MODULE- 4

REGULAR PROGRAMS – POISONOUS PLANT, TRUCK TOUR, QUEUE USING TWO STACKS

1. POISONOUS PLANT

```
#include <stdio.h>
#include <stdlib.h>
// Define the structure for a stack element
typedef struct {
   int pesticide;
   int days;
} Plant;
// Function to find the number of days until no plants die
int poisonousPlants(int n, int* p) {
   Plant* stack = (Plant*)malloc(n * sizeof(Plant));
   int top = -1, max_days = 0;
   for (int i = 0; i < n; i++) {
      int days = 0;
      while (top >= 0 && stack[top].pesticide >= p[i])
     {
         days = days > stack[top].days ? days : stack[top].days;
         top--;
      }
      if (top >= 0) {
         days++;
      } else {
      days = 0;
      }
      max_days = days > max_days ? days : max_days;
      stack[++top] = (Plant){p[i], days};
   }

   free(stack);
   return max_days;
}

int main() {
   int n;
   scanf("%d", &n);
   int* p = (int*)malloc(n * sizeof(int));
   for (int i = 0; i < n; i++) {
      scanf("%d", &p[i]);
   }
   int result = poisonousPlants(n, p);
   printf("%d\n", result);
   free(p);
   return 0;
}
```

2. Truck Tour

```
#include <stdio.h>

int main() {
   int n;
   scanf("%d", &n);

   int petrol[n], distance[n];

   for (int i = 0; i < n; i++) {
      scanf("%d %d", &petrol[i], &distance[i]);
   }

   int start = 0;
   int deficit = 0;
   int capacity = 0;

   for (int i = 0; i < n; i++) {
      capacity += petrol[i] - distance[i];
      if (capacity < 0) {
         start = i + 1;
         deficit += capacity;
         capacity = 0;
      }
   }

   if (capacity + deficit >= 0) {
      printf("%d\n", start);
   } else {
      printf("-1\n");
   }

   return 0;
```

3. **Queue using two stacks**

```
#include <stdio.h>
#include <stdlib.h>

// Stack structure
typedef struct Stack {
   int top;
   unsigned capacity;
   int* array;
} Stack;

// Create a stack of given capacity
Stack* createStack(unsigned capacity) {
   Stack* stack = (Stack*)malloc(sizeof(Stack));
   stack->capacity = capacity;
   stack->top = -1;
   stack->array = (int*)malloc(stack->capacity * sizeof(int));
   return stack;
}

// Check if the stack is full
int isFull(Stack* stack) {
   return stack->top == (int)(stack->capacity) - 1;
}

// Check if the stack is empty
int isEmpty(Stack* stack) {
   return stack->top == -1;
}

// Push item to stack
void push(Stack* stack, int item) {
   if (isFull(stack)) return;
   stack->array[++stack->top] = item;
}

// Pop item from stack
int pop(Stack* stack) {
   if (isEmpty(stack)) return -1;
   return stack->array[stack->top--];
}

// Queue structure using two stacks
typedef struct Queue {
```

```
    Stack* stack1;
    Stack* stack2;
} Queue;

// Create a queue
Queue* createQueue(unsigned capacity) {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->stack1 = createStack(capacity);
    queue->stack2 = createStack(capacity);
    return queue;
}

// Enqueue operation
void enqueue(Queue* queue, int item) {
    push(queue->stack1, item);
}

// Dequeue operation
int dequeue(Queue* queue) {
    if (isEmpty(queue->stack2)) {
        while (!isEmpty(queue->stack1)) {
            push(queue->stack2, pop(queue->stack1));
        }
    }
    return pop(queue->stack2);
}

// Display the queue
void displayQueue(Queue* queue) {
    if (isEmpty(queue->stack1) && isEmpty(queue->stack2)) {
        printf("Queue is empty\n");
        return;
    }

    printf("Queue: ");
    // Print stack2 (front part of queue)
    for (int i = 0; i <= queue->stack2->top; i++) {
        printf("%d ", queue->stack2->array[i]);
    }
    // Print stack1 in reverse (back part of queue)
    for (int i = queue->stack1->top; i >= 0; i--) {
        printf("%d ", queue->stack1->array[i]);
    }
    printf("\n");
}

// Main function to test
```

```
int main() {
    Queue* queue = createQueue(100);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);

    printf("Dequeued item is %d\n", dequeue(queue));

    displayQueue(queue);

    return 0;
}
```

