FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMANT

2022-2023 SPRING

CSE1120 DISCRETE STRUCTURES

COMPUTER PROJECT

Student Id	Name Surname	1st /2nd Ed.
210316011	Furkan Bulut	2nd
210316080	Melika Mokari	2nd

Question 1:

1. question 8

Mathmatical induction

1. Base Case:

$n=0 \quad n=1$

2. induction step

that proves statelement is true for any given case

$n=k$, it also true for $(n=k+1)$

1. Base Case

$1 = 2^0 + 1 - 1$

$1 = 1$

true

2. we need to assume that $n=k$

$n=k+1$

it's going to be true in both case

we should add $2^{(k+1)}$ on both side

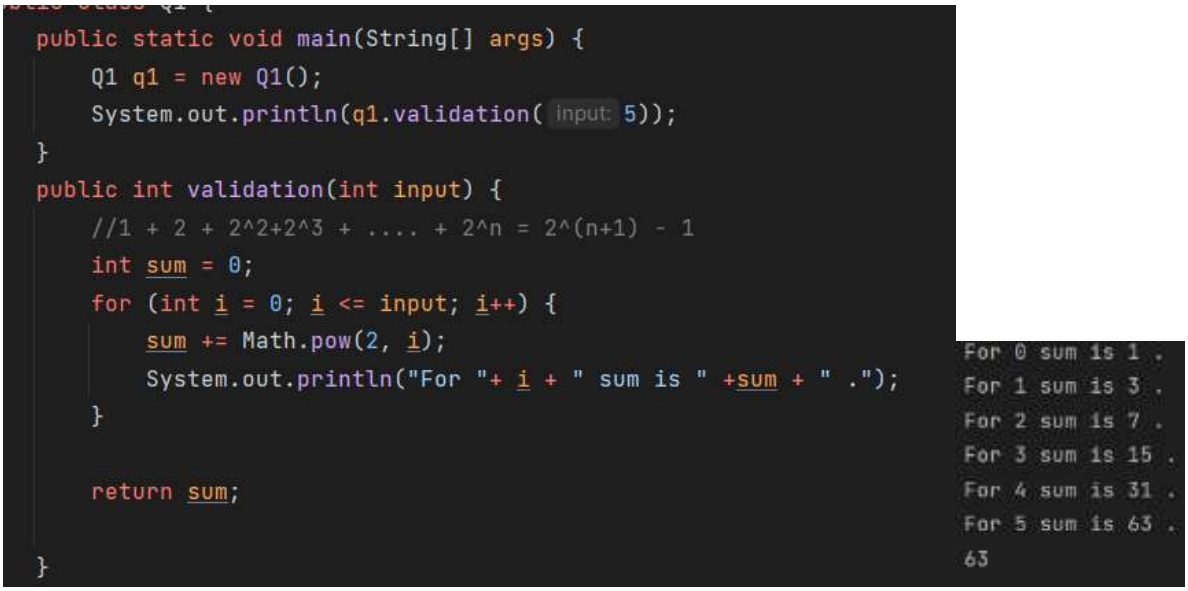
Rearrenge the right han

$1+2+\ldots 2^k + 2^{k+1} = 2^{k+1} - 1) + 2^{k+1}$

$= 2 \times 2^{k+1} - 1$

$= 2^k + 2 - 1$

this equation is true for all cases.



