Name : Joel Neharu Gavit

Roll.No.: 20121048

ASSIGNMENT NO: 1 (A)

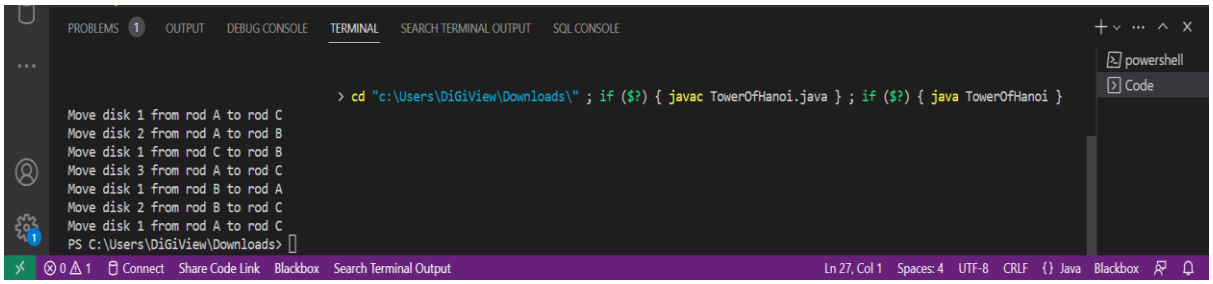
Algorithms and Problem Solving

a) Towers of Hanoi

Java Code:

```
public class TowerOfHanoi {
    public static void main(String[] args) {
        int numDisks = 3;
        char sourceRod = 'A';
        char auxiliaryRod = 'B';
        char destinationRod = 'C';

        towerOfHanoi(numDisks, sourceRod, auxiliaryRod, destinationRod);
    }

    public static void towerOfHanoi(int numDisks, char source, char auxiliary,
char destination) {
        if (numDisks == 1) {
            System.out.println("Move disk 1 from rod " + source + " to rod " +
destination);
            return;
        }

        // Move n-1 disks from source to auxiliary using destination as the
auxiliary rod
        towerOfHanoi(numDisks - 1, source, destination, auxiliary);

        // Move the nth disk from source to destination
        System.out.println("Move disk " + numDisks + " from rod " + source + "
to rod " + destination);

        // Move the n-1 disks from auxiliary to destination using source as
the auxiliary rod
        towerOfHanoi(numDisks - 1, auxiliary, source, destination);
    }
}
```

OUTPUT:


PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
SEARCH TERMINAL OUTPUT
SQL CONSOLE
powershell
Code
> cd "c:\Users\DiGiView\Downloads\" ; if ($?) { javac TowerOfHanoi.java } ; if ($?) { java TowerOfHanoi }
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
PS C:\Users\DiGiView\Downloads>
Connect
Share Code Link
Blackbox
Search Terminal Output
Ln 27, Col 1
Spaces: 4
UTF-8
CRLF
Java
Blackbox

Name : Joel Neharu Gavit

Roll.No.: 20121048

ASSIGNMENT NO: 1 (B)

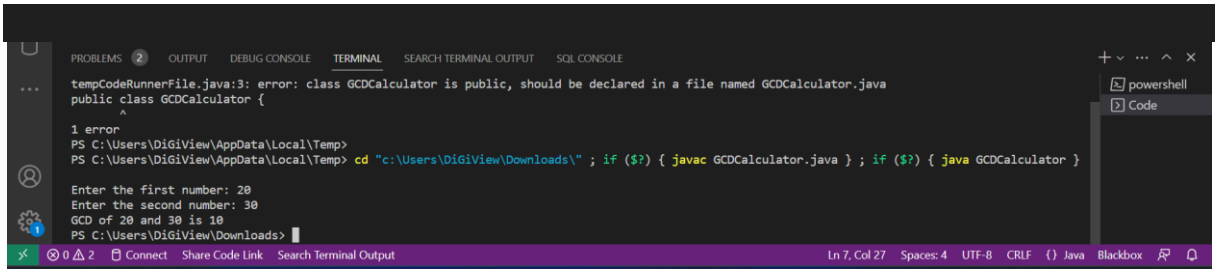
Algorithms and Problem Solving

b) GCD of Given Two Numbers:

Java Code:

```java
import java.util.Scanner;

public class GCDCalculator {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter the first number: ");
        int number1 = input.nextInt();

        System.out.print("Enter the second number: ");
        int number2 = input.nextInt();

        int gcd = findGCD(number1, number2);

        System.out.println("GCD of " + number1 + " and " + number2 + " is " +
gcd);
    }

    public static int findGCD(int a, int b) {
        // Ensure both numbers are positive
        a = Math.abs(a);
        b = Math.abs(b);

        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }

        return a;
    }
}
```

OUTPUT :


PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
SEARCH TERMINAL OUTPUT
SQL CONSOLE
tempCodeRunnerFile.java:3: error: class GCDCalculator is public, should be declared in a file named GCDCalculator.java
public class GCDCalculator {
^
1 error
PS C:\Users\DiGiView\AppData\Local\Temp>
PS C:\Users\DiGiView\AppData\Local\Temp> cd "c:\Users\DiGiView\Downloads\" ; if ($?) { javac GCDCalculator.java } ; if ($?) { java GCDCalculator }
Enter the first number: 20
Enter the second number: 30
GCD of 20 and 30 is 10
PS C:\Users\DiGiView\Downloads>
powershell
Code
Connect
Share Code Link
Search Terminal Output
Ln 7, Col 27
Spaces: 4
UTF-8
CRLF
Java
Blackbox

Name : Joel Neharu Gavit

Roll No : 20121048

ASSIGNMENT NO: 2

Analysis of Algorithms and Complexity Theory

c. Analysis of iterative and recursive algorithm

Java Code for calculating factorial using recursion:

```
import java.util.*;
public class Fact{
public static void main(String args[]){
Scanner sc = new Scanner(System.in);
int number = sc.nextInt();
int ans = fact(number);
System.out.println(ans);
}
static int fact(int n )
{
if(n<=1){
return 1;
}
else{
return n*fact(n-1);
}
}}
```

OUTPUT :


PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
SEARCH TERMINAL OUTPUT
SQL CONSOLE
powershell
Code
> cd "c:\Users\DiGiView\Downloads\" ; if ($?) { javac Fact.java } ; if ($?) { java Fact }
7
5040
PS C:\Users\DiGiView\Downloads>
0 2
Connect
Share Code Link
Blackbox
Search Terminal Output
Ln 18, Col 1
Spaces: 4
UTF-8
CRLF
Java
Blackbox

Name : Joel Neharu Gavit

Roll No : 20121048

ASSIGNMENT NO: 3

Greedy And Dynamic Programming algorithmic Striate

d. Job Scheduling using Greedy Algorithm

Java Code :

```
import java.util.Arrays;
import java.util.Comparator;
class Job {
char id;
int deadline;
int profit;
public Job(char id, int deadline, int profit) {
this.id = id;
this.deadline = deadline;
this.profit = profit;
}
}
public class JobScheduling {
public static void main(String[] args) {
Job[] jobs = {
new Job('A', 2, 100),
new Job('B', 1, 19),
new Job('C', 2, 27),
new Job('D', 1, 25),
new Job('E', 3, 15)
};
int n = jobs.length;
// Sort jobs in decreasing order of profit
Arrays.sort(jobs, Comparator.comparing((Job job) -> job.profit).reversed());
char[] result = new char[n];
boolean[] slot = new boolean[n];
for (int i = 0; i < n; i++) {
// Find a free slot for this job (from the end of the array to the start)
for (int j = Math.min(n, jobs[i].deadline) - 1; j >= 0; j--) {
if (!slot[j]) {
result[j] = jobs[i].id;
```

```
slot[j] = true;
break;
}
}
}
System.out.println("Job sequence for maximum profit:");
for (char jobId : result) {
System.out.print(jobId + " ");
}
}
}
```