Module4 (Additional)

1.Find the middle element of a stack

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100  // Define maximum size of stack

// Stack structure
typedef struct {
    int arr[MAX_SIZE];
    int top;
} Stack;

// Function to initialize the stack
void init(Stack* stack) {
    stack->top = -1;
}

// Push operation
void push(Stack* stack, int data) {
    if (stack->top == MAX_SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    stack->arr[++stack->top] = data;
}

// Pop operation
int pop(Stack* stack) {
    if (stack->top == -1) {
        printf("Stack underflow\n");
        return -1;
    }
    return stack->arr[stack->top--];
}

// Function to get the middle element
int getMiddle(Stack* stack) {
    if (stack->top == -1) {
        printf("Stack is empty\n");
        return -1;
    }
    return stack->arr[stack->top / 2];  // Middle index calculation
}
```

```
// Main function
int main() {
    Stack stack;
    init(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    push(&stack, 40);
    push(&stack, 50);

    printf("Middle Element: %d\n", getMiddle(&stack));

    pop(&stack);

    printf("Middle Element after pop: %d\n", getMiddle(&stack));

    return 0;
}
```

2.The celebrity Problem

```
#include <stdio.h>
#include <stdbool.h>
#define N 4  // Number of people
// Mock knows function (should be given or implemented based on input)
int MATRIX[N][N] = {
{0, 1, 1, 1},
{0, 0, 0, 1},
{0, 1, 0, 1},
{0, 0, 0, 0}  // Person 3 is the celebrity
};
// Function that returns whether A knows B
int knows(int a, int b) {
int i=0;
return MATRIX[a][b];
}
// Function to find the celebrity
int findCelebrity(int n) {
int candidate = 0;
// Step 1: Find the potential celebrity
int i=0;
for (i = 1; i < n; i++) {
if (knows(candidate, i)) {
candidate = i;  // Candidate cannot be a celebrity
}
}
// Step 2: Verify if the candidate is a real celebrity
for (i = 0; i < n; i++) {
if (i != candidate) {
if (knows(candidate, i) || !knows(i, candidate)) {
return -1;  // No celebrity exists
}
}
}
return candidate;  // Found a celebrity
}
// Main function
int main() {
int celebrity = findCelebrity(N);
if (celebrity == -1)
printf("No Celebrity found\n");
else
printf("Celebrity is Person %d\n", celebrity);
return 0;
}
```

Module1

1.strong Password

```
#include <stdio.h>
#include <string.h>
// Function to determine the minimum number of characters to
add
int minimumNumber(int n, char *password) {
   int required_chars = 0,i;
   int has_digit = 0, has_lower = 0, has_upper = 0, has_special = 0;
   const char *special_characters = "!@#$%^&*()-+";
   // Check the existing characters in the password
   for ( i = 0; i < n; i++) {
      if (password[i] >= '0' && password[i] <= '9') has_digit = 1;
      else if (password[i] >= 'a' && password[i] <= 'z') has_lower =1;
      else if (password[i] >= 'A' && password[i] <= 'Z') has_upper
=1;
      else if (strchr(special_characters, password[i])) has_special
=1;
   }
   // Count the missing types of characters
   if (!has_digit) required_chars++;
   if (!has_lower) required_chars++;
   if (!has_upper) required_chars++;
   if (!has_special) required_chars++;
   // Ensure the password length is at least 6 characters
   if (n + required_chars < 6) {
      required_chars += (6 - (n + required_chars));
   }
   return required_chars;
}
 int main() {
   int n;
   char password[101];
   // Input the length of the password and the password itself
   scanf("%d", &n);
   scanf("%s", password);
   // Calculate and print the minimum number of characters to
add
   int result = minimumNumber(n, password);
   printf("%d\n", result);
   return 0;
}
```

2.Running Time of algorithms

