CSE 8B: Introduction to Programming and Computational Problem Solving - 2

Assignment 9

Recursion

Due: Wednesday, November 30, 11:59 PM

Learning Goals:

- Understanding recursion in practice

NOTE: This assignment must be completed INDIVIDUALLY. Pair programming is NOT allowed for this assignment.

Your grade will be determined by your most recent submission. If you submit to Gradescope after the deadline, it will be marked late and the late penalty will apply regardless of whether you had past submissions before the deadline.

If your code does not compile on Gradescope, you will receive an automatic zero on the assignment.

Coding Style (10 points)

For this programming assignment, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have COMPLETE file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Getting started (0 points)

1. Make sure there is no problem with your Java software development environment. If there is any, then review Assignment 1, or come to the office/lab hours before you start Assignment 9.
2. First, navigate to the `cse8b` folder that you have created in Assignment 1 and create a new folder titled `assignment9`

3. Download the starter code. You can download the starter code from Piazza → Resources → Homework → `Assignment9.java`.
4. Place `Assignment9.java` within the `assignment9` folder that you have just created.
5. Compile the starter code within the `assignment9` using the command `javac Assignment9.java`. You should get output that looks like this:

```
$ javac Assignment9.java
$ java Assignment9
FAILURE: Unexpected output for binarySearch()
ERROR: Failed test.
```

Part 1: Implement `binarySearch` method (30 points)

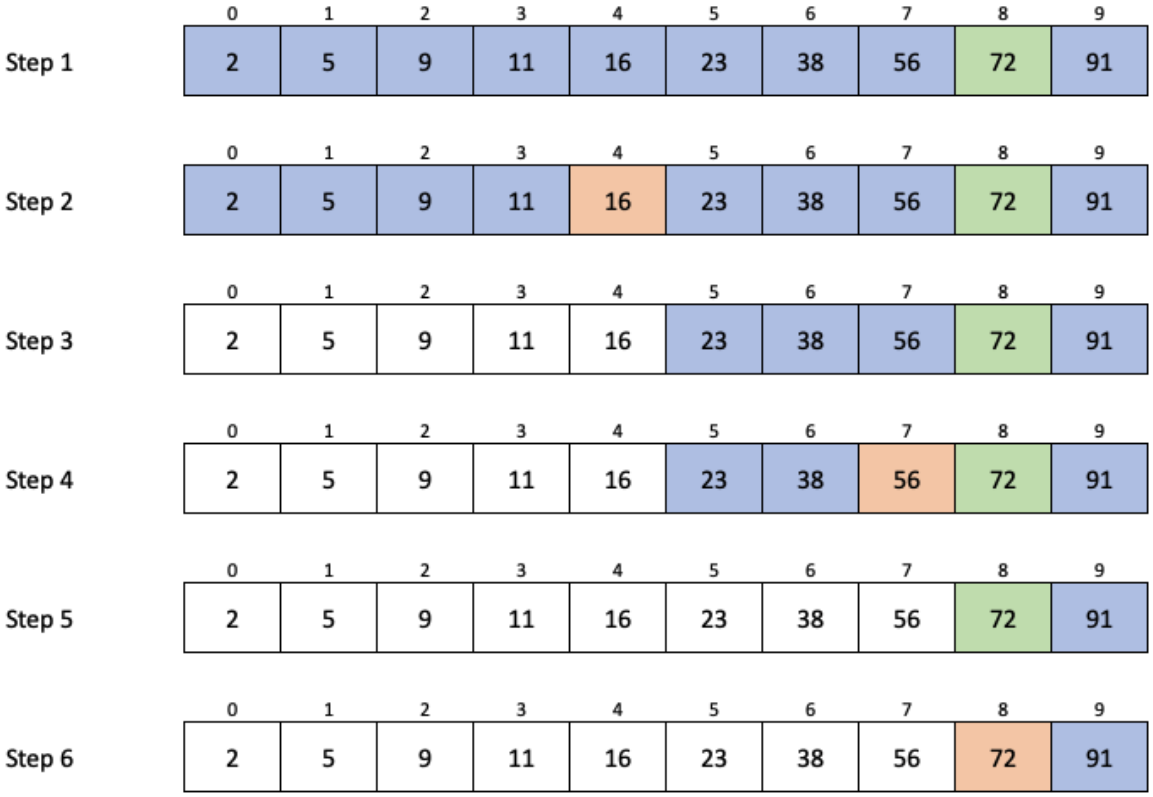
In `Assignment9.java`, you are provided with a method `binarySearch(int array[], int lower, int upper, int token)`. This method takes an integer `array`, the `lower` and `upper` index to search within the array, and the `token` to search for in the array. It returns the *index* of the `token` in the `array` if it is present, else it returns `-1` when the `token` is not present. You will need to implement this using *recursion*.

Binary search is a technique used to find elements in a sorted array. Since the elements are arranged in ascending order, we can find an element using this technique instead of looking at all the elements of the array as explained below.

Given an array of elements of `size` indexed from `0` to `size - 1`,

1. Check the value of the middle element. If it is equal to the search token, return the index of the middle element. Here, the search interval is `0` to `size - 1`.
2. If the value of the middle element is greater than the search token, it implies that the token is present in the 1st half of the array. Repeat step 1 for the 1st half of the array. Here, the search interval from index `0` to the index of the middle element `- 1`.
3. If the value of the middle element is less than the search token, it implies that the token is present in the 2nd half of the array. Repeat step 1 for the 2nd half of the array. Here, the search interval is from the index of the middle element `+ 1` to the index `size - 1`..
4. Repeat this process until the search token is found or the search interval is empty.

Consider the example given below. Suppose we have an array of 10 elements sorted in ascending order. Since an array is indexed from `0`, the valid indices range from `0` to `9`. Let's assume we want to search for the token `72` (found at index `8`).

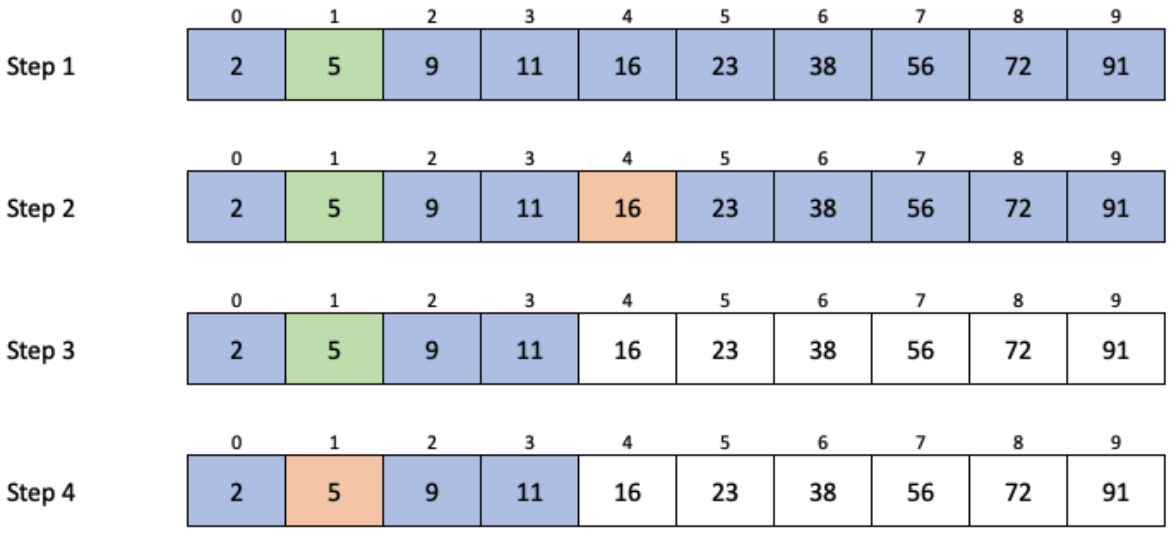


Here's how binary search would work:

- In step 1, we cover all the elements in the full array (i.e., from index `0` to `9`). Here, the middle element is the element at index `4` since `(0+9)/2 = 4`.
- In step 2, we check if the value at index `4` (i.e., `16`) is equal to `72`.
- In step 3, since `72 > 16` we only cover the subarray consisting of the 2nd half of the full array. Therefore, we only consider indexes from `5` to `9`. Note that the lower index changes from `0` to `5`.
- In step 4, in this subarray the middle element is now at index `7` since `(5+9)/2 = 7`.
- Now, we check if the value at index `7` (i.e., `56`) is equal to `72`.
- In step 5, since `72 > 56` we only need to check the 2nd half of the subarray. Therefore, we only consider indexes from `8` to `9`. Note that the lower index changes from `5` to `8`.
- In step 6, in this sub-subarray the middle element is now at index `8` since `(8+9)/2 = 8`.
- Now, we check if the value at index `8` (i.e., `72`) is equal to `72`. Since it is, we return the index as `8`.

Hint: How is the lower index changing relative to the index of the middle element (in step 3 and step 5) in the partial array under consideration?

Let's consider another example where we want to search for element `5` (found at index `1`) in the same array.



