# Comp151 Lab05

In Lab05 you will be working on **six** separate applications:

1. Working with Bags
2. BaseToDecimal
3. MinAndSumInArray
4. StringsOfNumbers
5. MazeSolver
6. MazeSolverWithStack

Each application has a main inside the corresponding .java file and there are no dependencies between them. See the descriptions below. The skeletons for each program are provided.

### UML Diagram:

**A screenshot of a social media post with text and people in background

Description automatically generated**

1. **Working with Bags**

Implement recursively two methods getFrequencyOf and isBagSorted in two classes: ResizableArrayBag and LinkedBag. Iterative implementations of getFrequencyOf methods are left commented out in the classes for your reference.

isBagSorted returns true if all elements in the bag are sorted in ascending order.

1. **BaseToDecimal**

If n is a positive integer in Java, n % 10 is its rightmost digit and n / 10 is the integer obtained by dropping the rightmost digit from n. Using these facts, write a recursive method called displayDigits that displays the digits of an integer n in decimal. For example, the integer number 345 should be displayed as a String "3 4 5"; where the negative integer number -345 should be displayed as a String "-3 4 5". Note the spaces between the digits.

Now observe that you can display n in any base between 2 and 9 by replacing 10 with the new base. Write displayDigits2to9Base method that is a revised version of yours displayDigits method, to accommodate a given base. Please note that the modified method will essentially convert a decimal number into its equivalent in the given base. For example, 10 in base 8 will be displayed as a String "1 2"; 5 in base 2 will be displayed as a String "1 0 1", and so on.

Test your methods with the driver provided in main.

## MinAndSumInArray

1. Implement a recursive method called minimum that returns the smallest integer in an array of integers.

If you divide the array into two pieces - halves, for example - and find the smallest integer in each of the pieces, the smallest integer in the entire array will be the smaller of these two integers. Since you will be searching a portion of the array - for example, the elements array[first] through array[last] - it will be convenient for your method to have three parameters: the array and two indices: first and last.

NOTE: You can refer to the method displayArray in Segment 7.18 in the textbook for the inspiration.

Next write **four different** **recursive** methods that each compute the sum of the integers in an array of integers where the elements in **all solutions** are **added in the order of their position in the array**. Model your methods after the displayArray methods given in Segments 7.15 through 7.18:

1. sumArrayStartWithFirst – add all the elements considering the first element and recursively considering the rest
2. sumArrayStartWithLast – add all the elements considering the last element and recursively consider the rest
3. sumArraySplitInTwo – *“divide and conquer”* solution – divide the array in two pieces and process each of the pieces separately (see the minimum description above). In this solution computed middle element should be included in the left “half” of the array
4. sumArraySplitInTwoAddMid – *“divide and conquer”* solution – divide the array in two pieces and process each of the pieces separately. In this solution however, the middle element should be excluded from both “halves”, instead you recursively add elements in the left half, add the middle element and add recursively elements from the right half.

Test your methods with the driver provided in main.

1. **StringsOfNumbers**

Write two recursive methods that generate string of numbers. The first method generates all the possible strings that contain the combinations of *n* bits, where *n* is given by the user. The second method is generalization of the first method. It also generates all the permutations of *n* numbers, but the numbers are drown from [0..k); where the *k* is given by the user. See sample runs below. Implement the algorithms that you designed as part of your pre-lab.

**SAMPLE RUN:**

**Please enter an integer value of n representing the number of digits in a string**

**3**

**Generating binary-Strings:**

**000**

**001**

**010**

**011**

**100**

**101**

**110**

**111**

**Please enter an integer value k; strings of length n will be drown from 0..k-1**

**4**

**Generating k-Strings:**

**000**

**001**

**002**

**003**

**010**

**011**

**012**

**013**

**020**

**021**

**022**

**023**

**030**

**031**

**032**

**033**

**100**

**101**

**102**

**103**

**110**

**111**

**112**

**113**

**120**

**121**

**122**

**123**

**130**

**131**

**132**

**133**

**200**

**201**

**202**

**203**

**210**

**211**

**212**

**213**

**220**

**221**

**222**

**223**

**230**

**231**

**232**

**233**

**300**

**301**

**302**

**303**

**310**

**311**

**312**

**313**

**320**

**321**

**322**

**323**

**330**

**331**

**332**

**333**

**Process finished with exit code 0**

**SAMPLE RUN WITH DEBUGGING STATEMENTS:**

**Please enter an integer value of n representing the number of digits in a string**

**3**

**Generating binary-Strings:**

**--> FOR DEBUGGING: At the start of the method: str is "" and n is 3**

**----> FOR DEBUGGING: About to call itself to append '0': str is "" and n is 3**

**--> FOR DEBUGGING: At the start of the method: str is "0" and n is 2**

**----> FOR DEBUGGING: About to call itself to append '0': str is "0" and n is 2**

**--> FOR DEBUGGING: At the start of the method: str is "00" and n is 1**

**----> FOR DEBUGGING: About to call itself to append '0': str is "00" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "000" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**000**