```
#include <stdio.h>
// Function to perform Insertion Sort and count the number of shifts
int runningTime(int arr[], int n) {
    int shifts = 0;
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        // Move elements of arr[0..i-1] that are greater than key
     //to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
            shifts++;
        }
        arr[j + 1] = key;
    }
    return shifts;
}
int main() {
    int n;
    // Read the number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements: "); // Read the array elements
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    int result = runningTime(arr, n); // Get the number of shifts and print it
    printf("Number of shifts: %d\n", result);
    return 0;
   }
```

3. Power Sum

```
#include <stdio.h>
#include <math.h>
// Function to recursively find the power sums
int powerSumHelper(int X, int N, int num) {
    int power = pow(num, N);
    if (power > X) {
        return 0;
    } else if (power == X) {
        return 1;
    } else {
        return powerSumHelper(X - power, N, num + 1) + powerSumHelper(X, N, num + 1);
    }
}
// Main function to find the number of ways to express X as sum of N-th powers of unique natural
numbers
int powerSum(int X, int N) {
    return powerSumHelper(X, N, 1);
}
int main() {
    int X, N;
    printf("Enter X: ");// Input the values of X and N
    scanf("%d", &X);
    printf("Enter N: ");
    scanf("%d", &N);
    int result = powerSum(X, N);     // Calculate and print the number of combinations
    printf("%d\n", result);
    return 0;
}
```

4.Water Connection Problem (Aditional)

```
#include <stdio.h>
#include <limits.h>
#define MAX 1000
int start[MAX], end[MAX], diameter[MAX];
int visited[MAX];
void initialize(int n) {
for (int i = 0; i <= n; i++) {
start[i] = -1;
end[i] = -1;
diameter[i] = INT_MAX;
visited[i] = 0;
}
}
void solve(int n) {
int count = 0;
int result[MAX][3];
for (int i = 1; i <= n; i++) {
if (end[i] == -1 && start[i] != -1) {
int curr = i;
int minDiameter = INT_MAX;
while (start[curr] != -1) {
minDiameter = (minDiameter < diameter[curr]) ? minDiameter :
diameter[curr];
curr = start[curr];
}
result[count][0] = i;
result[count][1] = curr;
result[count][2] = minDiameter;
count++;
}
}
printf("%d\n", count);
for (int i = 0; i < count; i++) {
printf("%d %d %d\n", result[i][0], result[i][1], result[i][2]);
}
}
int main() {
int n = 9, p = 6;
int a[] = {7, 5, 4, 2, 9, 3};
int b[] = {4, 9, 6, 8, 7, 1};
```

```
int d[] = {98, 72, 10, 22, 17, 66};
initialize(n);
for (int i = 0; i < p; i++) {
start[a[i]] = b[i];
diameter[a[i]] = d[i];
end[b[i]] = a[i];
}
printf("\nOutput:\n");
solve(n);
return 0;
}
```

5.Gold Mine Problem (additional)

```
#include <stdio.h>
#include <string.h>
#define MAX 100
// Function to get maximum of three integers
int max(int a, int b, int c) {
if (a > b && a > c) return a;
if (b > c) return b;
return c;
}
// Function to find the maximum gold collected
int getMaxGold(int gold[MAX][MAX], int n, int m) {
int dp[MAX][MAX],col,row;
memset(dp, 0, sizeof(dp));
// Fill the DP array starting from the last column
for ( col = m - 1; col >= 0; col--) {
for (row = 0; row < n; row++) {
// Possible moves
int right = (col == m - 1) ? 0 : dp[row][col + 1];
int right_up = (row == 0 || col == m - 1) ? 0 : dp[row - 1][col + 1];
int right_down = (row == n - 1 || col == m - 1) ? 0 : dp[row + 1][col + 1];
// DP state transition
dp[row][col] = gold[row][col] + max(right, right_up, right_down);
}
}
// Find the maximum collected gold in the first column
int maxGold = dp[0][0] , i;
for ( i = 1; i < n; i++) {
if (dp[i][0] > maxGold) {
maxGold = dp[i][0];
}
}
return maxGold;
}
// Driver Code
int main() {
int gold[MAX][MAX] = {
{1, 3, 1, 5},
{2, 2, 4, 1},
{5, 0, 2, 3},
{0, 6, 1, 2}
};
int n = 4, m = 4; // Grid size
printf("Maximum gold collected: %d\n", getMaxGold(gold, n, m));
return 0;
}
```

MODULE – 2

1. Sorting: Comparator

