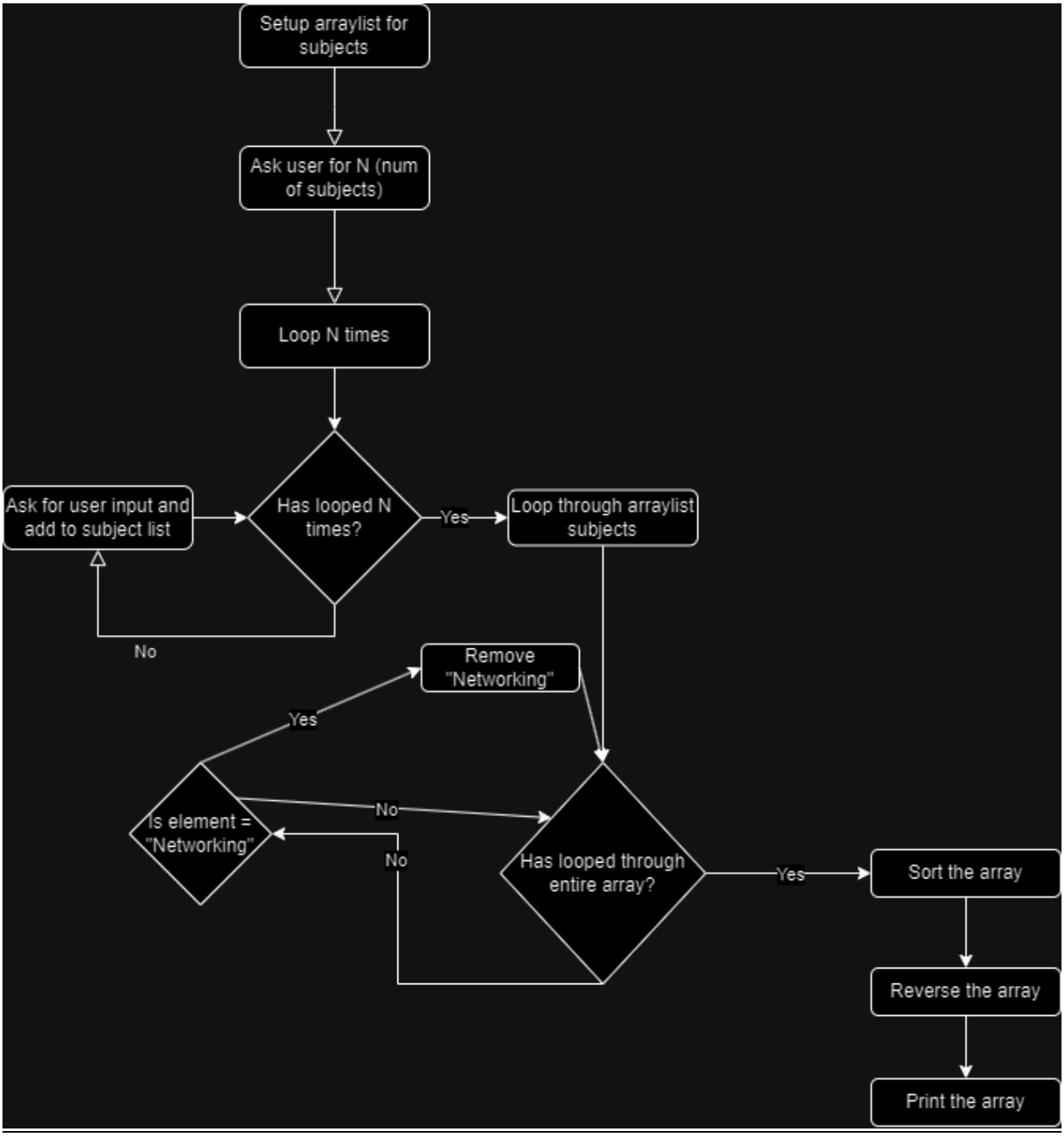
Problem 1

Code

```java
import java.util.ArrayList;
import java.util.Scanner;
import java.util.Collections;

public class subjects {
    public static void main(String[] args) {
        // setup list
        ArrayList<String> subjects = new ArrayList<String>();

        // setup user input, ask for number of subjects and ask for subjects
        Scanner sc = new Scanner(System.in); //setup a scanner object
        System.out.print("Number of subjects: ");   // ask user for a number of subjects
        int length = sc.nextInt();  // scan number of subjects inputted
        sc.nextLine();      // to go next line to avoid being on the same line
        System.out.println("Give computer science subjects: "); // ask user for N subjects

        // loop N times to ask for input
        for (int i = 0; i < length; i++){
            subjects.add(sc.nextLine());    // add each line to the array
        }
        sc.close(); // close canner

        // loop through arraylist until we find "networking" and remove it
        for (int i = 0; i < subjects.size() ; i++){
            if (subjects.get(i).toLowerCase().equals("networking")){    // check if lower case is equal to networking
                subjects.remove(i); // if it is remove it
                i -= 1;     // move i back by one because the length of the list has reduced
            }
        }

        // sort and reverse list, then print
        Collections.sort(subjects, String.CASE_INSENSITIVE_ORDER);  // sort the list, ignore case
        Collections.reverse(subjects);  // reverse the list
        System.out.println(subjects);   // print the list

    }
}
```

Logic



Results

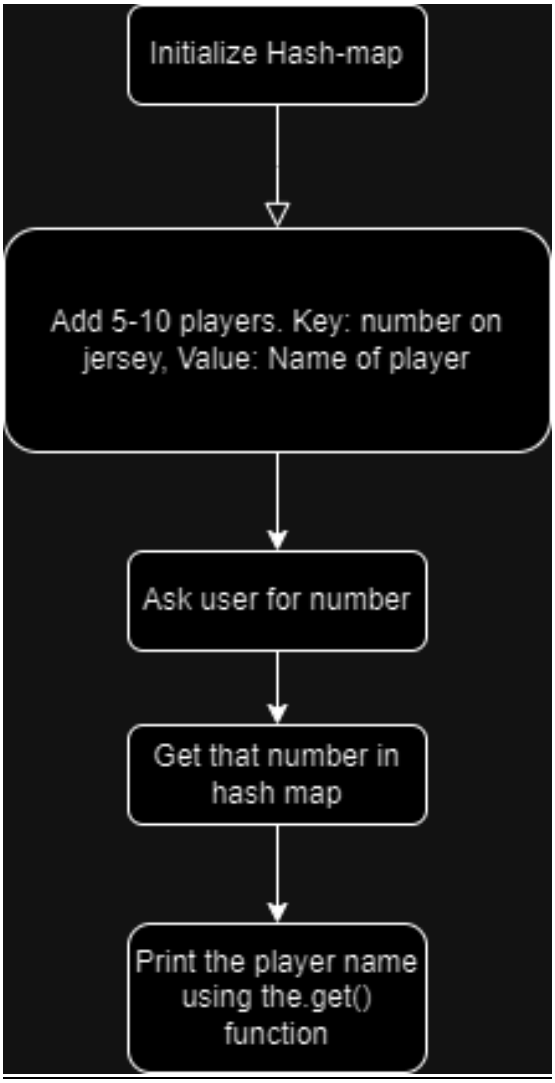
```
Number of subjects: 4
Give computer science subjects:
Crypotography
Database
OS
Networking
[OS, Database, Crypotography]
```

Problem 2

Code

```java
import java.util.HashMap;
import java.util.Scanner;

public class hash {
    public static void main(String[] args) {
        // setup hash map, with integer key and string value
        HashMap<Integer, String> players = new HashMap<Integer, String>();

        // add all values, random names
        players.put(1, "Bob");
        players.put(2, "Ben");
        players.put(3, "Alice");
        players.put(4, "Charlie");
        players.put(5, "David");
        players.put(6, "Eva");
        players.put(7, "Frank");
        players.put(8, "Grace");
        players.put(9, "Henry");
        players.put(10, "Ivy");

        // setup user input and read it
        Scanner sc = new Scanner(System.in);    // setup scanner
        System.out.print("Give player number: ");   // print "Give player number: " to the terminal
        int player_key = sc.nextInt();  // scan what integer the user inputted and store it
        sc.close(); // close scanner

        // print value based of the key the user gave
        System.out.println(players.get(player_key));

    }
}

```

Logic



Results

```
Give player number: 7
Frank
```

Problem 3

Code + Explanation

```java
public class linked_list {
    // init size and dummy nodes
    public int size = 0;    // keep track of how many elements in list
    private DLLNode start = new DLLNode();  // dummy start node
    private DLLNode end = new DLLNode();    // dummy end node

```

First, I initialise the start, end and size instance variables, to keep a pointer to the start and the end of the list.

```
1  public linked_list(){
2      // connect dummy nodes to each other
3      start.next = end;
4      end.prev = start;
5  }
```

In the constructor, I connect the start and end nodes together, using their internal pointers

```
32   public void display(){
33       // set current node as the start
34       DLLNode current = start.next;
35       // loop through each node
36       for (int i = 0; i < size; i++){
37           // check that this isn't the first element
38           if (i > 0){
39               // print node
40               System.out.print("->" + current.element);
41           } else {
42               // print now without -> prefix
43               System.out.print(current.element);
44           }
45           // set current to next node
46           current = current.next;
47       }
48       // go to next line
49       System.out.println();
50   }
51
```

The display function simply loops through all of the connected nodes, starting at the one in front of the start node, printing its contents, then updating the current node to the next node, and so on. This function is O(n) time complexity, where n is the length of the list.

```
52  public DLLNode append(int e){
53      // create new node for the element
54      DLLNode new_node = new DLLNode();
55      new_node.set_element(e);    // set element in node to element specified
56
57      // adjust pointers to add node to the list
58      DLLNode previous_node = end.prev;   // get last node in list
59      previous_node.next = new_node;  // set the next pointer in the last node to the new node
60      end.prev = new_node;    // set the prev pointer in the end dummy node to the new node
61      new_node.prev = previous_node;  // set new node prev pointer to last node
62      new_node.next = end;    // set new node next pointer to end dummy pointer
63
64      // add one to size
65      size++;
66
67      // return pointer to node
68      return new_node;
69   }
```

This “append” function adds an item to the end of the list. It first creates the node, with the specified element. It then gets the last node already in the list, and changes its pointers add slot in the newly created

node. It then adds one to the size counter, and returns the reference to the newly created node. This is done in O(1) time complexity

```
71  public int pop(){
72        // get last node in list
73        DLLNode node_to_remove = end.prev;
74
75        // adjust pointers
76        end.prev = node_to_remove.prev;
77        node_to_remove.prev.next = end;
78
79        // remove one from size
80        size--;
81
82        // return element
83        return node_to_remove.element;
84    }
```

The “pop” function removes the last element in the list. It first gets the last element in the list and stores it. It then updates the pointers on the last dummy node to point to the 2nd last node. We then update the “next” pointer on the 2nd last node to point to the end dummy node. We then remove one from the size variable and return the removed node object. This is done O(1) time complexity.

```
86  public DLLNode insert(int element, int index){
87      // create node from element
88      DLLNode new_node = new DLLNode();
89      new_node.set_element(element);  // set element attribute to element specified
90
91      // loop through list and find correct node to add in front of
92      DLLNode current_node = start;   // set start dummy node as "current_node"
93      for (int i = 0; i < index; i++){    // loop until reach the index
94          current_node = current_node.next;   // go to next node in list
95      }
96
97      // adjust pointers
98      DLLNode next_node = current_node.next;  // get node after current node
99      current_node.next = new_node;   // set current node's next pointer to the new node
100     new_node.prev = current_node;   // set new node prev pointer to current node
101     new_node.next = next_node;      // set new node next pointer to next_note
102     next_node.prev = new_node;      // set next_node prev pointer to new node
103
104     // add size
105     size++;
106     // return node object
107     return new_node;
108 }
```

