CSDS 233 : Introduction to Data Structures

Assignment 3

Rohan Singh and Hieu Dang

1 Instructions

This assignment covers the basics of recursion and lists. This assignment is divided into a Written Assignment and a Programming assignment worth 40 and 60 points respectively.

You can contact us at:

- Rohan Singh: rxs1182@case.edu, office hours: Th 5:30-6:30 pm, extra office hours: T 4:00-5:00pm (PBL student lounge).
- Hieu Dang: htd9@case.edu, office hours: Fr 5:00-6:00 pm, extra office hours: Fr 4:00-5:00pm (KSL lower level).

1.1 Instruction for Written Assignment

Here are the instructions for the written assignment:

- Do **NOT** put your solutions inside the zip file for your programming assignment.
- Write your solutions neatly.
- Submit your written assignment in a pdf format on canvas with the following naming convention **CSDS233-W3-YourCaseID.pdf**.

1.2 Instructions for Programming Assignment

Here are the instructions for the programming assignment:

- Please use the class and method names that are provided. Additionally, please use the same argument and return types as asked in the question for a given method. You will lose points for not following this instruction.
- You may use helper methods or Nested classes, but they must follow proper rules of encapsulation.
- You may **NOT** import anything from the Java API such as LinkedList, ArrayList, Iterator, etc. You will be given no credit for the questions if you use them.
- In addition to correctness of output, your solutions will be graded on the basis of runtime and complexity and you will lose points if your code has greater runtime or space complexity than the optimal solution.

- You need to provide a seperate testing file, where you will be writing your JUnit test cases for each of the methods. Make sure to include simple, complicated and edge cases in your testing files to get full credit.
- Submit your programming assignment in a zip file with the following naming convention **CSDS233-P3-YourCaseID.zip**. Only include your .java source files, do **NOT** include any .class files, do not use any other intermediate directories within your compressed folder.

2 Written Assignment [40pts]

Problem 1
Binary Search Analysis [10pts]

Here are two implementations of the binary search algorithm: one using recursion and the other using an iterative approach.

```
public static int recursiveBinarySearch(int[] array, int target,
int left, int right) {
    if (left <= right) {
        int mid = left + (right - left) / 2;
        if (array[mid] == target)
            return mid;
        if (array[mid] < target)
            return recursiveBinarySearch(array, target, mid + 1, right);
        return recursiveBinarySearch(array, target, left, mid - 1);
    }
    return -1;
}

public static int iterativeBinarySearch(int[] array, int target) {
    int left = 0;
    int right = array.length - 1;

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

        if (array[mid] == target)
            return mid;

        if (array[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}
```

Compare the two algorithms in terms of space and time complexity. Explain your answer.

Problem 2
Iterator [10pts]

1. Is it possible to create an iterator on a custom object that does not implement the Iterable interface? Briefly explain why (3 points).
2. Compare and contrast the Iterator and ListIterator interfaces in Java (7 points).

Problem 3
Sorting [10pts]

1. Given an array of 6 elements: 2, 24, 69, 8, 1, 15. Please explain step-by-step how selection sort and insertion sort will be performed on the array. You can choose either the recursive or the iterative way (7 points).
2. Why is the best case for Insertion Sort $O(N)$? Give an example of the scenario in which the best case occurs (3 points).

Problem 4
Linkedlist [10pts]

Write pseudo-code/code in Java for the following problem:

Reverse a section of a singly linked list: You have a singly linked list and two integers, left and right, with left ≤ right. Your task is to reverse the nodes of the list from position left to position right and then return the modified list.