```
import java.util.*;
class Player {
   String name;
   int score;

   Player(String name, int score) {
      this.name = name;
      this.score = score;
   }
}
class Checker implements Comparator<Player> {
   // complete this method
      public int compare(Player p1, Player p2) {
      return p1.score != p2.score ? (p2.score - p1.score) : p1.name.compareTo(p2.name);
   }
}
public class Solution {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      int n = scan.nextInt();

      Player[] player = new Player[n];
      Checker checker = new Checker();

      for(int i = 0; i < n; i++){
         player[i] = new Player(scan.next(), scan.nextInt());
      }
      scan.close();

      Arrays.sort(player, checker);
      for(int i = 0; i < player.length; i++){
         System.out.printf("%s %s\n", player[i].name, player[i].score);
      }
   }
}
```

2. Pattern-syntax-checker

```
import java.util.Scanner;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;
public class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of patterns to check:");
        int testCases = Integer.parseInt(scanner.nextLine());

        while (testCases > 0) {
            String pattern = scanner.nextLine();
            try {
                Pattern.compile(pattern);
                System.out.println("Valid");
            } catch (PatternSyntaxException e) {
                System.out.println("Invalid");
            }
            testCases--;
        }
        scanner.close();
    }
}
```

3. Java SHA-256

```
 import java.io.*;
import java.util.*;
import java.security.*;
public class Solution {
   public static void main(String[] args) {
      /* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should
 be named Solution. */
      Scanner scanner = new Scanner(System.in);
      String key = scanner.next();
      try{
      MessageDigest md = MessageDigest.getInstance("SHA-256");
         md.update(key.getBytes());
            byte[] digest = md.digest();
            StringBuffer stringbuffer = new StringBuffer();
      for (byte b: digest)
      {  // needed to print it in hexadecimal format
         stringbuffer.append(String.format("%02x", b));
      }
            System.out.println(stringbuffer.toString());
            }
       catch (NoSuchAlgorithmException exception)
      {
         System.out.println(exception);
      }

   }
}
```

Module2 (additional)
Java Regex 2 - Duplicate Words

```
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class DuplicateWords {
public static void main(String[] args) {
String regex = "\\b(\\w+)(?:\\W+\\1\\b)+";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Scanner in = new Scanner(System.in);
int numSentences = Integer.parseInt(in.nextLine());
while (numSentences-- > 0) {
String input = in.nextLine();
Matcher m = p.matcher(input);
// Check for subsequences of input that match the compiled pattern
while (m.find()) {
input = input.replaceAll(m.group(), m.group(1));
}
// Prints the modified sentence.
System.out.println(input);
}
in.close();
}
}
```

2.Java Reflection - Attributes (additional)

```
import java.lang.reflect.*;
import java.util.*;
class Student {
private String name;
private String id;
private String email;
public String getName() { return name; }
public String getId() { return id; }
public String getEmail() { return email; }
public void setName(String name) { this.name = name; }
public void setId(String id) { this.id = id; }
public void setEmail(String email) { this.email = email; }
}
public class Solution {
public static void main(String[] args) {
Class student = Student.class;
Field[] fields = student.getDeclaredFields();
List<String> fieldNames = new ArrayList<>();
for (Field field : fields) {
f
 ieldNames.add(field.getName());
}
Collections.sort(fieldNames);
for (String name : fieldNames) {
System.out.println(name);
}
// Display methods
Method[] methods = student.getDeclaredMethods();
List<String> methodNames = new ArrayList<>();
for (Method method : methods) {
methodNames.add(method.getName());
}
Collections.sort(methodNames);
for (String name : methodNames) {
System.out.println(name);
}
}
}
```