

Name : Tanmay Deokar
Class : BE Comp SS

Assignment A1

Code :

```
#include<iostream>
#include<vector>

using namespace std;

//Iteratively using memoization
int iStepFibonacci(int n){
    vector<int> f;
    f.push_back(0);
    f.push_back(1);
    //[]
    int cnt = 2;
    for(int i = 2; i < n; i++){
        cnt++;
        f.push_back(f[i - 1] + f[i - 2]);
    }
    return n;
}

int rSteps = 0;

//Recursively
int rStepFibonacci(int n){
    rSteps++;
    if(n < 0) return 0;
    if(n == 1 || n == 0) return 1;
    return rStepFibonacci(n - 1) + rStepFibonacci(n - 2);
}

int main(){
    int n;
    cin >> n;
    cout << "Fibonacci Value : " << rStepFibonacci(n) << '\n';
    cout << "Steps required using Iteration : " << iStepFibonacci(n) << '\n';
}
```

```

        cout << "Steps required using recursion : " << rSteps << '\n';
        return 0;
    }

```

Output :

```

1 #include<iostream>
2 #include<vector>
3
4 using namespace std;
5
6 //Iteratively using memoization
7 int iStepFibonacci(int n){
8     vector<int> f;
9     f.push_back(0);
10    f.push_back(1);
11    //[]
12    int cnt = 2;
13    for(int i = 2; i < n; i++)
14        cnt++;
15    f.push_back(f[i - 1] + f[i - 2]);
16    return f[n - 1];
17 }
18
19 int rSteps = 0;
20 //Recursively
21 int rStepFibonacci(int n){
22     rSteps++;
23     if(n < 0) return 0;
24     if(n == 1 || n == 0) return 1;
25     return rStepFibonacci(n - 1) + rStepFibonacci(n - 2);
26 }
27
28 int main(){
29     int n;
30 }

```

```

admin1@401-S: ~
File Edit View Search Terminal Help
bash: /u01/app/oracle/product/11.2.0/xe/bin/nls_lang.sh: No such file or directory
(base) admin1@401-S:~$ g++ a1.cpp
(base) admin1@401-S:~$ ./a.out
5
Fibonacci Value : 5
Steps required using Iteration : 5
Steps required using recursion : 15
(base) admin1@401-S:~$

```

A2:

```

import java.util.PriorityQueue;
import java.util.HashMap;

```

```

class HuffmanNode implements Comparable<HuffmanNode> {
    int frequency;
    char data;
    HuffmanNode left, right;

    public HuffmanNode(int frequency, char data) {
        this.frequency = frequency;
    }
}

```

```

        this.data = data;
        this.left = null;
        this.right = null;
    }

    @Override
    public int compareTo(HuffmanNode node) {
        return this.frequency - node.frequency;
    }
}

public class HuffmanEncoding {

    static HashMap<Character, String> huffmanCodes = new HashMap<>();

    public static void generateCodes(HuffmanNode root, String code) {
        if (root == null) return;

        if (root.left == null && root.right == null) {
            huffmanCodes.put(root.data, code);
        }

        generateCodes(root.left, code + "0");
        generateCodes(root.right, code + "1");
    }

    public static void buildHuffmanTree(String input) {
        HashMap<Character, Integer> frequencyMap = new HashMap<>();

        for (char c : input.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
        }

        PriorityQueue<HuffmanNode> priorityQueue = new PriorityQueue<>();

        for (char key : frequencyMap.keySet()) {
            priorityQueue.add(new HuffmanNode(frequencyMap.get(key), key));
        }

        while (priorityQueue.size() > 1) {
            HuffmanNode left = priorityQueue.poll();
            HuffmanNode right = priorityQueue.poll();

            int sum = left.frequency + right.frequency;