OUTPUT:

Name : Joel Neharu Gavit

Roll No : 20121048

ASSIGNMENT NO: 4(E)

Backtracking and Branch-n-Bound

e) Knapsack problem,

Java Code:

```
public class KnapsackProblem {
    // Function to solve 0/1 Knapsack problem using dynamic programming
    public static int knapsack(int[] weights, int[] values, int totalWeight) {
    int n = weights.length;
    int[][] dp = new int[n + 1][totalWeight + 1];
    // Build dp table
    for (int i = 1; i <= n; i++) {
    for (int w = 1; w <= totalWeight; w++) {
    if (weights[i - 1] <= w) {
    dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i -
1][w]);
    } else {
    dp[i][w] = dp[i - 1][w];
    }
    }
    }
    // The maximum value that can be obtained
    return dp[n][totalWeight];
    }
    // Main method to test the knapsack function
    public static void main(String[] args) {
    int[] weights = {2, 3, 4, 5};
    int[] values = {3, 4, 5, 6};
    int totalWeight = 5;
    int maxValue = knapsack(weights, values, totalWeight);
    System.out.println("Maximum value that can be obtained: " + maxValue);
    }
    }
```

OUTPUT:


PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\USER> & 'C:\Program Files\Eclipse Adoptium\jdk-8.0.345.1-hotspot\bin\java.exe' '-cp' 'C:\Users\USER\AppData\Local\Temp\vscodesws_8636f\jdt_ws\jdt.ls-java-project\bin' 'KnapsackProblem'
Maximum value that can be obtained: 7
PS C:\Users\USER>

Name : Joel Neharu Gavit

Roll No : 20121048

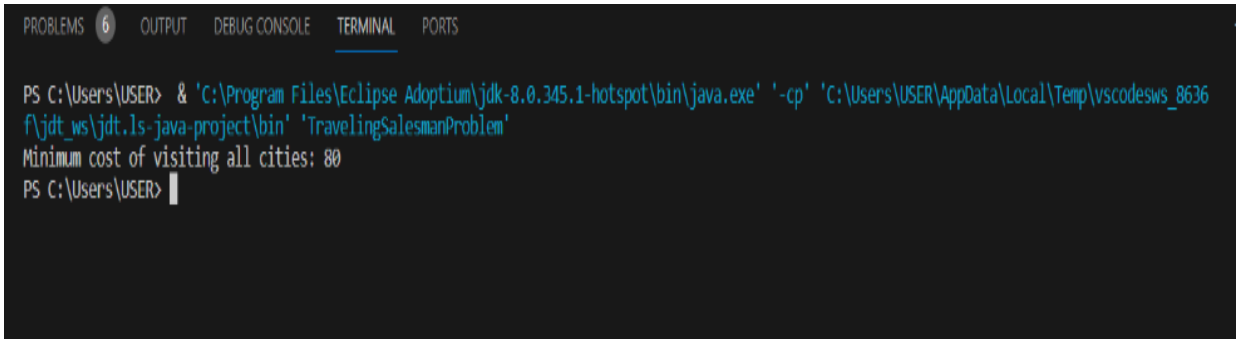
ASSIGNMENT NO: 4(F)