The “insert” function is similar to the “append” function, however it’s able to specify which index to add the node to. The only drawback is that this is much less efficient then “append”, as it must first loop through up to n elements. Therefore, this is O(n) time complexity.

The function first creates the node with the specified element. It then loops through all the connected nodes until it reaches the correct index. It now has the correct reference the index node and can now adjust the pointers around it to slot in the newly created node. It then adds one to the size variable and returns the new element.

```
110     public int remove(int index){
111         // find correct node by iterating through each node
112         DLLNode current_node = start;   // set current node as start dummy node
113         for (int i = 0; i <= index; i++){   // loop until we reach the index specified
114             current_node = current_node.next;       // set current node to next node in the list
115         }
116 
117         // get previous and next nodes
118         DLLNode node_to_remove = current_node;  // this is the node at the index that we want removed
119         DLLNode previous_node = current_node.prev;  // node before it
120         DLLNode next_node = current_node.next;  // node after it
121 
122         // adjust pointers
123         previous_node.next = next_node; // set the previous node's next pointer to the node after the index
124         next_node.prev = previous_node; // set the next node's prev pointer to the node before the index
125 
126         // remove and return node
127         size--;
128         // return the element of the node
129         return node_to_remove.element;
130     }
131 
132     public int length(){
133         // return the instance attribute size
134         return this.size;
135     }
136 }
```

The “remove” function takes in an index number and then removes that element from the list and returns it. It first searches through the list for the specified index, which makes this an O(n) time complexity function. Once it’s found it, it adjusts the pointers, by setting the previous node’s “next” pointer to the one in front, and setting the next node’s “prev” pointer to the one in the back. It then removes one from the size variable and returns the element of the removed node.

```
138 class DLLNode {
139     // setup element, previous and next nodes
140     public int element;
141     public DLLNode prev = null;
142     public DLLNode next = null;
143 
144     public void set_element(int e){
145         // set the element variable to the input
146         this.element = e;
147     }
148 }
```

This is simply to show you what the “DLLNode” class looks like. It simply contains 3 instance variables. “Element” which stores the value, “prev” which is a pointer to the previous node, and “next” which is a pointer to the next node. It also contains a “set_element” function which sets the element.

Results

```java
public static void main(String[] args){
    linked_list test_list = new linked_list();
    test_list.append(11);
    test_list.append(22);
    test_list.append(6);
    test_list.append(89);
    test_list.append(99);
    test_list.display();
    test_list.insert(50, 2);
    test_list.display();
    test_list.remove(1);
    test_list.display();
    test_list.remove(0);
    test_list.display();
    test_list.pop();
    test_list.display();

}
```

When this “main” function above is ran, this is the output:

```
11->22->6->89->99
11->22->50->6->89->99
11->50->6->89->99
50->6->89->99
50->6->89
```

Problem 4

```java
import java.util.Scanner;
import java.util.HashMap;

public class strings {
    public static void main(String[] args) {

        // get 2 inputs
        Scanner sc = new Scanner(System.in);    //setup scanner object
        System.out.print("Line 1: ");   // print "Line 1: " to the terminal
        String string1 = sc.nextLine();   // read what user typed and store in "string1"

        System.out.print("Line 2: ");   // print "Line 2: " to terminal
        String string2 = sc.nextLine();   // read what user typed and store in "string2"

        sc.close(); //close scanner
```