```

```

        HuffmanNode newNode = new HuffmanNode(sum, '-');
        newNode.left = left;
        newNode.right = right;

        priorityQueue.add(newNode);
    }

    HuffmanNode root = priorityQueue.poll();
    generateCodes(root, "");
}

public static String encode(String input) {
    StringBuilder encodedString = new StringBuilder();

    for (char c : input.toCharArray()) {
        encodedString.append(huffmanCodes.get(c));
    }

    return encodedString.toString();
}

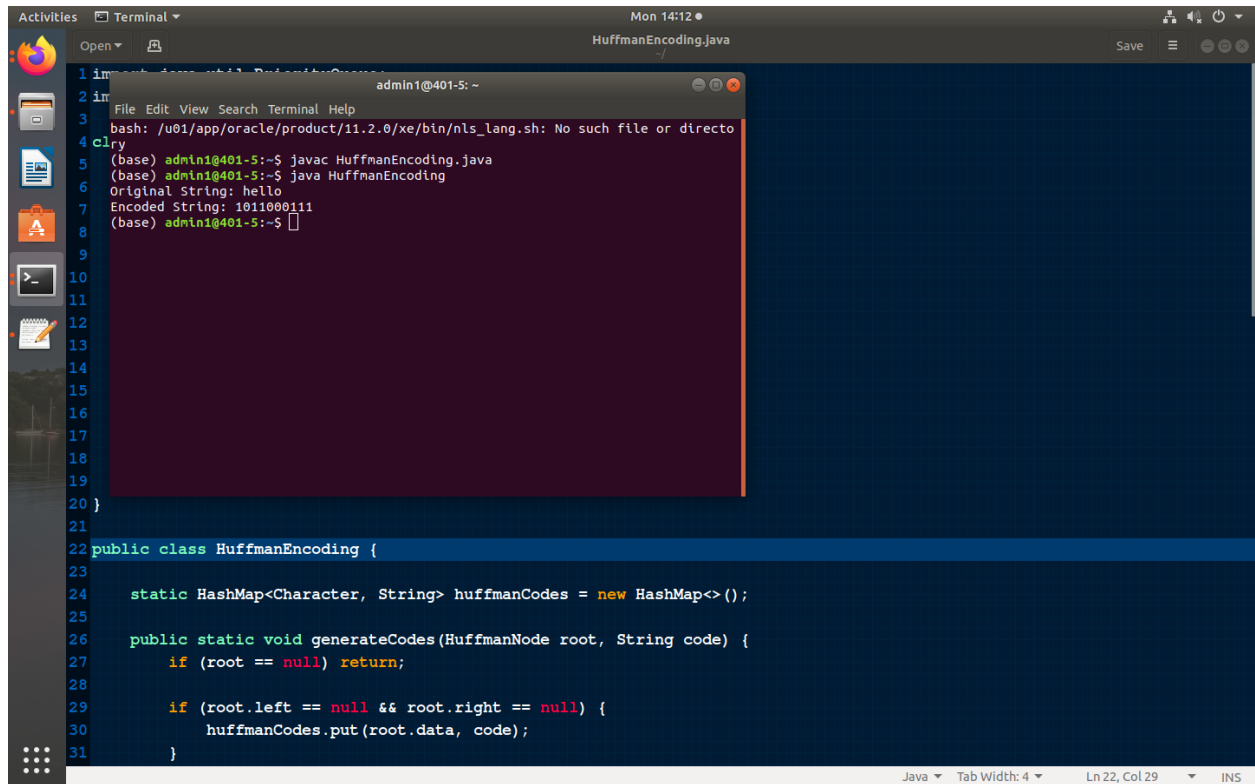
public static void main(String[] args) {
    String input = "hello";
    buildHuffmanTree(input);

    String encodedString = encode(input);

    System.out.println("Original String: " + input);
    System.out.println("Encoded String: " + encodedString);
}
}
...

```

Output :



```
1 import java.util.Scanner;
2
3 File Edit View Search Terminal Help
4 bash: /u01/app/oracle/product/11.2.0/xe/bin/nls_lang.sh: No such file or directo
5
6 (base) admin1@401-S:~$ javac HuffmanEncoding.java
7 (base) admin1@401-S:~$ java HuffmanEncoding
8 Original String: hello
9 Encoded String: 1011000111
10 (base) admin1@401-S:~$
11
12
13
14
15
16
17
18
19
20 }
21
22 public class HuffmanEncoding {
23
24     static HashMap<Character, String> huffmanCodes = new HashMap<>();
25
26     public static void generateCodes(HuffmanNode root, String code) {
27         if (root == null) return;
28
29         if (root.left == null && root.right == null) {
30             huffmanCodes.put(root.data, code);
31         }
32     }
33 }
```

A3 :

Code :

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

```
class Item {
    int weight;
    int value;

    public Item(int weight, int value) {
        this.weight = weight;
        this.value = value;
    }
}
```

```
class FractionalKnapsack {
    public static double getMaxValue(int[] weights, int[] values, int capacity) {
        int n = weights.length;
        Item[] items = new Item[n];

        for (int i = 0; i < n; i++) {
```

```

        items[i] = new Item(weights[i], values[i]);
    }

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if ((double) items[i].value / items[i].weight < (double) items[j].value /
items[j].weight) {
                Item temp = items[i];
                items[i] = items[j];
                items[j] = temp;
            }
        }
    }

    double totalValue = 0;
    int currentWeight = 0;

    for (Item item : items) {
        if (currentWeight + item.weight <= capacity) {
            currentWeight += item.weight;
            totalValue += item.value;
        } else {
            int remainingCapacity = capacity - currentWeight;
            totalValue += (double) item.value * remainingCapacity / item.weight;
            break;
        }
    }

    return totalValue;
}

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

    System.out.print("Enter the number of items: ");
    int n = scanner.nextInt();

    int[] weights = new int[n];
    int[] values = new int[n];

    System.out.println("Enter weights and values for each item:");
    for (int i = 0; i < n; i++) {
        weights[i] = scanner.nextInt();
        values[i] = scanner.nextInt();
    }
}

