

Question 1 (10 marks)

a) Write a Java program called Dav.java which reads a positive integer, d, from the command line, and generates the 2^d **Davison** sequence using recursion.

Code :

```
public class Dav {
    Run | Debug
    public static void main(String[] args) {
        int inpt = 5;
        String numString = "0";
        numString = Davison_Seq(numString, inpt);
        System.out.println(numString);}

    public static String Davison_Seq(String numString,int inpt){
        int i = 0;
        if(inpt == 0 )
            return numString;
        else{
            try{
                System.out.println(numString);
                char[] bitwise = numString.toCharArray();
                int str_length = numString.length();
                while(i < str_length){
                    if(bitwise[i] == '1') bitwise[i] = '0';
                    else if (bitwise[i] == '0') bitwise[i] = '1';
                    else bitwise[i] = '@';
                    i++;}
                String newString = String.valueOf(bitwise);
                numString = numString.concat(newString);
            }
            catch (Exception e){
                System.out.println("Error in"+e);
            }
            return Davison_Seq(numString,inpt-1);
        }
    }
}
```

Output :

```
PS E:\HW_Algo\Maze> java Dav
0
01
0110
01101001
0110100110010110
01101001100101101001011001101001
```

b) Calculate the **big-Oh** running time for the program in terms of the size of d.

- term size of d, big-Oh running time คือ $O(n^2)$


- N loop of while ($i < \text{str_length}$), big-Oh running time คือ $O(n)$

running time = (term size of d * number of loop)

$$= O(n^2) * O(n)$$

$$= O(n^3)$$

Question 2 (10 marks)

 pseudo.txt

```

1  step 1 : Initial a array of num
2  step 2 : Initial a value of of check=0, sum=0, total=0, n=0 and i=0
3  step 3 : Assign length of num to num
4  step 4 : While loop if n lower than i
5  step 5 : sum += array of num
6  step 6 : check = (n+1)*(n+2)/2
7  step 7 : total = check - sum
8  step 8 : print total

```

Output : 6

Question 3 (40 marks)

a) Implement the **explore()** function in **MazeSearcher.java**. It must use **backtracking** to search for the exit from the maze, and build a path as it searches. A path is a list of coordinates.

Code :

```
private boolean explore(Maze maze, int x, int y, ArrayList<Coord> path) {
    if (!maze.isValidLoc(x, y) || maze.isWall(x, y) || maze.wasVisited(x, y))
        return false;

    path.add(new Coord(x, y));
    maze.setVisited(x, y);

    if (maze.isExit(x, y))
        return true;

    for (int[] step : STEPS) {
        Coord coord = getNextCoord(x, y, step);
        if (explore(maze, coord.getX(), coord.getY(), path))
            return true;
    }
    path.remove(path.size() - 1);
    return false;
}

private Coord getNextCoord(int x, int y, int[] step) {
    return new Coord(x + step[0], y + step[1]);
}
```

Output :

Maze1.txt

```
PS E:\HW_Algo\Maze> javac *.java
PS E:\HW_Algo\Maze> java MazeSearcher maze1.txt
#S#####
#       #
#   ##  ##
#  #   #  #
# #   #  #
# ## #####
# #       #
# # #   #  #
#####
#   #   E
# #   #  #
#####
```

```
PS E:\HW_Algo\Maze> javac *.java
PS E:\HW_Algo\Maze> java MazeSearcher maze1.txt
#S#####
#  L   #
#   ##  ##
#  #   #  #
# #   #  #
# ## #####
# #       #
# # #   #  #
#####
#   #   E
# #   #  #
#####
```

Maze2.txt

```

PS E:\HW_Algo\Maze> java MazeSearcher maze2.txt
#####
# #   #                               #
# # #### #####                       #
# # #   # #                               #
# # #### # # #####                   #
# # #   # # #                               #
# # # #### # # #####                 #
# # # #   #                               #
#   #####                               #
#####   # # #   # #
# #   ##### # ##### # #
# # ### # # # # #   # #
#   #   # # # ##### # #
##### ##### # #   # # #
#   #   # # # ##### # #
##### #####   # #
#   #   ##### #
# ##### ##### # #   # E
#   #   #   # # #
#####

PS E:\HW_Algo\Maze> java MazeSearcher maze2.txt
#####
# #   #                               #
# # #### #####                       #
# # #   # #                               #
# # #### # # #####                   #
# # #   # # #                               #
# # # #### # # #####                 #
# # # #   #                               #
#   #####                               #
#####   # # #   # #
# #   ##### # ##### # #
# # ### # # # # #   # #
#   #   # # # ##### # #
##### ##### # #   # # #
#   #   # # # ##### # #
##### #####   # #
#   #   ##### #
# ##### ##### # #   # E
#   #   #   # # #
#####

```

b) Explain in words (and perhaps with diagrams) why the Maze class includes the methods `wasVisited()` and `setVisited()`. *Hint: if your `explore()` function does not use these functions, then your search may take a very, very long time.*

Maze class มีวิธีการที่ต่อเนื่อง เพราะ ฟังก์ชัน `wasVisited()` ทำให้ไฟล์ maze จำเส้นทางที่เคยไปมาแล้ว จึงทำให้ไม่ต้องไปซ้ำอีก และ ฟังก์ชัน `setVisited()` ทำให้ maze บันทึกเส้นทางระหว่างการค้นหาเส้นทางอยู่ด้วย ซึ่งมี 2 วิธี คือ ช่วยหาเส้นทางที่ถูกต้องและหาเส้นทางที่สั้นที่สุดที่จะทำให้ไปถึงจุดหมายได้อย่างรวดเร็ว