For example:
List: 12 → 15 → 24 → 10 → 19
Left: 3
Right: 5
The resulting list should be: 12 → 15 → 19 → 10 → 24

3 Programming Assignment [60pts]

Problem 1
public class Recursion [20pts]

Write a class **Recursion** and implement the following methods using recursion:

- **public int sumDigits(int n)**: This recursive method should efficiently compute the sum of the digits of a given non-negative integer n.
- **public int gcd(int a, int b)**: This recursive method should efficiently compute the greatest common divisor (gcd) of two positive integers a and b. You can assume both a and b are positive integers. Hint:

$$\gcd(a, b) = \gcd(b, r)$$

- **public boolean isPalindrome(String str)**: This recursive method will return a boolean value depending on whether or not the given string *str* is a palindrome or not. A palindrome is a word that reads the same backward as forward, e.g., "aba", "cdddc".
- **public Node swapNodesInPairs(Node head)**: This recursive method will return the head of a new LinkedList that is the original linked list with nodes swapped for every two nodes, starting from the head. Modifying the values in the list's nodes are not allowed. For example, list: $1 \to 2 \to 3 \to 4 \to 5 \to$ null will become $2 \to 1 \to 4 \to 3 \to 5 \to$ null
- **public int binomial(int n, int k)**: This use recursion to compute the binomial choose defined as:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Hint:

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

You must add an initial check to avoid negative numbers for both n and k and throw an **IllegalArgumentException**.

Problem 2
public class ArrayList [20pts]

Write a class **ArrayList** with integer elements and implement the following methods:

- **public void add(int n)**: This method will add the integer n to the end of the list.
- **public void add(int n, int index)**: This method will add the integer n at index *index*. You must add an initial check to avoid negative values for *index* and throw an **IllegalArgumentException**. In the case where *index* is greater than the length of the list, you must simply add it to the end of the list.
- **public int indexof(int n)**: This method will return the index of the element n if it is in the list. If it is not it in the list, you must return -1, if there are mulitple instances of the integer n in the list, you must return the index of the first instance of n.

- **public int remove(int index)**: This method will remove the at index *index* and return the value of the removed element. You must add an initial check to avoid negative values for *index* and throw an **IllegalArgumentException**. In the case where *index* is greater than the length of the list, you must simply remove the last element of the list.
- **public void removeValue(int n)**: This method will remove the first instance of the element n from the list.
- **public void removeall(int n)**: This method will remove **all** instances of the element n from the list.
- **public double mean()**: This method will return the mean of all of the elements of the list.
- **public double variance()**: This method will return the variance of all of the elements of the list. Variance is given by the formula:

  $$\sigma^2 = \frac{\sum_{\forall i}(x_i - \bar{x})^2}{n-1}$$

  where $\bar{x}$ is the mean of the list.
- **public ArrayList sublist(int lower, int upper)**: This method will return a new ArrayList with all of the elements of the original list that are within the lower and upper bounds.
- **public ArrayList removeNoise()**: This method will return a new ArrayList with all of the elements of the original list that are within 3 standard deviations of the mean.

Problem 3
public class LinkedList [20pts]

Write a class **LinkedList** with integer elements and implement the following methods:

- **public void add(int n)**: This method will add the integer n to the end of the list.
- **public void add(int n, int index)**: This method will add the integer n at index *index*. You must add an initial check to avoid negative values for *index* and throw an **IllegalArgumentException**. In the case where *index* is greater than the length of the list, you must simply add it to the end of the list.
- **public int indexof(int n)**: This method will return the index of the element n if it is in the list. If it is not it in the list, you must return -1, if there are mulitple instances of the integer n in the list, you must return the index of the first instance of n.
- **public int remove(int index)**: This method will remove the at index *index* and return the value of the removed element. You must add an initial check to avoid negative values for *index* and throw an **IllegalArgumentException**. In the case where *index* is greater than the length of the list, you must simply remove the last element of the list.
- **public void removeValue(int n)**: This method will remove the first instance of the element n from the list.

- **public void removeall(int n)**: This method will remove **all** instances of the element n from the list.
- **public double mean()**: This method will return the mean of all of the elements of the list.
- **public double variance()**: This method will return the variance of all of the elements of the list. Variance is given by the formula:

  $$\sigma^2 = \frac{\sum_{\forall i}(x_i - \bar{x})^2}{n-1}$$

  where $\bar{x}$ is the mean of the list.
- **public LinkedList sublist(int lower, int upper)**: This method will return a new LinkedList with all of the elements of the original list that are within the lower and upper bounds.
- **public LinkedList removeNoise()**: This method will return a new LinkedList with all of the elements of the original list that are within 3 standard deviations of the mean.