```

```

    }

    System.out.print("Enter the capacity of the knapsack: ");
    int capacity = scanner.nextInt();

    double maxVal = getMaxValue(weights, values, capacity);

    System.out.println("Maximum value in Knapsack = " + maxVal);
}
}

```

The screenshot shows a Java IDE with a file named `FractionalKnapsack.java` open. The code in the editor includes a bubble sort algorithm and the start of a knapsack function. The terminal window shows the execution of the program, where the user enters 3 items with weights [10, 20, 30] and values [20, 30, 40], and a knapsack capacity of 50. The program outputs a maximum value of 76.66666666666667.

```

1 import java.util.Scanner;
2
3 public class FractionalKnapsack {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         System.out.print("Enter the number of items: ");
7         int n = scanner.nextInt();
8         System.out.print("Enter weights and values for each item: ");
9         int[] weights = new int[n];
10        int[] values = new int[n];
11        for (int i = 0; i < n; i++) {
12            weights[i] = scanner.nextInt();
13            values[i] = scanner.nextInt();
14        }
15        System.out.print("Enter the capacity of the knapsack: ");
16        int capacity = scanner.nextInt();
17        double maxVal = getMaxValue(weights, values, capacity);
18        System.out.println("Maximum value in Knapsack = " + maxVal);
19    }
20
21    // Bubble Sort
22    public static void bubbleSort(int[] arr) {
23        for (int i = 0; i < arr.length - 1; i++) {
24            for (int j = i + 1; j < arr.length; j++) {
25                if ((double) arr[i].value / arr[i].weight < (double) arr[j].value / arr[j].weight) {
26                    Item temp = arr[i];
27                    arr[i] = arr[j];
28                    arr[j] = temp;
29                }
30            }
31        }
32    }
33
34    public static double getMaxValue(int[] weights, int[] values, int capacity) {
35        // Bubble Sort
36        bubbleSort(values);
37        // Create a dp array
38        int[][] dp = new int[n + 1][capacity + 1];
39        // Fill the dp array
40        for (int i = 1; i <= n; i++) {
41            for (int j = 1; j <= capacity; j++) {
42                dp[i][j] = Math.max(dp[i - 1][j], values[i - 1] * (double) dp[i - 1][j] / weights[i - 1]);
43            }
44        }
45        return dp[n][capacity];
46    }
47 }

```

A4 :

Code :

```

import java.util.Scanner;

class Knapsack {

    public static int knapSack(int capacity, int weights[], int values[], int n) {
        int[][] dp = new int[n + 1][capacity + 1];
    }
}

```

```

for (int i = 0; i <= n; i++) {
for (int w = 0; w <= capacity; w++) {
    if (i == 0 || w == 0)
        dp[i][w] = 0;
    else 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];
}
}

return dp[n][capacity];
}

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

System.out.print("Enter the number of items: ");
int n = scanner.nextInt();

int values[] = new int[n];
int weights[] = new int[n];

System.out.println("Enter values for each item:");
for (int i = 0; i < n; i++) {
values[i] = scanner.nextInt();
}

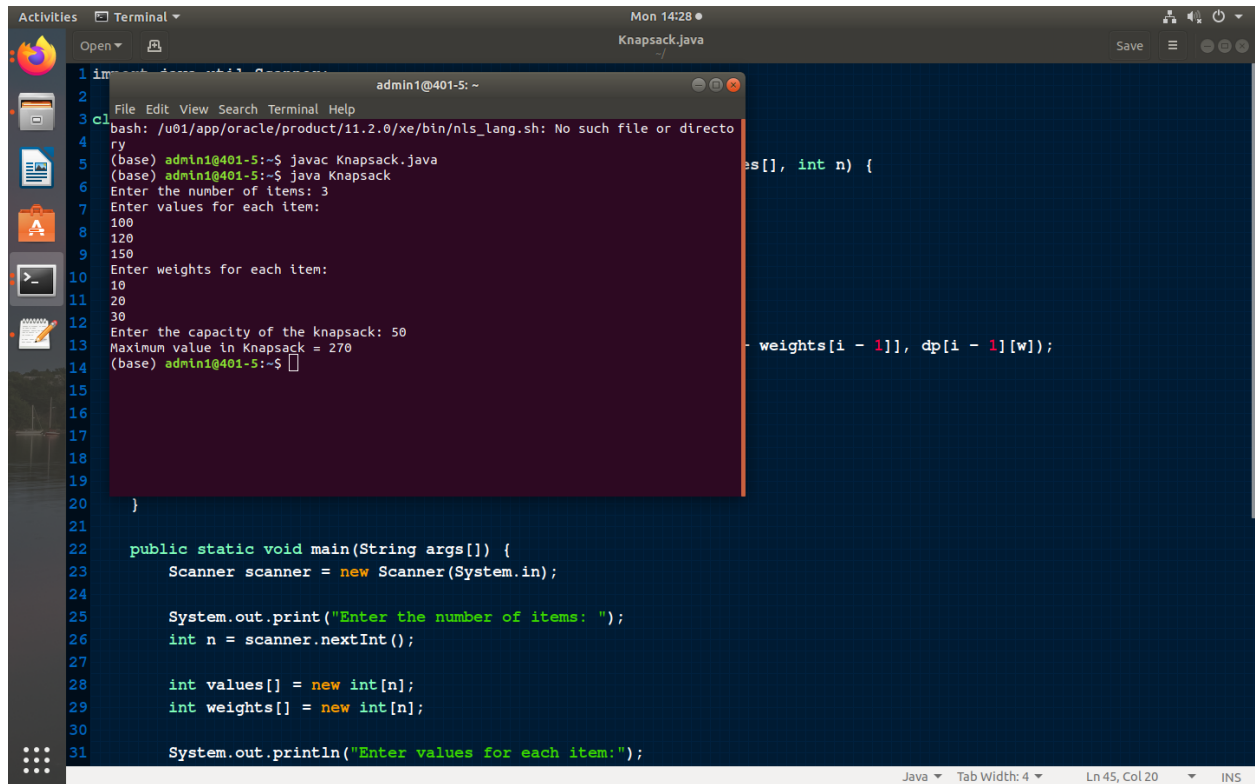
System.out.println("Enter weights for each item:");
for (int i = 0; i < n; i++) {
weights[i] = scanner.nextInt();
}

System.out.print("Enter the capacity of the knapsack: ");
int capacity = scanner.nextInt();

int maxValue = knapSack(capacity, weights, values, n);
System.out.println("Maximum value in Knapsack = " + maxValue);
}
}

```

Output :



```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11 11
12 12
13 13
14 14
15 15
16 16
17 17
18 18
19 19
20 20
21 21
22 22
23 23
24 24
25 25
26 26
27 27
28 28
29 29
30 30
31 31

admin1@401-S: ~
File Edit View Search Terminal Help
bash: /u01/app/oracle/product/11.2.0/xe/bin/nls_lang.sh: No such file or directory
(base) admin1@401-S:~$ javac Knapsack.java
(base) admin1@401-S:~$ java Knapsack
Enter the number of items: 3
Enter values for each item:
100
120
150
Enter weights for each item:
10
20
30
Enter the capacity of the knapsack: 50
Maximum value in Knapsack = 270
(base) admin1@401-S:~$

}

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

    System.out.print("Enter the number of items: ");
    int n = scanner.nextInt();

    int values[] = new int[n];
    int weights[] = new int[n];

    System.out.println("Enter values for each item:");
```

A5 :

```
public class NQueens {
    static int[][] board;

    public static boolean isSafe(int row, int col, int N) {
        int i, j;

        // Check the left side of the current row
        for (i = 0; i < col; i++) {
            if (board[row][i] == 1) {
                return false;
            }
        }

        // Check upper diagonal on left side
        for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }
    }
}
```

```

// Check lower diagonal on left side
for (i = row, j = col; j >= 0 && i < N; i++, j--) {
    if (board[i][j] == 1) {
        return false;
    }
}

return true;
}

public static boolean solveNQueensUtil(int N, int col) {
    if (col >= N) {
        return true;
    }

    for (int i = 0; i < N; i++) {
        if (isSafe(i, col, N)) {
            board[i][col] = 1;

            if (solveNQueensUtil(N, col + 1)) {
                return true;
            }

            board[i][col] = 0; // Backtrack
        }
    }

    return false;
}

public static void solveNQueens(int N) {
    board = new int[N][N];

    // Place the first queen
    board[0][0] = 1;

    if (!solveNQueensUtil(N, 1)) {
        System.out.println("Solution does not exist");
        return;
    }

    // Print the final board
    System.out.println("Final N-Queens Board:");
    printBoard(N);
}