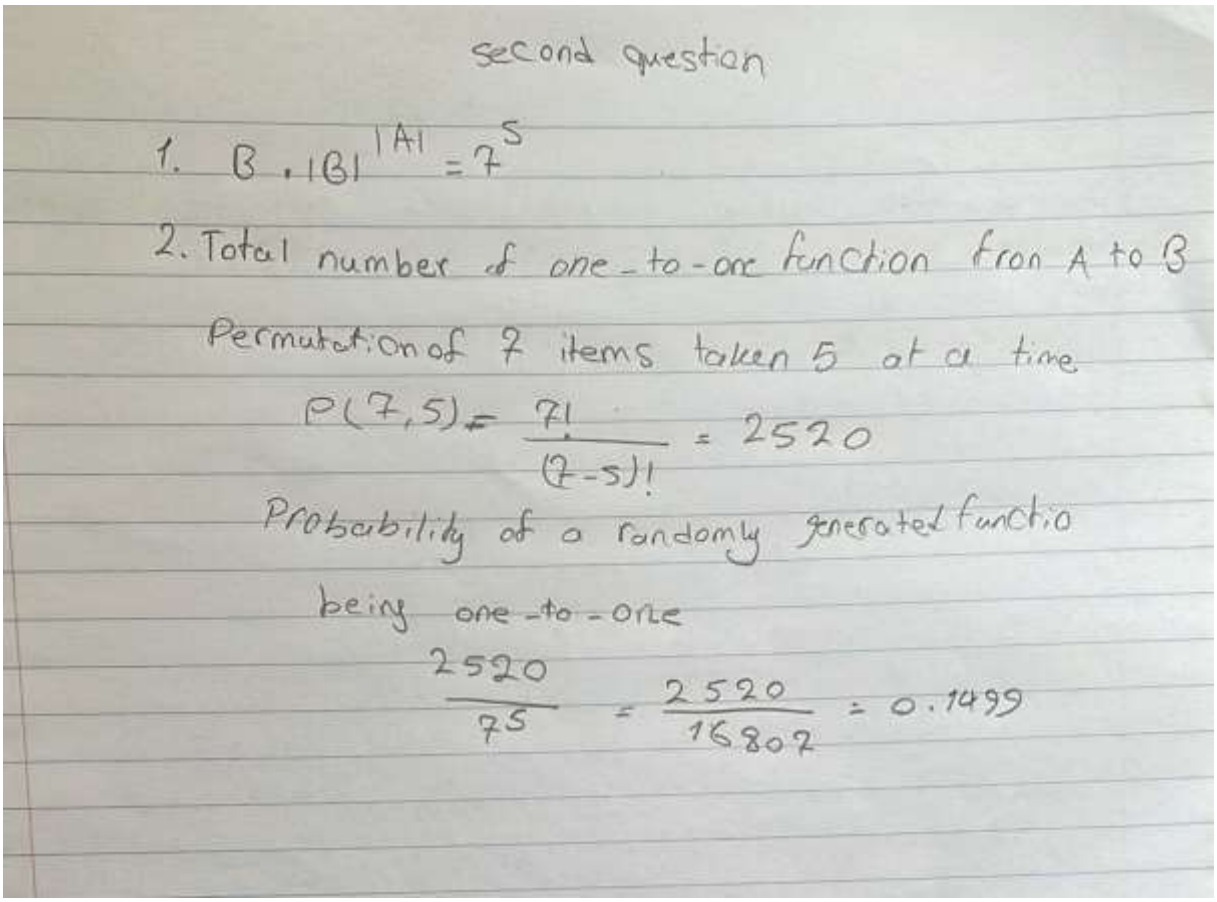
```
public class Q1 {
    public static void main(String[] args) {
        Q1 q1 = new Q1();
        System.out.println(q1.validation( input: 5));
    }
    public int validation(int input) {
        //1 + 2 + 2^2+2^3 + .... + 2^n = 2^(n+1) - 1
        int sum = 0;
        for (int i = 0; i <= input; i++) {
            sum += Math.pow(2, i);
            System.out.println("For "+ i + " sum is " +sum + " .");
        }

        return sum;

    }
}
```

```
For 0 sum is 1 .
For 1 sum is 3 .
For 2 sum is 7 .
For 3 sum is 15 .
For 4 sum is 31 .
For 5 sum is 63 .
63
```

Question 2:



second question

1. $B \cdot |B|^{|A|} = 7^5$

2. Total number of one-to-one function from A to B

Permutation of 7 items taken 5 at a time

$$P(7,5) = \frac{7!}{(7-5)!} = 2520$$

Probability of a randomly generated functio being one-to-one

$$\frac{2520}{7^5} = \frac{2520}{16807} = 0.1499$$

First Solution

```
public static boolean isOneToOne(int[] f) {
    HashSet<Integer> outputs = new HashSet();

    for(int i = 0; i < f.length; ++i) {
        if (outputs.contains(f[i])) {
            return false;
        }

        outputs.add(f[i]);
    }

    return true;
}
```

```
public static void main(String[] args) {
    int[] A = new int[]{1, 2, 3, 4, 5};
    int[] B = new int[]{20, 21, 22, 23, 24, 25, 26};
    int n = 100;
    int count = 0;
    Random random = new Random();

    for(int i = 0; i < n; ++i) {
        int[] f = new int[A.length];

        for(int j = 0; j < A.length; ++j) {
            f[j] = B[random.nextInt(B.length)];
        }

        if (isOneToOne(f)) {
            ++count;
        }
    }

    System.out.println("Out of " + n + " random functions from A to B, " + count + " are one to one.");
}
```

```
Out of 100 random functions from A to B, 15 are one to one.
```

Second Solution

```java
import java.util.*;
public class Q2M {
    public static void main(String[] args) {
        int[] A = {1, 2, 3, 4, 5};
        int[] B = {1, 2, 3, 4, 5, 6, 7};

        int oneToOneCount = 0;
        int totalFunctions = (int) Math.pow(B.length, A.length);

        Random rand = new Random();
        for (int i = 0; i < totalFunctions; i++) {
            Map<Integer, Integer> function = new HashMap<>();
            for (int a : A) {
                function.put(a, B[rand.nextInt(B.length)]);
            }
            if (isOneToOne(function)) {
                oneToOneCount++;
            }
        }

        System.out.println("Number of one-to-one functions: " + oneToOneCount);
    }

    public static boolean isOneToOne(Map<Integer, Integer> function) {
        Set<Integer> imageSet = new HashSet<>(function.values());
        return imageSet.size() == function.size();
    }
}
```

```
Number of one-to-one functions: 2540

Process finished with exit code 0
```

Question 3:

```java
public static int lucas(int n) {
    if (n == 0) {
        return 2;
    } else {
        return n == 1 ? 1 : lucas( n: n - 1) + lucas( n: n - 2);
    }
}
```

```java
public static void printLucas(int n) {
    int[] lucas = new int[n];
    lucas[0] = 2;
    lucas[1] = 1;

    for(int i = 2; i < n; ++i) {
        lucas[i] = lucas(i);
    }

    int[] var6 = lucas;
    int var3 = lucas.length;

    for(int var4 = 0; var4 < var3; ++var4) {
        int num = var6[var4];
        System.out.print("" + num + " ");
    }

}
```

```java
public static void main(String[] args) {
    // Print the first 10 Lucas numbers
    printLucas( n: 10);
}
```

```
2 1 3 4 7 11 18 29 47 76
Process finished with exit code 0
```

Question 4:

```java
public class WeightedEdge {
    private GraphNode neighbor;
    private int weight;

    public WeightedEdge(GraphNode neighbor, int weight) {
        this.neighbor = neighbor;
        this.weight = weight;
    }

    public GraphNode getNeighbor() { return this.neighbor; }

    public int getWeight() { return this.weight; }
}
```

```java
public class GraphNode {
    private String name;
    private List<WeightedEdge> neighbors;

    public GraphNode(String name) {
        this.name = name;
        this.neighbors = new ArrayList();
    }

    public String getName() { return this.name; }

    public List<WeightedEdge> getNeighbors() { return this.neighbors; }

    public void addNeighbor(GraphNode neighborName, int weight) {
        WeightedEdge edge = new WeightedEdge(neighborName, weight);
        this.neighbors.add(edge);
    }

    public void removeEdge(GraphNode neighborName) {
        for(int i = 0; i < this.neighbors.size(); ++i) {
            if (((WeightedEdge)this.neighbors.get(i)).getNeighbor() == neighborName) {
                this.neighbors.remove(i);
                return;
            }
        }

    }
}
```

Main Class

```
public static void main(String[] args) {
    GraphNode sanfrancisco = new GraphNode( name: "San Francisco");
    GraphNode losangeles = new GraphNode( name: "Los Angeles");
    GraphNode detroit = new GraphNode( name: "Detroit");
    GraphNode newyork = new GraphNode( name: "New York");
    GraphNode denver = new GraphNode( name: "Denver");
    sanfrancisco.addNeighbor(losangeles, weight: 69);
    sanfrancisco.addNeighbor(detroit, weight: 329);
    sanfrancisco.addNeighbor(newyork, weight: 359);
    sanfrancisco.addNeighbor(denver, weight: 179);
    losangeles.addNeighbor(detroit, weight: 349);
    losangeles.addNeighbor(newyork, weight: 379);
    losangeles.addNeighbor(denver, weight: 209);
    losangeles.addNeighbor(sanfrancisco, weight: 69);
    newyork.addNeighbor(denver, weight: 279);
    newyork.addNeighbor(detroit, weight: 189);
    newyork.addNeighbor(sanfrancisco, weight: 359);
    newyork.addNeighbor(losangeles, weight: 379);
    denver.addNeighbor(detroit, weight: 229);
    denver.addNeighbor(newyork, weight: 279);
    denver.addNeighbor(sanfrancisco, weight: 179);
    denver.addNeighbor(losangeles, weight: 209);
    detroit.addNeighbor(newyork, weight: 189);
    detroit.addNeighbor(losangeles, weight: 349);
    detroit.addNeighbor(sanfrancisco, weight: 329);
    detroit.addNeighbor(denver, weight: 229);
    int totalCost = 0;
```

```
Scanner scanner = new Scanner(System.in);
System.out.println("Please enter the city you want to start: (Los Angeles, San Francisco, New York, Denver, Detroit))");
String city = scanner.nextLine();
GraphNode current = null;
switch (city) {
    case "San Francisco":
        current = sanfrancisco;
        break;
    case "Los Angeles":
        current = losangeles;
        break;
    case "New York":
        current = newyork;
        break;
    case "Denver":
        current = denver;
        break;
    case "Detroit":
        current = detroit;
        break;
    default:
        System.out.println("Please enter a valid city name!");
}
```

```java
    ArrayList<GraphNode> visitedCities = new ArrayList<>();
    while (current.getNeighbors().size() > 0) {
        GraphNode minNode = null;
        int minCost = Integer.MAX_VALUE;
        for (WeightedEdge edge : current.getNeighbors()) {
            if (edge.getWeight() < minCost && !visitedCities.contains(edge.getNeighbor())) {
                minCost = edge.getWeight();
                minNode = edge.getNeighbor();} }
        if (minNode == null) { break;}
        current.removeEdge(minNode);
        minNode.removeEdge(current);
        visitedCities.add(current);
        System.out.println("-------------------------------------------------");
        System.out.println("Your current location is " + current.getName());
        System.out.println("Going to " + minNode.getName() + " with cost " + minCost + "$");
        totalCost += minCost;
        current = minNode;}
    System.out.println("-------------------------------------------------");
    System.out.println("You have been visited in order: ");
    for (GraphNode cityNode : visitedCities) {
        System.out.print(cityNode.getName()+" -> " );}
    System.out.println(current.getName());
    System.out.println("Total cost: " + totalCost+ "$");
}
```

```
Please enter the city you want to start: (Los Angeles, San Francisco, New York, Denver, Detroit))
Los Angeles
-------------------------------------------------
Your current location is Los Angeles
Going to San Francisco with cost 69$
-------------------------------------------------
Your current location is San Francisco
Going to Denver with cost 179$
-------------------------------------------------
Your current location is Denver
Going to Detroit with cost 229$
-------------------------------------------------
Your current location is Detroit
Going to New York with cost 189$
-------------------------------------------------
You have been visited in order:
Los Angeles -> San Francisco -> Denver -> Detroit -> New York
Total cost: 666$

Process finished with exit code 0
```