Here's how binary search would work:

- In step 1, we cover all the elements in the full array (i.e., from index `0` to `9`). Here, the middle element is the element at index `4` since `(0+9)/2 = 4`.
- In step 2, we check if the value at index `4` (i.e., `16`) is equal to `5`.
- In step 3, since `5 < 16` we only cover the subarray consisting of the 1st half of the full array. Therefore, we only consider indexes from `0` to `3`. Note that the upper index changes from `9` to `3`.
- In step 4, in this subarray the middle element is now at index `1` since `(0+3)/2 = 1`.
- Now, we check if the value at index `1` (i.e., `5`) is equal to `5`. Since it is, we return the index as `1`.

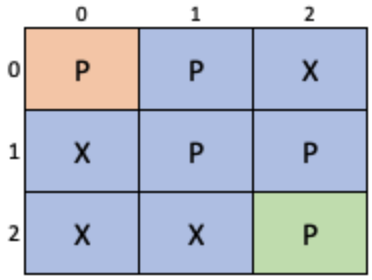
Hint: How is the upper index changing relative to the index of the middle element (in step 5) in the partial array under consideration?

Hint: What will be the size of the partial array when an element is not found in the array? What will be the value of the lower index and the upper index in that case?

Part 2: Implement `mazeSolver` method (50 points)

In `Assignment9.java`, you are provided with a method `mazeSolver(char[][] maze, int row, int col)`. This method takes a 2D array of characters `maze`, and the `row` and `col` index to enter the maze from. This method returns `true` if the maze has an escape route or `false` if it does not have an escape route.

In your starter code, you are provided with three constants: `PATH`, `WALL` and `ESCAPE` which represent a path in the maze, a wall in the maze that's blocking the maze and a path that is a part of the escape route respectively. The below image represents a maze:



For simplicity, you will assume that the `maze` entry is always at the *upper left corner* (colored orange above) of the `maze` and the exit is always at the *bottom right corner* (colored green above) of the `maze`.

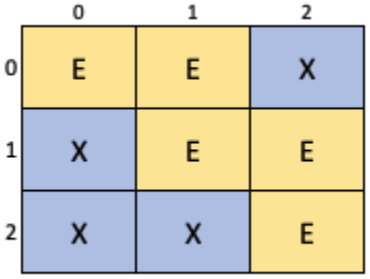
Hint: What are the indexes of the entry and exit of the `maze` with respect to the dimensions of the matrix?

Below are some things to keep in mind.

- To trace the escape route, you can move in *four* directions: **left**, **right**, **up** and **down**.
- If any of the positions have the value `WALL`, it implies that the path is blocked.
- You can only move in any of the positions marked as `PATH`.
- You need to trace the escape route by modifying the `maze` in-place. That is, you record the position of the escape route by using the variable `ESCAPE` to replace `PATH`.
- If such an escape route exists, it must return `true`, else it must return `false`.
- Your method must also do this *recursively*.
- Make sure to check for boundary conditions such as:
    - You cannot move left when you are at the first element of each row.
    - You cannot move up when you are at the first element of each column.
    - You cannot move down when you are in the last row.
    - You cannot move right when you are in the last column.

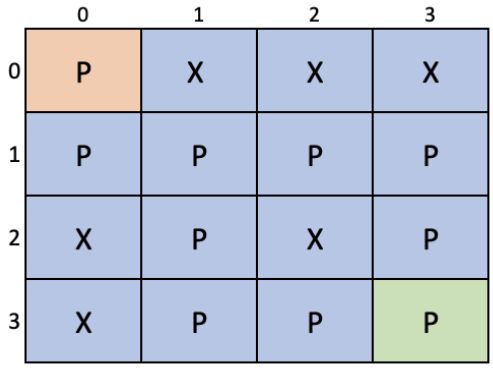
Hint: To do this using recursion, you can think of each position in the `maze` as a sub-maze to solve. What are the *indexes* of the positions you need to check from the *index* of your current position?

For example, after tracing the above example, the `maze` should look like the image below:

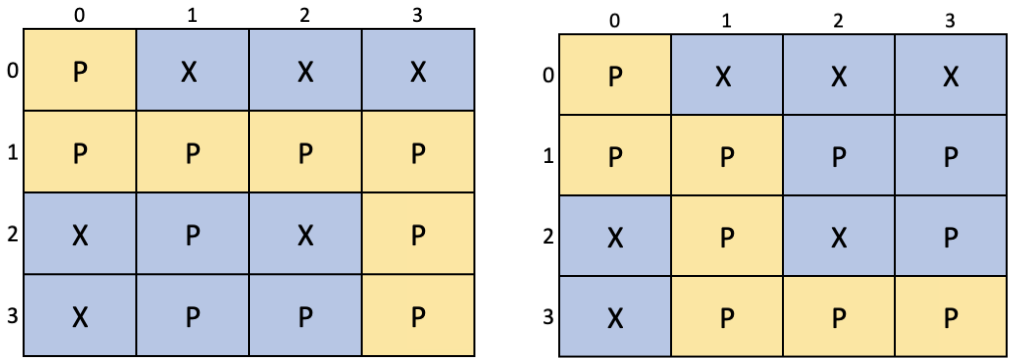


Since there is an escape route from entry to exit, this example returns `true`.