```

```

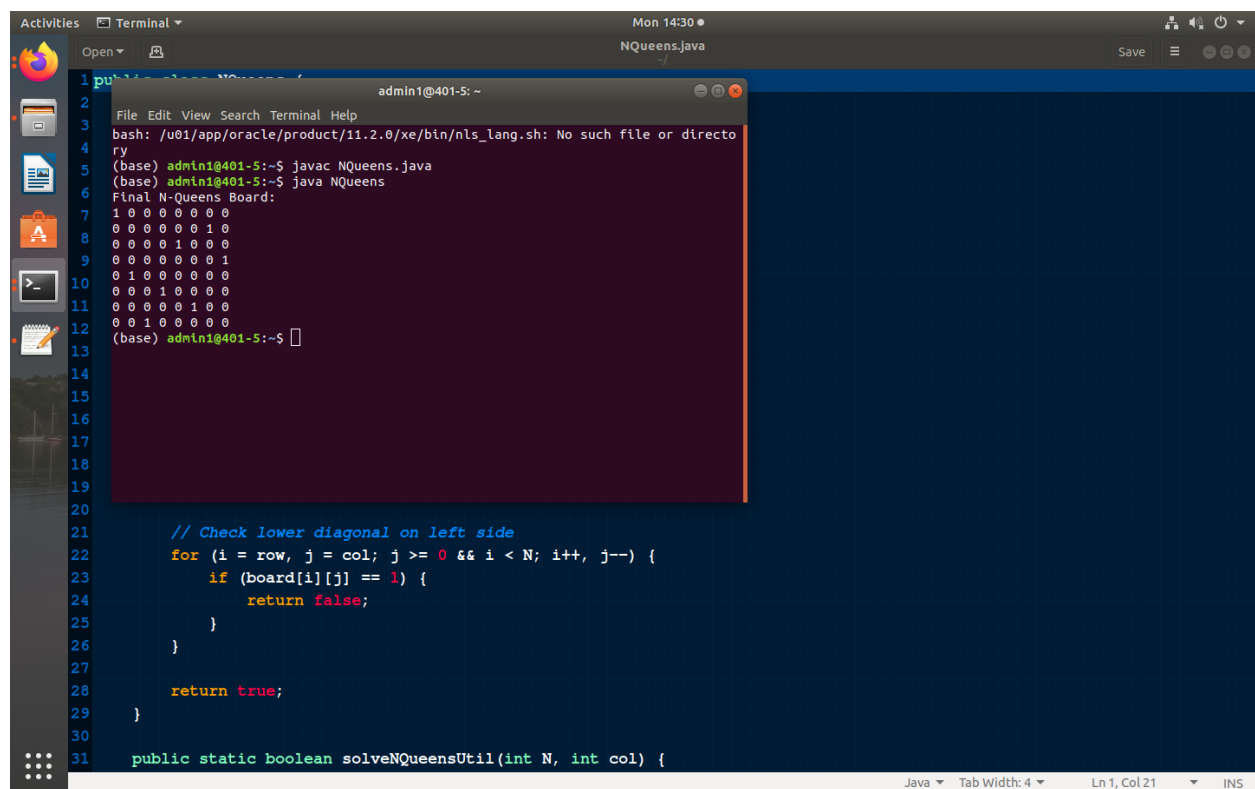
    }

    public static void printBoard(int N) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int N = 8; // Change N for different board sizes
        solveNQueens(N);
    }
}

```

Output :



```

1 public class NQueens {
2
3
4
5
6 Final N-Queens Board:
7 1 0 0 0 0 0 0 0
8 0 0 0 0 0 0 1 0
9 0 0 0 0 1 0 0 0
10 0 0 0 0 0 0 0 1
11 0 1 0 0 0 0 0 0
12 0 0 0 1 0 0 0 0
13 0 0 0 0 0 1 0 0
14 0 0 1 0 0 0 0 0
15
16
17
18
19
20
21 // Check lower diagonal on left side
22 for (i = row, j = col; j >= 0 && i < N; i++, j--) {
23     if (board[i][j] == 1) {
24         return false;
25     }
26 }
27
28 return true;
29 }
30
31 public static boolean solveNQueensUtil(int N, int col) {

```

MINI PROJECT :

Mini Project - Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.

Merge Sort :

```
public class MergeSort {

    public static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        int[] leftArr = new int[n1];
        int[] rightArr = new int[n2];

        for (int i = 0; i < n1; ++i)
            leftArr[i] = arr[left + i];
        for (int j = 0; j < n2; ++j)
            rightArr[j] = arr[mid + 1 + j];

        int i = 0, j = 0;

        int k = left;
        while (i < n1 && j < n2) {
            if (leftArr[i] <= rightArr[j]) {
                arr[k] = leftArr[i];
                i++;
            } else {
                arr[k] = rightArr[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = leftArr[i];
            i++;
            k++;
        }

        while (j < n2) {
```

```

arr[k] = rightArr[j];
j++;
k++;
}
}

public static void mergeSort(int[] arr, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

public static void printArray(int[] arr) {
    for (int i : arr)
        System.out.print(i + " ");
    System.out.println();
}

public static void main(String[] args) {
    int[] arr = { 38, 27, 43, 3, 9, 82, 10 };

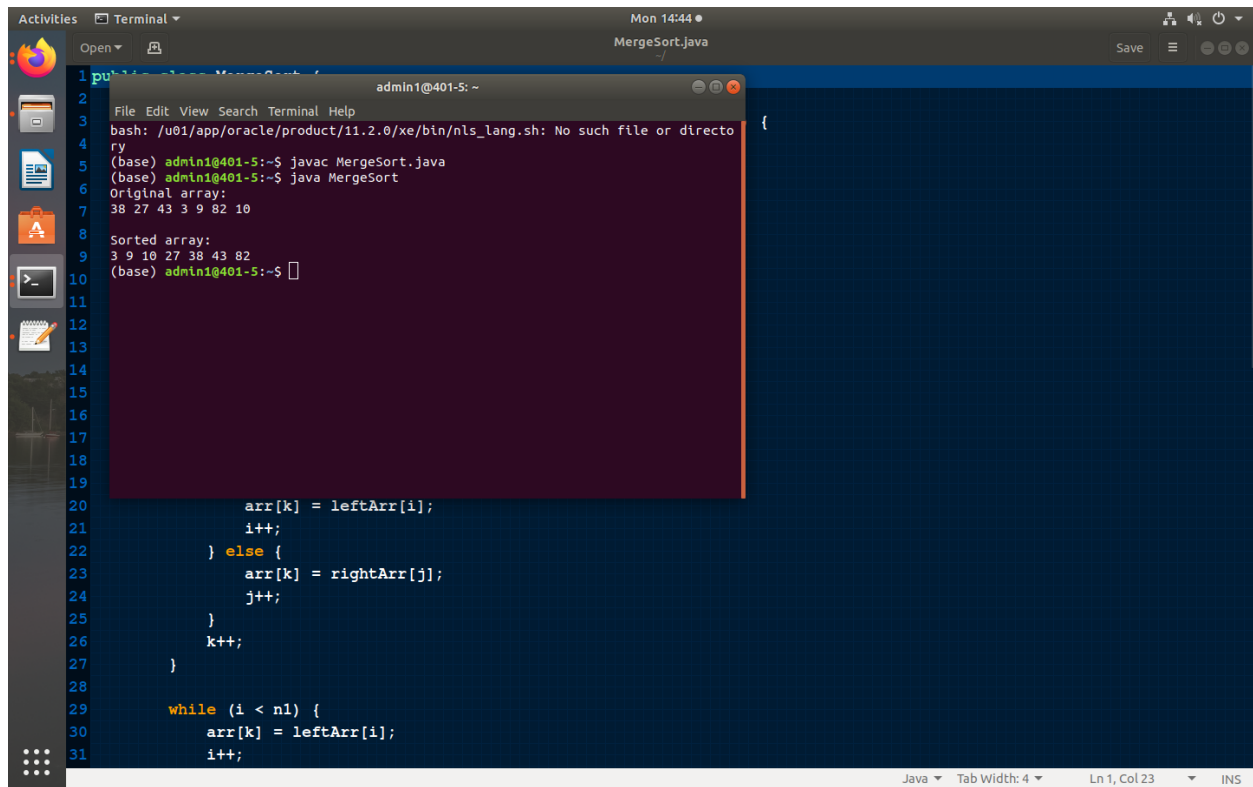
    System.out.println("Original array:");
    printArray(arr);

    mergeSort(arr, 0, arr.length - 1);

    System.out.println("\nSorted array:");
    printArray(arr);
}
}

```

Output :



```
1 public class MergeSort {
2
3     File Edit View Search Terminal Help
4     bash: /u01/app/oracle/product/11.2.0/xe/bin/nls_lang.sh: No such file or directory
5     (base) admin1@401-5:~$ javac MergeSort.java
6     (base) admin1@401-5:~$ java MergeSort
7     Original array:
8     38 27 43 3 9 82 10
9
10    Sorted array:
11    3 9 10 27 38 43 82
12    (base) admin1@401-5:~$
13
14
15
16
17
18
19
20    arr[k] = leftArr[i];
21    i++;
22    } else {
23        arr[k] = rightArr[j];
24        j++;
25    }
26    k++;
27    }
28
29    while (i < n1) {
30        arr[k] = leftArr[i];
31        i++;
```

Multithreaded merge sort :

Code :

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class MultithreadedMergeSort {

    static class MergeSortTask extends RecursiveAction {
        private final int[] array;
        private final int left;
        private final int right;

        public MergeSortTask(int[] array, int left, int right) {
            this.array = array;
            this.left = left;
            this.right = right;
        }
    }
}
```

```

@Override
protected void compute() {
    if (left < right) {
        int mid = left + (right - left) / 2;

        MergeSortTask leftTask = new MergeSortTask(array, left, mid);
        MergeSortTask rightTask = new MergeSortTask(array, mid + 1, right);

        invokeAll(leftTask, rightTask);

        merge(array, 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[] leftArr = new int[n1];
    int[] rightArr = new int[n2];

    for (int i = 0; i < n1; ++i)
        leftArr[i] = arr[left + i];
    for (int j = 0; j < n2; ++j)
        rightArr[j] = arr[mid + 1 + j];

    int i = 0, j = 0;

    int k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
    }
}

```

```

        k++;
    }

    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

public static void main(String[] args) {
    int[] arr = {38, 27, 43, 3, 9, 82, 10};

    System.out.println("Original array:");
    printArray(arr);

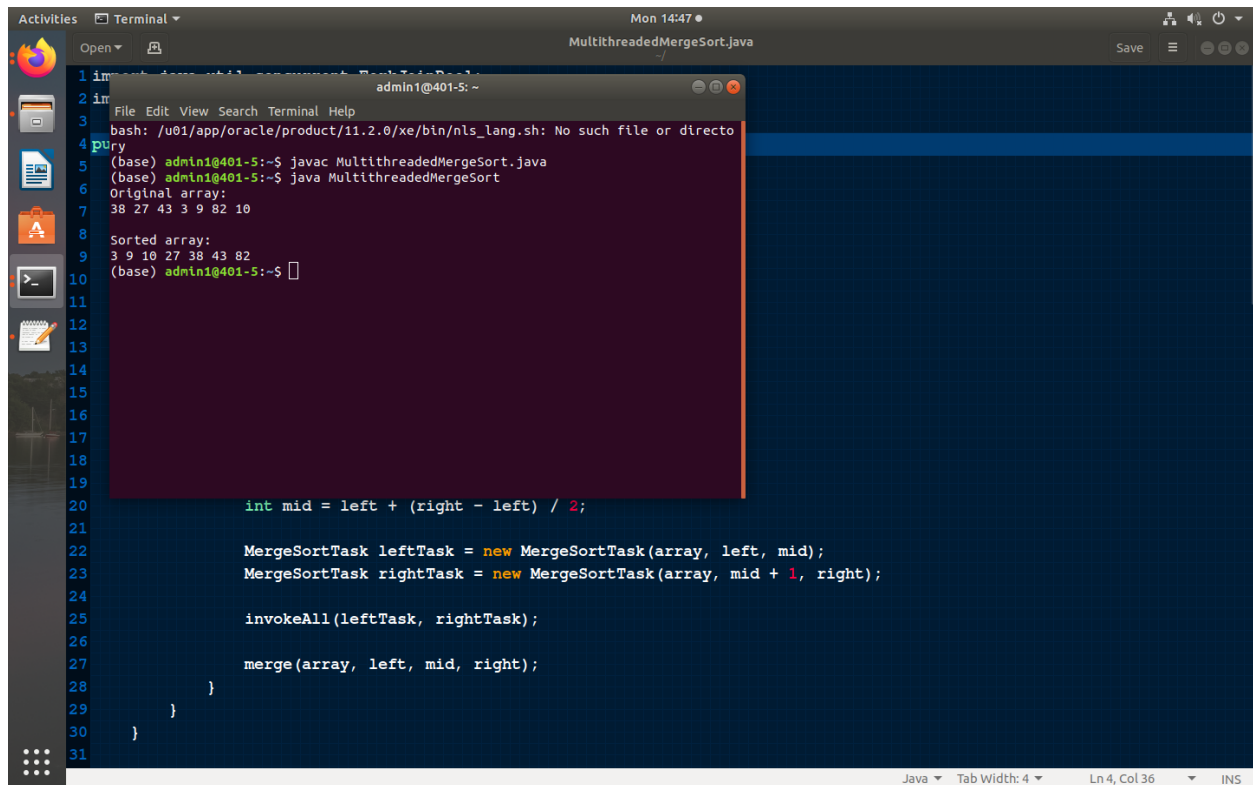
    ForkJoinPool pool = new ForkJoinPool();
    MergeSortTask task = new MergeSortTask(arr, 0, arr.length - 1);
    pool.invoke(task);

    System.out.println("\nSorted array:");
    printArray(arr);
}

public static void printArray(int[] arr) {
    for (int i : arr)
        System.out.print(i + " ");
    System.out.println();
}
}

```


Output :



```
1 import java.util.concurrent.*;
2 import java.util.*;
3
4 public class MultithreadedMergeSort {
5     public static void main(String[] args) {
6         int[] array = {38, 27, 43, 3, 9, 82, 10};
7         System.out.println("Original array:");
8         for (int i : array) {
9             System.out.print(i + " ");
10        }
11        System.out.println();
12
13        // Sorting the array
14        MergeSortTask task = new MergeSortTask(array, 0, array.length - 1);
15        task.execute();
16
17        System.out.println("Sorted array:");
18        for (int i : array) {
19            System.out.print(i + " ");
20        }
21        System.out.println();
22    }
23
24    // Merge Sort Task
25    static class MergeSortTask implements Runnable {
26        int[] array;
27        int left;
28        int right;
29
30        MergeSortTask(int[] array, int left, int right) {
31            this.array = array;
32            this.left = left;
33            this.right = right;
34        }
35
36        @Override
37        public void run() {
38            if (left < right) {
39                int mid = (left + right) / 2;
40
41                MergeSortTask leftTask = new MergeSortTask(array, left, mid);
42                MergeSortTask rightTask = new MergeSortTask(array, mid + 1, right);
43
44                invokeAll(leftTask, rightTask);
45
46                merge(array, left, mid, right);
47            }
48        }
49
50        // Merge function
51        void merge(int[] array, int left, int mid, int right) {
52            // Create temporary arrays
53            int n1 = mid - left + 1;
54            int n2 = right - mid;
55
56            int[] L = new int[n1];
57            int[] R = new int[n2];
58
59            // Copy data to temporary arrays L[] and R[]
60            for (int i = 0; i < n1; i++)
61                L[i] = array[left + i];
62            for (int j = 0; j < n2; j++)
63                R[j] = array[mid + 1 + j];
64
65            // Merge the temporary arrays back into array[left..right]
66            int i = 0, j = 0, k = left;
67            while (i < n1 && j < n2)
68                if (L[i] <= R[j])
69                    array[k++] = L[i++];
70                else
71                    array[k++] = R[j++];
72
73            // Copy the remaining elements of L[], if there are any
74            while (i < n1)
75                array[k++] = L[i++];
76
77            // Copy the remaining elements of R[], if there are any
78            while (j < n2)
79                array[k++] = R[j++];
80        }
81    }
82}
```

Time Complexity Analysis:

Sequential Merge Sort:

- Time Complexity: $O(n \log n)$
- Best Case: $O(n \log n)$ - In all cases, the algorithm has to perform $\log n$ merge steps, and in each step, it takes linear time to merge.
- Worst Case: $O(n \log n)$ - Same as the best case. Merge Sort always performs the same number of operations regardless of the input.

Multithreaded Merge Sort:

- Time Complexity: $O(n \log n)$
- Best Case: $O(n \log n)$ - Similar to the sequential case. The algorithm still has to perform $\log n$ merge steps, but with the potential for parallelism.
- Worst Case: $O(n \log n)$ - Even in the worst case, where parallelism may not be fully utilized, the overall time complexity remains $O(n \log n)$.

Performance Analysis:

Sequential Merge Sort:

- **Advantages:**
 - Simplicity: Easier to implement and understand.
 - Predictability: Performance doesn't depend on system architecture or the number of available threads.
- **Disadvantages:**
 - Limited Scalability: As it is sequential, it does not make full use of multi-core processors. The execution time increases linearly with the input size.

Multithreaded Merge Sort:

- **Advantages:**
 - Improved Performance on Multi-core Processors: Can utilize multiple cores to perform parallel sorting, potentially leading to faster execution times for large datasets.
 - Better Scalability: Scales better with increasing input sizes.
- **Disadvantages:**
 - Increased Complexity: The multithreaded version requires managing thread synchronization, which can introduce complexity and potential for bugs.
 - Potentially Suboptimal for Small Inputs: The overhead of thread creation and management can outweigh the benefits for small datasets.