at index mid
        } else if (arr[mid] < target) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return -1; // Target not found
}
```

Part II: Proofs

Given the following equations. For Q1, write down the time complexity and explain reasoning. For Q2-Q5, find n and c (as shown in lecture slides):

1. **Equation:** $10n^3 + 24n^2 + 3n \log n + 144$
 - **Complexity:**
 - $O(n)$
 - $\Omega(n)$
 - $\Theta(n)$
2. **Prove:** $n^3 + 20n + 1 = O\left(\frac{n^3}{6}\right)$
3. **Prove:** $7n^2 + 5 = O(n^3)$
4. **Prove:** $n^3 + 20n = \Omega(n^2)$
5. **Prove:** $3n \log n + 4n + 5n \log n = \Theta(n \log n)$

Note: The "O," " Ω ," and " Θ " notations are used to describe the upper, lower, and tight bounds, respectively, on the growth rate of a function. The provided equations and proofs involve analyzing the asymptotic behavior of the functions in terms of these notations.

Part III: Set ADT: Array based Implementation.

Getting Started

- Open Eclipse and create a Java Project (File -> New -> Java Project)
- Give a project name (For example: assignment_2_CPSC_215)
- Go to Project -> Right Click -> New Package
- Give a package name (For example: assignment_2_CPSC215)
- Now to create the classes, go to package -> Right Click -> New Java class.
- Paste contents of the given files. Keep the first line in place (package "package_name";)
- Understand the code structure before starting to write code.

You will need the following classes.

- Set.java (No need to change)
- ASet.java (**Need to change**)
- SetTest.java (No need to change) [contains the main method]

Set ADT Interface

A set is an unordered collection of elements. An empty set is a set that contains no elements. The union of two sets, A and B, denoted $A \cup B$, is the collection of elements contained in either A or B. For example, if $A = \{1, 2, 3, 4, 5\}$ and $B = \{4, 5, 6, 7\}$, then $A \cup B = \{1, 2, 3, 4, 5, 6, 7\}$. Note that there is only one of each element in the union even though the elements 4 and 5 occur in both A and B.

The intersection of two sets, A and B, denoted $A \cap B$, is the collection of elements that are contained in both A and B, that is, $A \cap B = \{4, 5\}$. The difference of two sets, A and B, denoted $A - B$, is the collection of elements that are contained in A, but not in B, that is, $A - B = \{1, 2, 3\}$. The symmetric difference of two sets, A and B, denoted $A \Delta B$, is the set of elements that are in one but not both sets, that is, $A \Delta B = \{1, 2, 3, 6, 7\}$.

The interface for Set ADT (Set.java) supports the following operations:

- contains(x) – returns true if the item x belongs to this set and false otherwise:
- isEmpty() – returns true if this set is empty false otherwise.
- size() – returns the number of elements in this set.
- insert(x) – inserts an item x into this set.
- getValue(i) – returns the item at position i in this set.
- union(S) – returns the union of this set with the set S.
- intersect(S) – returns the intersection of this set with the set S.
- diff(S) – returns the difference of this set with the set S.
- symDiff(S) – returns the symmetric difference of this set with the set S

Set ADT Implementation

Tasks:

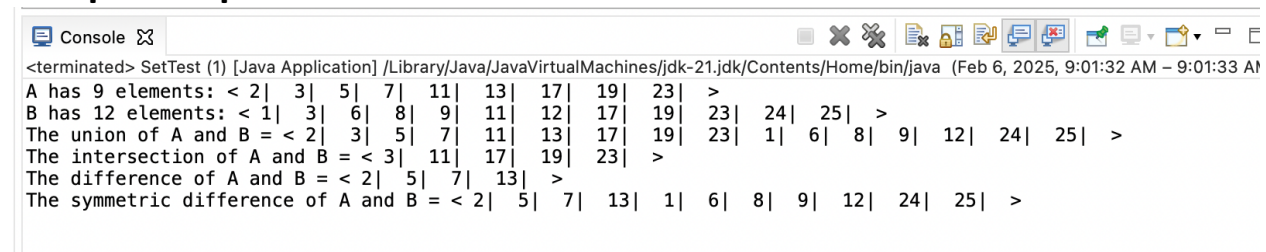
Using an array, implement the methods from Set.java in ASet.java. In addition to the methods in the interface, you will need to implement the following methods in the ASet class:

- ASet () – the constructor.
- toString() – returns a string that contains all elements in this set. (Check sample output)

Test your Set ADT

- Review SetTest.java (already completed)
- Use SetTest.java to test your ASet class.

Sample Output:



```
<terminated> SetTest (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Feb 6, 2025, 9:01:32 AM – 9:01:33 AM)
A has 9 elements: < 2| 3| 5| 7| 11| 13| 17| 19| 23| >
B has 12 elements: < 1| 3| 6| 8| 9| 11| 12| 17| 19| 23| 24| 25| >
The union of A and B = < 2| 3| 5| 7| 11| 13| 17| 19| 23| 1| 6| 8| 9| 12| 24| 25| >
The intersection of A and B = < 3| 11| 17| 19| 23| >
The difference of A and B = < 2| 5| 7| 13| >
The symmetric difference of A and B = < 2| 5| 7| 13| 1| 6| 8| 9| 12| 24| 25| >
```

Part IV: Linked Lists

- Examine the provided starter code (P3), which consists of three files:
 - Node.java (No need to change)
 - LinkedList.java (**Need to change**)
 - LinkedListTest.java (No need to change)) [contains the main method]
- **To do: Implement the missing parts of the LinkedList class:**
 - Understand the purpose and structure of each class.
 - Note the generic nature of the linked list to work with different data types.
 - Add Java Doc in each class.

Methods to Implement in LinkedList Class

- **LinkedList() Constructor:**
 - Initializes an empty linked list with both head and tail references set to null.
- **insertAtBeginning(T data) Method:**

- Inserts a new node with the specified data at the beginning of the linked list.
 - Creates a new node with the provided data.
 - Points the new node's next reference to the current head of the list.
 - Updates the head reference to point to the new node.
 - If the list is empty, sets the tail reference to the new node.
- **insertAtEnd (T data) Method:**
 - Inserts a new node with the specified data at the end of the linked list.
 - Creates a new node with the provided data.
 - If the list is empty, sets both head and tail references to the new node.
 - Otherwise, sets the next reference of the current tail node to the new node and updates the tail reference to point to the new node.
- **display() Method:**
 - Displays the elements of the linked list by traversing through the list starting from the head node.
 - Iterates through each node in the list and prints the data of each node.
 - Continues until it reaches the end of the list (node with null next reference).
 - After printing all elements, moves to the next line for better formatting.
- **removeDuplicates() Method:**
 - Removes duplicate elements from the linked list while preserving the order of elements.
 - Utilizes nested loops to compare each element with all subsequent elements in the list.
 - If a duplicate is found, removes the duplicate element by updating the next reference of the previous node to skip the duplicate node.
 - Continues until the end of the list is reached, ensuring that only the first occurrence of each element is retained in the list.
- **insertSorted(T data) Method:**
 - Inserts a new element into the linked list while maintaining the sorted order of elements.
 - Creates a new node with the specified data and identifies the appropriate position to insert the new node in the sorted list.
 - Utilizes comparisons using the compareTo method to determine the correct insertion position relative to existing elements.
 - Inserts the new node into the sorted list by updating the appropriate next references to maintain the sorted order.
 - Ensures that the list remains sorted after each insertion, and handles cases where the list is empty or the new node should be inserted at the beginning or end of the list.
- Test your implementation by running the **LinkedListTest.java file (Already completed)**.

Sample Output:

```
(base) nilchakraborttii@Administrator's-MacBook-Air P3 % javac *.java
(base) nilchakraborttii@Administrator's-MacBook-Air P3 % java LinkedListTest
Creating a new List - 1
Inserting 5 at beginning
Inserting 30 at end
Inserting 15 at end
Printing the Linked List - 1:
5 30 15
[
Creating another String List - 2
Inserting Hello at beginning
Inserting World at end
Inserting Morning at beginning

Printing the String Linked List - 2:
Morning Hello World

Testing remove duplicates
Creating another Integer List - 3
Inserting 5 at beginning
Inserting 10 at end
Inserting 50 at end
Inserting 15 at end
Inserting 10 at end
Inserting 5 at end
Original Integer Linked List:
5 10 50 15 10 5
After removing duplicates:
5 10 50 15

Testing Insert Sorted
Creating another Integer List - 4
Inserting 10
Inserting 5
Inserting 20
Inserting 15
Integer Linked List with elements added using insertSorted:
5 10 15 20
```

Handin

After completing your assignment, keep each part of the assignment into different folders (PI, PII, PIII and PIV) and upload your submission to the Moodle course website as a single ZIP file. You may complete Parts I and II on a digital document (e.g., Word or PDF) or handwritten on paper. If you complete it on paper, you can either submit the original or take a clear picture of your work and include it in the document before submission.