**----> FOR DEBUGGING: About to call itself to append '1': str is "00" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "001" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**001**

**----> FOR DEBUGGING: About to call itself to append '1': str is "0" and n is 2**

**--> FOR DEBUGGING: At the start of the method: str is "01" and n is 1**

**----> FOR DEBUGGING: About to call itself to append '0': str is "01" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "010" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**010**

**----> FOR DEBUGGING: About to call itself to append '1': str is "01" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "011" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**011**

**----> FOR DEBUGGING: About to call itself to append '1': str is "" and n is 3**

**--> FOR DEBUGGING: At the start of the method: str is "1" and n is 2**

**----> FOR DEBUGGING: About to call itself to append '0': str is "1" and n is 2**

**--> FOR DEBUGGING: At the start of the method: str is "10" and n is 1**

**----> FOR DEBUGGING: About to call itself to append '0': str is "10" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "100" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**100**

**----> FOR DEBUGGING: About to call itself to append '1': str is "10" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "101" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**101**

**----> FOR DEBUGGING: About to call itself to append '1': str is "1" and n is 2**

**--> FOR DEBUGGING: At the start of the method: str is "11" and n is 1**

**----> FOR DEBUGGING: About to call itself to append '0': str is "11" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "110" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**110**

**----> FOR DEBUGGING: About to call itself to append '1': str is "11" and n is 1**

**--> FOR DEBUGGING: At the start of the method: str is "111" and n is 0**

**---> FOR DEBUGGING: Executing base case**

**111**

**Please enter an integer value k; strings of length n will be drown from 0..k-1**

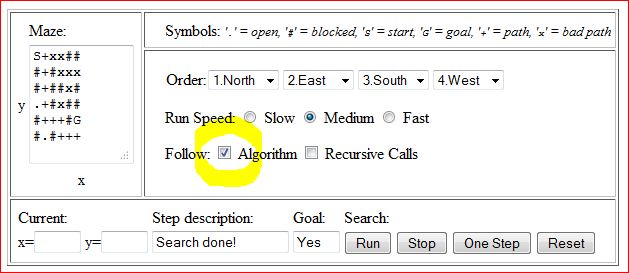
**4**

**Generating k-Strings:**

**[ … ]**

## Maze Solver

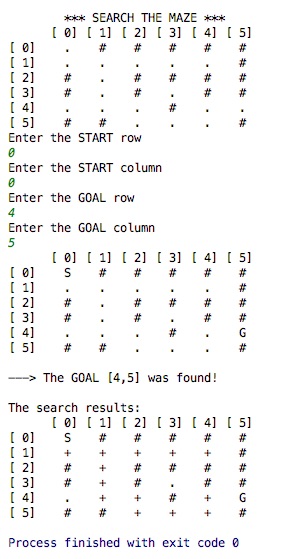
* 1. Go to <https://www.cs.bu.edu/teaching/alg/maze/> and analyze the given algorithm
  2. Run the applet at the bottom of the page with the “Algorithm” box checked:

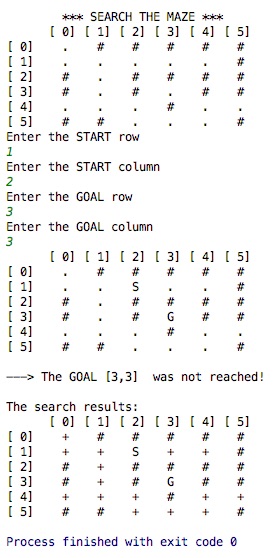


* 1. Implement the following maze search algorithm using recursion, use MazeSolver.java as the starting point (please notice that our algorithm does not unmark [x,y] if it is not in the solution path):

**FIND-PATH(x, y)**

* if ([x,y] outside maze) return false
* if ([x,y] is goal) return true
* if ([x,y] not open) return false
* mark [x,y] as part of solution path
* if (FIND-PATH(North of x,y) == true) return true
* if (FIND-PATH(East of x,y) == true) return true
* if (FIND-PATH(South of x,y) == true) return true
* if (FIND-PATH(West of x,y) == true) return true
* return false
  1. See two sample runs below:





## MazeSolverWithStack

Use stack instead of recursion to implement the previous algorithm, use MazeSolverWithStack.java as the starting point. Your program should produce similar results to the MazeSolver.java. See segment “Using a Stack Instead of Recursion” on page 224 of your textbook.

Sample runs:

