Algorithm and Data Structures

Assignment 10: Report

Name: Maeda Taishin

Student ID: 1W21CF15

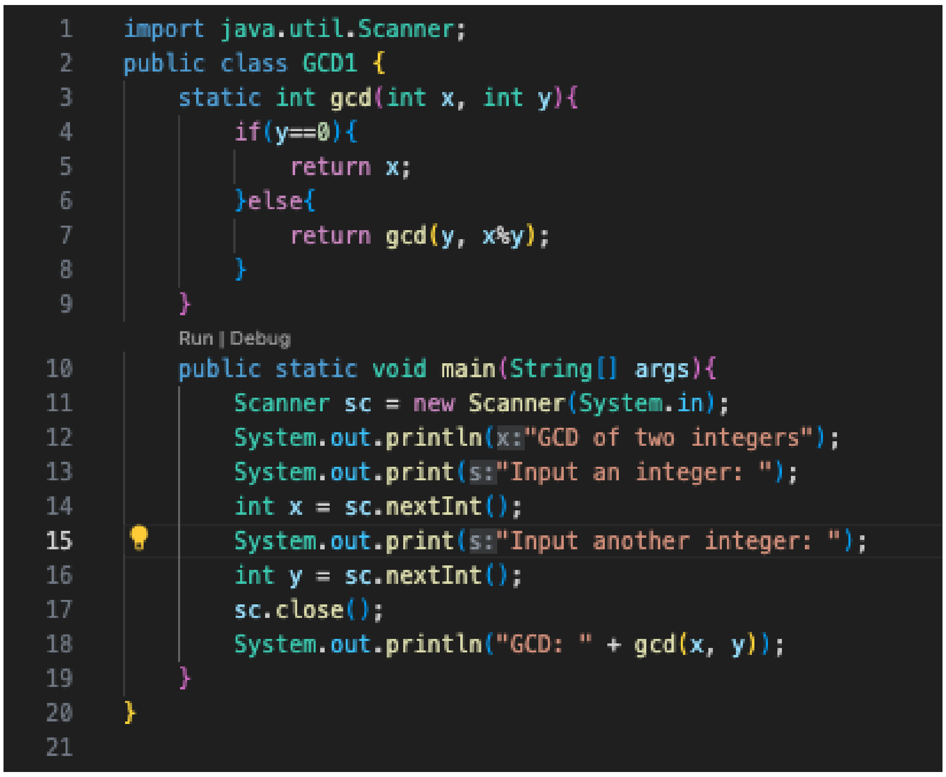
Date: 20/06/2023

Ex. 1: Euclid’s Algorithm

- Write Euclid’s algorithms to find GCD by using **recursion** in Java
  - Ex.1-1: GCD of two integer values
  - Ex.1-2: GCD of all elements in an integer array

  - Assumptions
    - Integer values are initialized by using data from the standard input
      - use java.util.Scanner

Ex. 1-1: GCD of two integer value



```
 1  import java.util.Scanner;
 2  public class GCD1 {
 3      static int gcd(int x, int y){
 4          if(y==0){
 5              return x;
 6          }else{
 7              return gcd(y, x%y);
 8          }
 9      }
        Run | Debug
10      public static void main(String[] args){
11          Scanner sc = new Scanner(System.in);
12          System.out.println(x:"GCD of two integers");
13          System.out.print(s:"Input an integer: ");
14          int x = sc.nextInt();
15          System.out.print(s:"Input another integer: ");
16          int y = sc.nextInt();
17          sc.close();
18          System.out.println("GCD: " + gcd(x, y));
19      }
20  }
21
```

Code Explanation:

<u>Line 1:</u>

Import Java package (Scanner in this case)

<u>Lines 3-9:</u>

The function called “gcd” is created to return the greatest common divisor of two integers x and y that the user prompted. For instance, two integers x and y are passed into this function, and if the value of “y” is equal to 0, then the program will return 0. Otherwise, the program will return the value of “gcd(y, x%y)” using the Euclidean algorithm. Consider x = ay + r where a is the multiplier and r is the remainder, gcd(x, y) will be gcd(ay + r, y). Then, it will be gcd(y, r) (in other words: gcd(old y, new y)), leading to gcd(y, x%y) (in other words: gcd(new y, old y%new y)).

The main function (Lines 10-19):

Line 11:

The Scanner “sc” is initialized.

Lines 12-16:

Asking the user to input two integers x and y respectively and scanned.

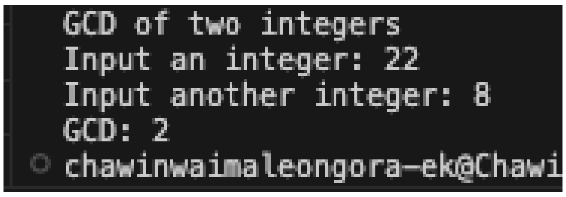
Line 17:

Close the Scanner “sc”.

Line 18:

Let the program print the greatest common divisor of x and y by calling the function “gcd”.

Execution Result:



```
GCD of two integers
Input an integer: 22
Input another integer: 8
GCD: 2
chawinwaimaleongora-ek@Chawi
```

The greatest common divisor of 22 and 8 is 2.

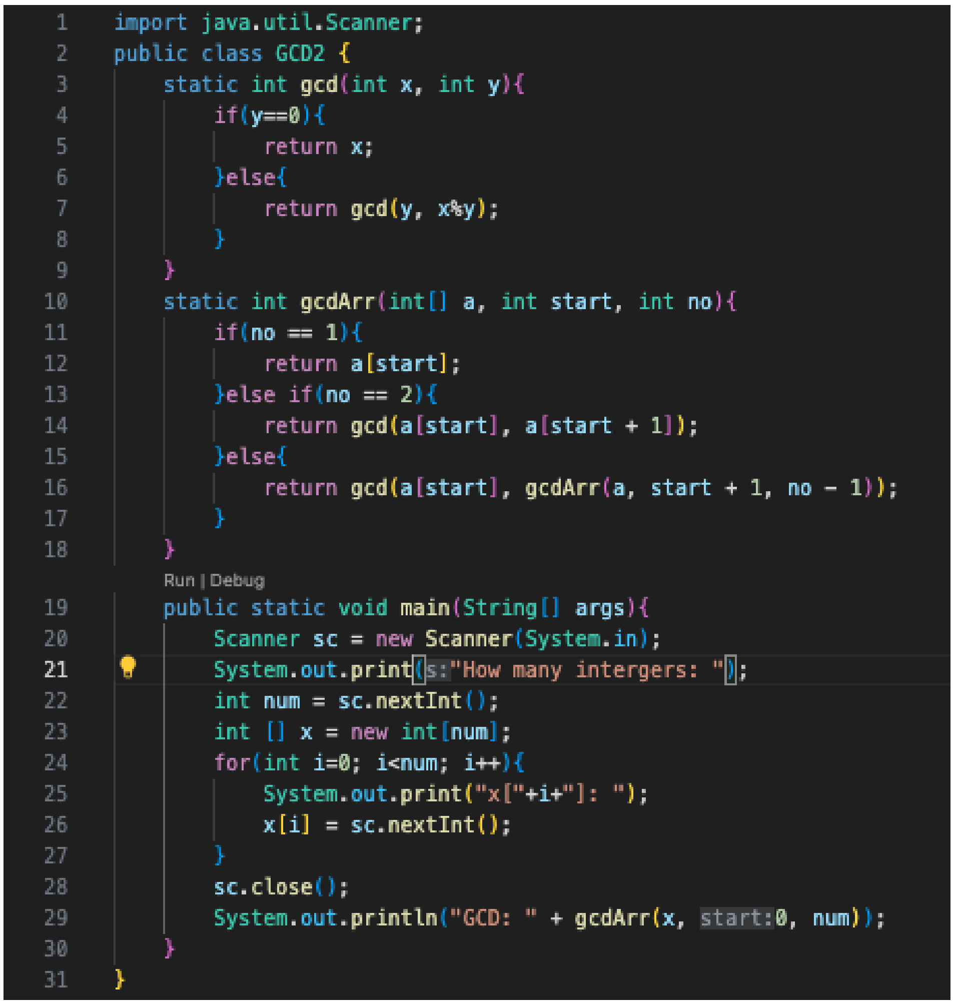
gcd(8, 22%8) = gcd(8, 6)

gcd(6, 8%6) = gcd(6, 2)

gcd(2, 6%2) = gcd(2,0)

Thus, the program will return x = 2.

Ex. 1-2: GCD of all elements in an integer array



```
1   import java.util.Scanner;
2   public class GCD2 {
3       static int gcd(int x, int y){
4           if(y==0){
5               return x;
6           }else{
7               return gcd(y, x%y);
8           }
9       }
10      static int gcdArr(int[] a, int start, int no){
11          if(no == 1){
12              return a[start];
13          }else if(no == 2){
14              return gcd(a[start], a[start + 1]);
15          }else{
16              return gcd(a[start], gcdArr(a, start + 1, no - 1));
17          }
18      }
        Run | Debug
19      public static void main(String[] args){
20          Scanner sc = new Scanner(System.in);
21          System.out.print(s:"How many intergers: ");
22          int num = sc.nextInt();
23          int [] x = new int[num];
24          for(int i=0; i<num; i++){
25              System.out.print("x["+i+"]: ");
26              x[i] = sc.nextInt();
27          }
28          sc.close();
29          System.out.println("GCD: " + gcdArr(x, start:0, num));
30      }
31  }
```

Code Explanation:

Line 1:

Import Java package (Scanner in this case)

Lines 3-9:

The function called “gcd” is created to return the greatest common divisor of two integers x and y that the user prompted. For instance, two integers x and y are passed into this function, and if the value of “y” is equal to 0, then the program will return 0. Otherwise, the program will return the value of “gcd(y, x%y)” using the Euclidean algorithm. Consider x = ay + r where a is the multiplier and r is the remainder, gcd(x, y) will be gcd(ay + r, y). Then, it will be gcd(y, r) (in other words: gcd(old y, new y)), leading to gcd(y, x%y) (in other words: gcd(new y, old y%new y)).

Lines 10-16:

The function called “gcdArr” is created to return the greatest common divisor of “no” elements of an array “a” from the index named “start”. There are three parameters included in this function which are array “a”, starting index “start”, and the number of elements “no”. Inside the function, the program will return a[start] if the value of “no” is equal to 1. However, if the value of “no” is equal to 2, the program will return the greatest common divisor (GCD) of a[start] and a[start + 1] using the gcd function above. Otherwise, the program will return GCD if a[start] and gcdArr(a, start + 1, no -1) using the gcd function above. This means that the program will return the GCD of "no" minus one element of the array "a" starting from index start plus one.

<u>The main function (Lines 19-30):</u>

<u>Line 20:</u>

Scanner "sc" is initialized.

<u>Lines 21-22:</u>

Asking the user to input a number of integers by scanning it and storing it to the variable "num".

<u>Line 23:</u>

Create an array "x" with the size of "num".

<u>Lines 24-27:</u>

A for-loop is defined to ask the user to input the element(s) at index "i" from 0 to "num" minus 1 of the array "x" and store to position "x[i]".

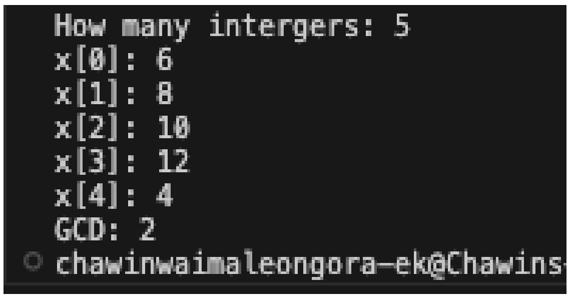
<u>Line 28:</u>

Close the Scanner "sc".

<u>Line 29:</u>

Let the program print the greatest common divisor of all elements of array "x" by calling the function "gcdArr".

Execution result:



```
How many intergers: 5
x[0]: 6
x[1]: 8
x[2]: 10
x[3]: 12
x[4]: 4
GCD: 2
○ chawinwaimaleongora-ek@Chawins
```

The greatest common divisor of 5 integers including 6, 8, 10, 12, and 4 is 2.

By using the recursion, gcd(6, 8, 10, 12, 4) will be:

= gcd(6, gcd(8, 10, 12, 4))

= gcd(6, gcd(8, gcd(10, 12, 4)))

= gcd(6, gcd(8, gcd(10, gcd(12, 4))))

= gcd(6, gcd(8, gcd(10, 4)))

= gcd(6, gcd(8, 2))

= gcd(6, 2)

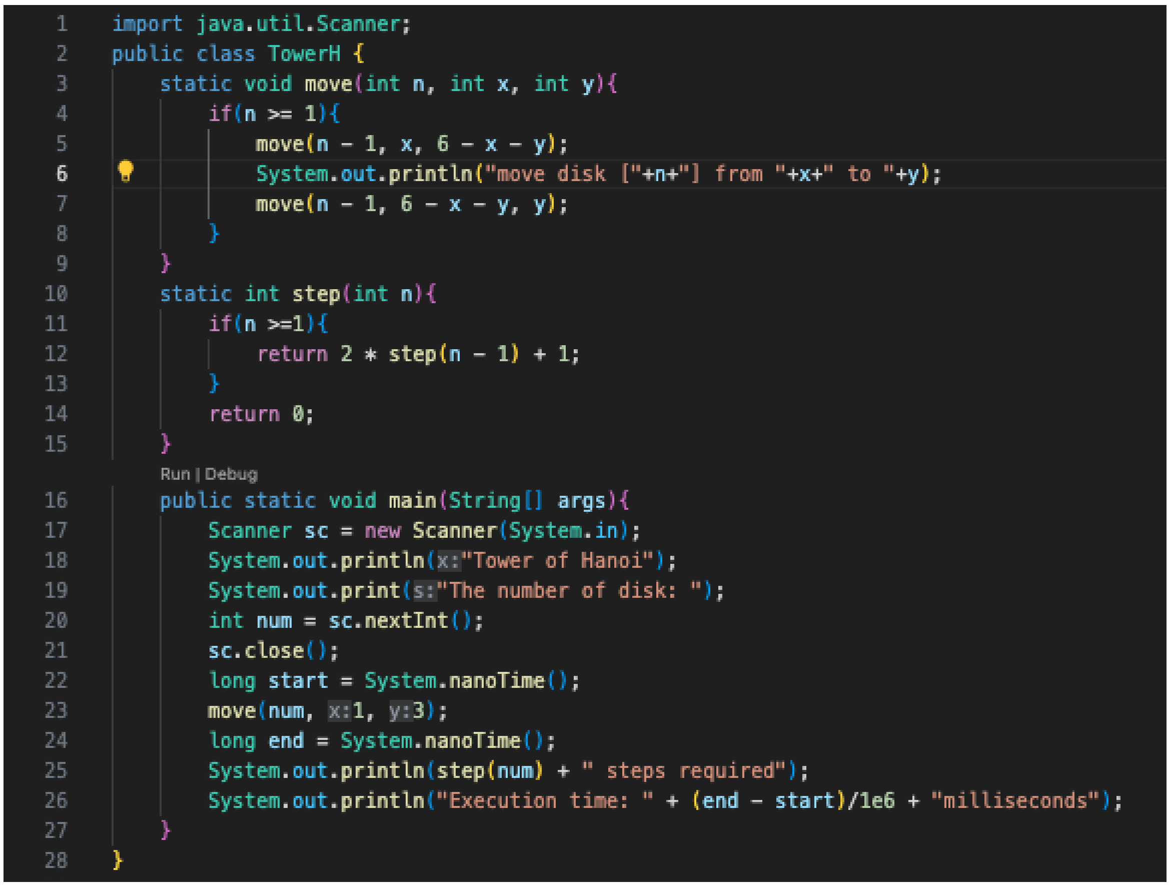
= 2

Ex. 2: Tower of Hanoi

- Write algorithm to solve Tower of Hanoi in Java
  - Ex2-1: show the sequence of actions in the standard output
  - Ex2-2: extend Ex2-1 program to show the total number of steps

  - Assumptions
    - The number of disk is initialized by using data from the standard input

Ex. 2-1 and 2-2: Tower of Hanoi



```
 1  import java.util.Scanner;
 2  public class TowerH {
 3      static void move(int n, int x, int y){
 4          if(n >= 1){
 5              move(n - 1, x, 6 - x - y);
 6              System.out.println("move disk ["+n+"] from "+x+" to "+y);
 7              move(n - 1, 6 - x - y, y);
 8          }
 9      }
10      static int step(int n){
11          if(n >=1){
12              return 2 * step(n - 1) + 1;
13          }
14          return 0;
15      }
        Run | Debug
16      public static void main(String[] args){
17          Scanner sc = new Scanner(System.in);
18          System.out.println(x:"Tower of Hanoi");
19          System.out.print(s:"The number of disk: ");
20          int num = sc.nextInt();
21          sc.close();
22          long start = System.nanoTime();
23          move(num, x:1, y:3);
24          long end = System.nanoTime();
25          System.out.println(step(num) + " steps required");
26          System.out.println("Execution time: " + (end - start)/1e6 + "milliseconds");
27      }
28  }
```

Code Explanation:

<u>Line 1:</u>

Import Java package (Scanner in this case)

<u>Lines 3-9:</u>

A function called "move" is created to move "n" number of disk(s) from tower x to tower y. Note that, "x" and "y" indicate the source tower and destination tower respectively. Inside the function, an if-else statement is used in order to check whether the value of “n” is at least 1. If so, then the program will move the first “n” minus 1 disk(s) from “x”, the source tower, to “6-x-y”, the aux tower. For instance, 6-x-y indicates another tower aside from tower x and tower y due to the fact that there are three towers: 1, 2, and 3. After that, the program will move the last disk (nth disk) from tower x to tower y. Then, in the last step, the program will move the rest of the disks (n-1 disks) from tower 6-x-y to tower y.

<u>Lines 10-15:</u>

A function called “step” is created aiming to keep tracking and returning the number of step(s) required to move “n” disk(s). The value of “n” is passed into this function representing the number of disk(s). Then, an if-else statement is created. If the value of “n” is at least 1, then the program will return 2*step(n-1)+1. Meaning that the program will return step(n-1) for the first step, 1 for the second step, and step(n-1)for the last step. Or else, the program will return 0.

<u>The main function (Lines 16-27):</u>

<u>Line 17:</u>

Scanner “sc” is initialized.

<u>Lines 18-20:</u>

Asking the user to input a number of disk(s) by scanning it and storing it to the variable “num”.

<u>Line 21:</u>

Close the Scanner “sc”.

<u>Line 22 and 24:</u>

Several syntaxes are included to measure the execution time of the program after calling the select “move” function. For instance, the data type “long” is used since it contains the minimum value of -2 to the power of 63 and a maximum value of, 2 to the power of 63, minus 1, so the program can return the time in nanoseconds when calculating the time interval. The program is designed to record the current time right before entering the function in line 23 and record again after the function is called and executed.

<u>Line 23:</u>

The function “move” is called by passing “num”, 1, and 3 to the function to move the “n” disk(s) from tower 1, source tower, to tower 3, destination tower.

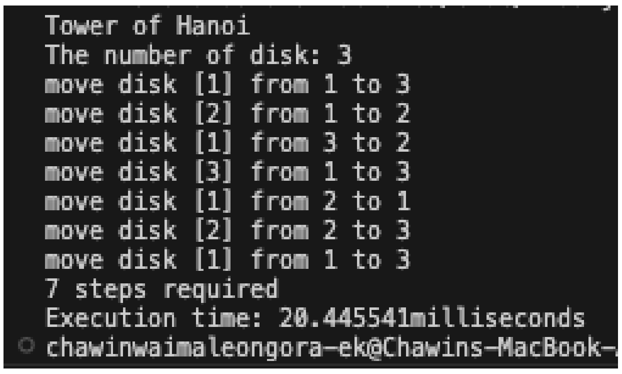
<u>Line 25:</u>

The number of step(s) will be printed using the “step” function.

<u>Line 26:</u>

The amount of time spent will be calculated by subtracting the “end” and “start” which are the time after and before respectively and printing it out in line 26 of this program. Note that, the time difference is divided by 1e6 (10^6) to get the time in milliseconds.

Execution result (n=3):



```
Tower of Hanoi
The number of disk: 3
move disk [1] from 1 to 3
move disk [2] from 1 to 2
move disk [1] from 3 to 2
move disk [3] from 1 to 3
move disk [1] from 2 to 1
move disk [2] from 2 to 3
move disk [1] from 1 to 3
7 steps required
Execution time: 20.445541milliseconds
○ chawinwaimaleongora-ek@Chawins-MacBook-
```

When the user inputs the number of the disk as 3, there will be 7 steps (2*3+1) required to complete.

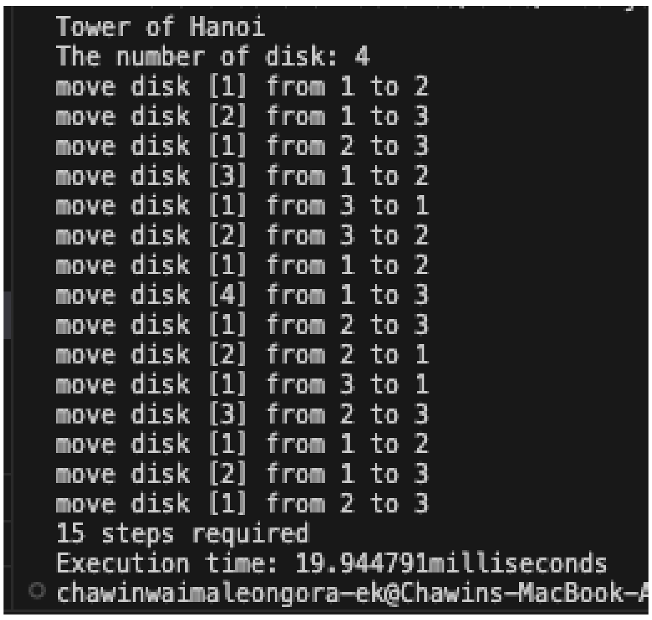
By following the steps:

Step 1-3: The two uppermost disks are moved from tower 1 (source tower) to tower 2 (aux tower).

Step 4: The last (3rd disk) disk is moved from tower 1 to tower 3 (source tower to destination tower).

Step 5-7: The two disks in Tower 2 are logically moved to Tower 3 (aux tower to destination tower).

Execution result (n=4):



```
Tower of Hanoi
The number of disk: 4
move disk [1] from 1 to 2
move disk [2] from 1 to 3
move disk [1] from 2 to 3
move disk [3] from 1 to 2
move disk [1] from 3 to 1
move disk [2] from 3 to 2
move disk [1] from 1 to 2
move disk [4] from 1 to 3
move disk [1] from 2 to 3
move disk [2] from 2 to 1
move disk [1] from 3 to 1
move disk [3] from 2 to 3
move disk [1] from 1 to 2
move disk [2] from 1 to 3
move disk [1] from 2 to 3
15 steps required
Execution time: 19.944791milliseconds
○ chawinwaimaleongora-ek@Chawins-MacBook-
```

When the user inputs the number of the disk as 4, there will be 15 steps (2*7+1) required to complete.

By following the steps:

Step 1-7: The three uppermost disks are moved from tower 1 (source tower) to tower 2 (aux tower).

Step 8: The last (4th disk) disk is moved from tower 1 to tower 3 (source tower to destination tower).

Step 9-15: The three disks in Tower 2 are logically moved to Tower 3 (aux tower to destination tower).

Discussion on the performance of the Tower of Hanoi algorithm:

Number of Disks	Steps Required	Execution Time (milliseconds)
5	31	20.620042
10	1023	32.437708
15	32767	218.424916
20	1048575	4528.092958

Table 1. The number of steps and execution time in milliseconds of the Tower of Hanoi program with respect to the number of disks

From Table 1, it can be seen that the greater the number of disks, the longer the time that the program will take to execute and the more steps are required. For the number of the steps, it can be calculated by $2^n - 1$ where $n \geq 1$ as follows: $2^5 - 1 = 31$, $2^{10} - 1 = 1023$, $2^{15} - 1 = 32767$, and $2^{20} - 1 = 1048575$. This shows that the time complexity of this algorithm is $T(n) = O(2^n - 1)$ or $O(2^n)$ which means that it has exponential time in complexity. Hence, the performance of the code will be significantly affected depending on the number of disks.