At first, I setup the scanner and ask the user for two strings, which are stored as “string1” and “string2”. The scanner is then closed because my IDE was bugging me about it.

```
17  // add two strings
18  String result = string1 + ' ' + string2;    // concat the two strings
19  int length = 0; // set length to 0
20
21  // calculate length
22  for (int i = 0; i < result.length(); i++){  // loop through all the characters in the result string
23      if (result.charAt(i) != ' '){   // if the char is not a space char
24          length += 1;    // add one to length var
25      }
26  }
```

I then add the two strings together through string concatenation, and store the result in the “result” variable. I initialise an integer variable called “length”, which will store the length of the “result” string (excluding white-spaces). I then loop through and check if the current character is a whitespace character. If it’s not, I add one to the length counter.
The reason I don’t simply use the “length()” function on the string is because this would count in white-space characters, whereas my method does not.

```
28  // setup a hash map to get count of each char in the string
29  HashMap<Character, Integer> letterCount = new HashMap<Character, Integer>();    // setup hash map with keys as chars and values as integers
```

I initialise a HashMap, with a character as the type for the keys and Integers for the type of the values. This HashMap will be to count each character and how many times it appears in the “result” string. We will need to loop through the string.

```
30  for (int i = 0; i < result.length(); i++){  // iterate through the result string
31      char key = Character.toLowerCase(result.charAt(i)); // get that char and make it lower case and make "key" var
32      // exclude whitespace
33      if (key != ' '){   // check if key var is an space char
34          if (letterCount.containsKey(key)){  // check if value exists in hashmap
35              // increment if exists in hashmap
36              letterCount.merge(key, 1, Integer::sum);
37          } else { // doesn't exist in hashmap
38              // add to hashmap
39              letterCount.put(key, 1);
40          }
41      }
42  }
43
```

We loop through the “result” string and change each character to lowercase, as we aren’t differentiating between upper and lower case. We exclude whitespace by checking if the character is a space character, and adding it if it isn’t. I also check if the character is already in the HashMap. If it is, I use the “merge” function to increment the existing value.

```
44  // print result, length and reversed st ring
45   System.out.println(result); // print result
46   System.out.println("Length: " + length);    // print length of result excluding whitespace
47   System.out.println(reverse_words(result));  // use the reverse_words function to print the string in reverse order
```

I then print the “result”, the length of “result” and the reverse of the “result” string, using a function I made, at the end of the program:

```java
public static String reverse_words(String text){
    // split string on space, to split by words
    String[] temp = text.split(" ");    // split on space, store in list called temp
    String new_string = "";     // setup new string that will be returned

    // reverse iterate through list and add to string
    for (int i = temp.length-1; i >= 0; i--){   // reverse iterate, start at the last value and go down
        if (i != 0){    // if it's not the first element in the list
            new_string += (temp[i] + " ");  // add that word to the new string and add an extra space at the end
        } else {    // if it is the first element in "text" element
            new_string += temp[i]; // add word to new_string but no extra space
        }

    }
    // return the reversed string
    return new_string;
}
```

This function takes in a String “text” as input and uses regex to split the string on space, giving me a list of each word. I then iterate through that list of words, however in reverse order. I start at the last element of the list and iterate downwards. I add each word to a string. I also check if this word is the last element of the list, if it is, I simply don’t add an extra space at the end. I then return the reversed string.

```java
    // iterate through hashmap and print letters with an assigned value of 2
    for (Character key : letterCount.keySet()){ // iterate through the keys in the hashmap
        if (letterCount.get(key) == 2){ // if the value associated with the key is 2
            System.out.print(key);  // print the key
        }
    }

}
```

The final thing I do in this program, is iterate through the keys in the HashMap and check if the value associated with that key is two, and if it is, print it.

Results:

```
Line 1: Hello my name is Dylan
Line 2: I study Computer Science
Hello my name is Dylan I study Computer Science
Length: 39
Science Computer study I Dylan is name my Hello
adotu
```