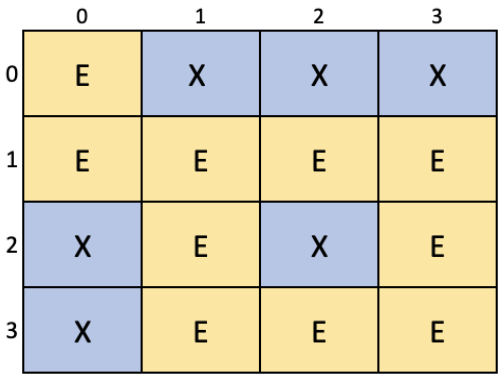
If there are multiple paths available from the entry to exit, mark *all* of the possible escape routes. Consider the below example:



It has two escape routes from entry to exit as shown below:



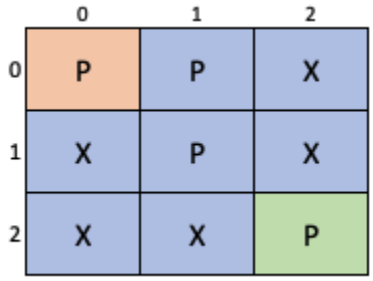
In this case, after tracing the above example, the `maze` will look like the image below.



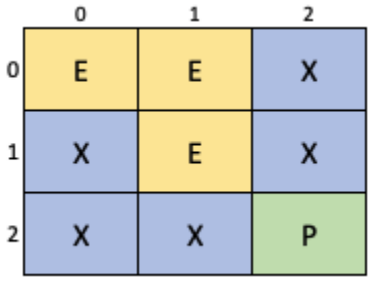
Since there is an escape route from entry to exit, this example also returns `true`.

Note: The autograder checks if your method returns the correct boolean value *and* if your `maze` after tracing has the correct value.

Now, let's consider the below example when there is no escape route.



Similar to the previous example, the entry and exit are marked in orange and green respectively. In this case, your method should trace out the escape route until there is a dead end. That is, until you know that there is no way out of the maze. After tracing, the above maze will look like the below image:



Since there is no escape route from the entry to exit, this example returns `false`.

Hint: Consider both the examples. What are the values of the entry and exit of the `maze` after tracing the escape route? What values in these positions indicate that an escape route exists?

Part 4: Compile, Run and UnitTest Your Code (10 points)

Just like in previous assignments, **in this part of the assignment, you need to implement your own test cases in the method called `unitTests` in the `Assignment9.java` class.**

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method must return `true` only when all the test cases are passed. Otherwise, it must return `false`.

To get full credit for this section, you must create at least six test cases that cover different situations. Since the unitTests method contains one sample test case for each method, you will need to create at least **five** tests that test `binarySearch(int array[], int lower, int upper, int token)` and `mazeSolver(char[][] maze, int row, int col)`.

If a test is not passing, try temporarily printing the result of your method(s) and comparing them to the expected output.

You can compile all the files present in the starter code and run your unit tests from `main()` using the following commands: (Make sure you are in the correct directory, else navigate to the starter code using cd)

```
> javac Assignment9.java
> java Assignment9
```

The first command `javac Assignment9.java` compiles `Assignment9.java`, which is what is required. Now, run the `main()` in `Assignment9` using `java Assignment9`. The below screenshot shows the output for the correct implementation:

Submission

You're almost there! Please follow the instructions below carefully and use the **exact submission format**. Because we will use scripts to grade, **you may receive a zero** if you do not follow the same submission format.

1. Go to Gradescope via Canvas and click on Assignment 9.
2. Click the DRAG & DROP section and directly select the required files (`Assignment9.java`). Drag & drop is fine. Please make sure you do NOT submit a zip, just the one file in one Gradescope submission. Make sure the file name is correct.
3. You can resubmit an unlimited number of times before the due date. Your score will depend on your final (most recent) submission, even if your former submissions have higher scores.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope.

   NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, it will only show the compilation results and the results of basic tests. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests`.**

5. Your submission should look like the screenshot below. **If you have any questions, then feel free to post on Piazza!**

Submit Programming Assignment

Upload all files for your submission

SUBMISSION METHOD

(•) Upload ( ) GitHub ( ) Bitbucket

Add files via Drag & Drop or Browse Files.

NAME	SIZE	PROGRESS ✖
Assignment9.java	**3.1** KB	

STUDENT NAME (OPTIONAL)

Enter student name

Upload Cancel