F) Travelling Salesman Problem Java Code:

```
import java.util.Arrays;
public class TravelingSalesmanProblem {
// Number of vertices in the graph
static int V = 4;
// Memoization table to store the results of subproblems
static int[][] dp;
// Adjacency matrix representing the graph
static int[][] graph = {
{0, 10, 15, 20},
{10, 0, 35, 25},
{15, 35, 0, 30},
{20, 25, 30, 0}
};
// Function to solve the Traveling Salesman Problem using dynamic programming
static int tsp(int mask, int pos) {
// If all cities have been visited, return the cost from the current city to
the starting city
if (mask == (1 << V) - 1) {
return graph[pos][0];
}
// If the subproblem has already been solved, return the stored result
if (dp[mask][pos] != -1) {
return dp[mask][pos];
}
// Initialize the result to a large value
int minCost = Integer.MAX_VALUE;
// Try to visit all cities
for (int city = 0; city < V; city++) {
// If the city is not visited yet and there is a direct edge from the current
city to this city
if ((mask & (1 << city)) == 0 && graph[pos][city] > 0) {
int newMask = mask | (1 << city);
int newCost = graph[pos][city] + tsp(newMask, city);
minCost = Math.min(minCost, newCost);
}
}
```

```
// Store the result of the subproblem in the memoization table
dp[mask][pos] = minCost;
return minCost;
}
public static void main(String[] args) {
// Initialize the memoization table with -1
dp = new int[1 << V][V];
for (int[] row : dp) {
Arrays.fill(row, -1);
}
// Start the TSP from city 0 and consider all other cities as unvisited (mask
= 1)
int mask = 1;
int minCost = tsp(mask, 0);
System.out.println("Minimum cost of visiting all cities: " + minCost);
}
}
```

OUTPUT:



Name : Joel Neharu Gavit

Roll No : 20121048

ASSIGNMENT NO: 5

Amortized Analysis

g) Sorting algorithm

1) Merge Sort :

Java Code :

```
public class MergeSort {
    public static void main(String[] args) {
    int[] arr = {12, 11, 13, 5, 6, 7};
    System.out.println("Unsorted array:");
    printArray(arr);
    mergeSort(arr, 0, arr.length - 1);
    System.out.println("\nSorted array:");
    printArray(arr);
    }
    public static void mergeSort(int[] arr, int left, int right) {
    if (left < right) {
    // Find the middle point of the array
    int mid = (left + right) / 2;
    // Recursively sort the first and second halves
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    // Merge the sorted halves
    merge(arr, left, mid, right);
    }
    }
    public static void merge(int[] arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int[] leftArray = new int[n1];
    int[] rightArray = new int[n2];
    // Copy data to temp arrays leftArray[] and rightArray[]
    for (int i = 0; i < n1; i++) {
    leftArray[i] = arr[left + i];
    }
    for (int j = 0; j < n2; j++) {
    rightArray[j] = arr[mid + 1 + j];
    }
```

```
// Merge the temp arrays
int i = 0, j = 0, k = left;
while (i < n1 && j < n2) {
if (leftArray[i] <= rightArray[j]) {
arr[k] = leftArray[i];
i++;
} else {
arr[k] = rightArray[j];
j++;
}
k++;
}
// Copy remaining elements of leftArray[] if any
while (i < n1) {
arr[k] = leftArray[i];
i++;
k++;
}
// Copy remaining elements of rightArray[] if any
while (j < n2) {
arr[k] = rightArray[j];
j++;
k++;
}
}
public static void printArray(int[] arr) {
for (int value : arr) {
System.out.print(value + " ");
}
System.out.println();
}
}
```

OUTPUT:

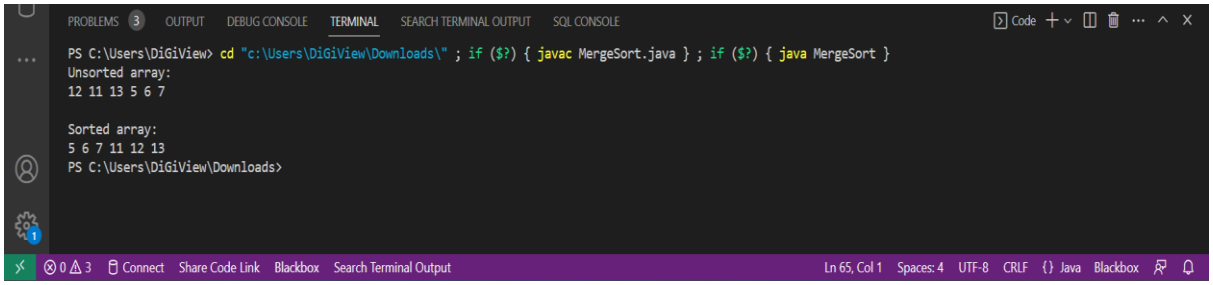
2) Quick Sort :

Java Code:

```
public class QuickSort {
    public static void main(String[] args) {
    int[] arr = {12, 11, 13, 5, 6, 7};
    System.out.println("Unsorted array:");
    printArray(arr);
    quickSort(arr, 0, arr.length - 1);
    System.out.println("\nSorted array:");
    printArray(arr);
    }
    public static void quickSort(int[] arr, int low, int high) {
    if (low < high) {
    // Partition the array into two sub-arrays
    int pivotIndex = partition(arr, low, high);
    // Recursively sort the sub-arrays
    quickSort(arr, low, pivotIndex - 1);
    quickSort(arr, pivotIndex + 1, high);
    }
    }
    public static int partition(int[] arr, int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
    if (arr[j] < pivot) {
    i++;
    swap(arr, i, j);
    }
    }
    swap(arr, i + 1, high);
    return i + 1;
    }
    public static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    }
    public static void printArray(int[] arr) {
    for (int value : arr) {
    System.out.print(value + " ");
    }
    System.out.println();
    }
```

```
    }
```

OUTPUT:


PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
SEARCH TERMINAL OUTPUT
SQL CONSOLE
Code
PS C:\Users\DiGiView> cd "c:\Users\DiGiView\Downloads\" ; if ($?) { javac MergeSort.java } ; if ($?) { java MergeSort }
Unsorted array:
12 11 13 5 6 7
Sorted array:
5 6 7 11 12 13
PS C:\Users\DiGiView\Downloads>
0 3
Connect
Share Code Link
Blackbox
Search Terminal Output
Ln 65, Col 1
Spaces: 4
UTF-8
CRLF
{} Java
Blackbox

Name : Joel Neharu Gavit

Roll No : 20121048

ASSIGNMENT NO: 6

Multithreaded and Distributed Algorithms

Multiplication of Matrix using threads

Java Code:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.Callable;
public class MatrixMultiplication {
public static void main(String[] args) {
int[][] matrixA = {{1, 2, 3}, {4, 5, 6}};
int[][] matrixB = {{7, 8}, {9, 10}, {11, 12}};
int numRowsA = matrixA.length;
int numColsA = matrixA[0].length;
int numRowsB = matrixB.length;
int numColsB = matrixB[0].length;
if (numColsA != numRowsB) {
System.out.println("Matrix multiplication is not possible.");
return;
}
int[][] result = new int[numRowsA][numColsB];
// Create a thread pool with a fixed number of threads (e.g., 4)
int numThreads = 4;
ExecutorService executor = Executors.newFixedThreadPool(numThreads);
// Perform matrix multiplication using threads
for (int i = 0; i < numRowsA; i++) {
for (int j = 0; j < numColsB; j++) {
Callable<Integer> task = new MatrixMultiplicationTask(matrixA, matrixB,
result, i, j);
Future<Integer> future = executor.submit(task);
}
}
// Shutdown the executor
executor.shutdown();
// Print the result
for (int i = 0; i < numRowsA; i++) {
for (int j = 0; j < numColsB; j++) {
System.out.print(result[i][j] + " ");
}
```

```
System.out.println();
}
}
}
class MatrixMultiplicationTask implements Callable<Integer> {
private int[][] matrixA;
private int[][] matrixB;
private int[][] result;
private int row;
private int col;
public MatrixMultiplicationTask(int[][] matrixA, int[][] matrixB, int[][]
result, int row, int col)
{
this.matrixA = matrixA;
this.matrixB = matrixB;
this.result = result;
this.row = row;
this.col = col;
}
@Override
public Integer call() {
int sum = 0;
for (int i = 0; i < matrixA[0].length; i++) {
sum += matrixA[row][i] * matrixB[i][col];
}
result[row][col] = sum;
return sum;
}
}
```

OUTPUT: