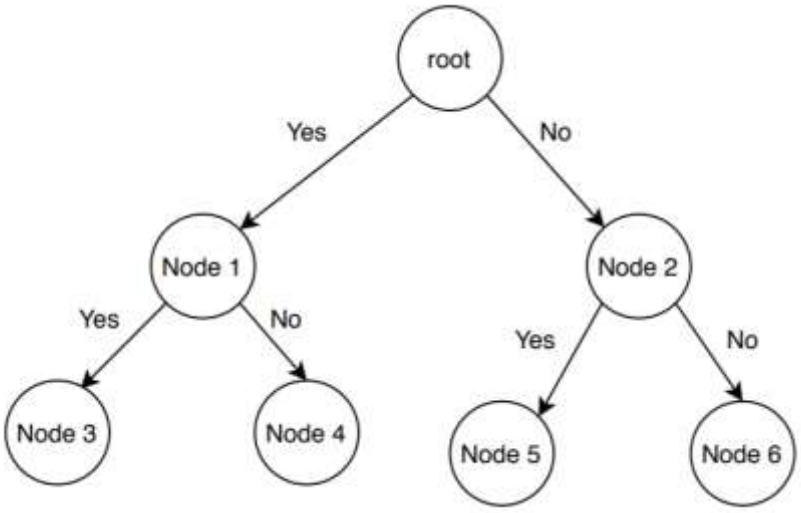
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	8		

AIM:	To implement 15 puzzle sum using the branch and bound
THEORY:	**What is Branch and Bound?** Branch and bound algorithms are used to find the optimal solution for combinatory, discrete, and general mathematical optimization problems**.** In general, given an NP-Hard problem, a branch and bound algorithm explores the entire search space of possible solutions and provides an optimal solution. A branch and bound algorithm consist of stepwise enumeration of possible candidate solutions by exploring the entire search space. With all the possible solutions, we first build a rooted decision tree. The root node represents the entire search space:  **Advantages**

In a branch and bound algorithm, we don't explore all the nodes in the tree. That's why the time complexity of the branch and bound algorithm is less when compared with other algorithms.
If the problem is not large and if we can do the branching in a reasonable amount of time, it finds an optimal solution for a given problem.
The branch and bound algorithm find a minimal path to reach the optimal solution for a given problem. It doesn't repeat nodes while exploring the tree.

Disadvantages

The branch and bound algorithm are time-consuming. Depending on the size of the given problem, the number of nodes in the tree can be too large in the worst case.
Also, parallelization is extremely difficult in the branch and bound algorithm.

Lets solve an example for branch and bound for 15 puzzle problem

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

Initial arrangement

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

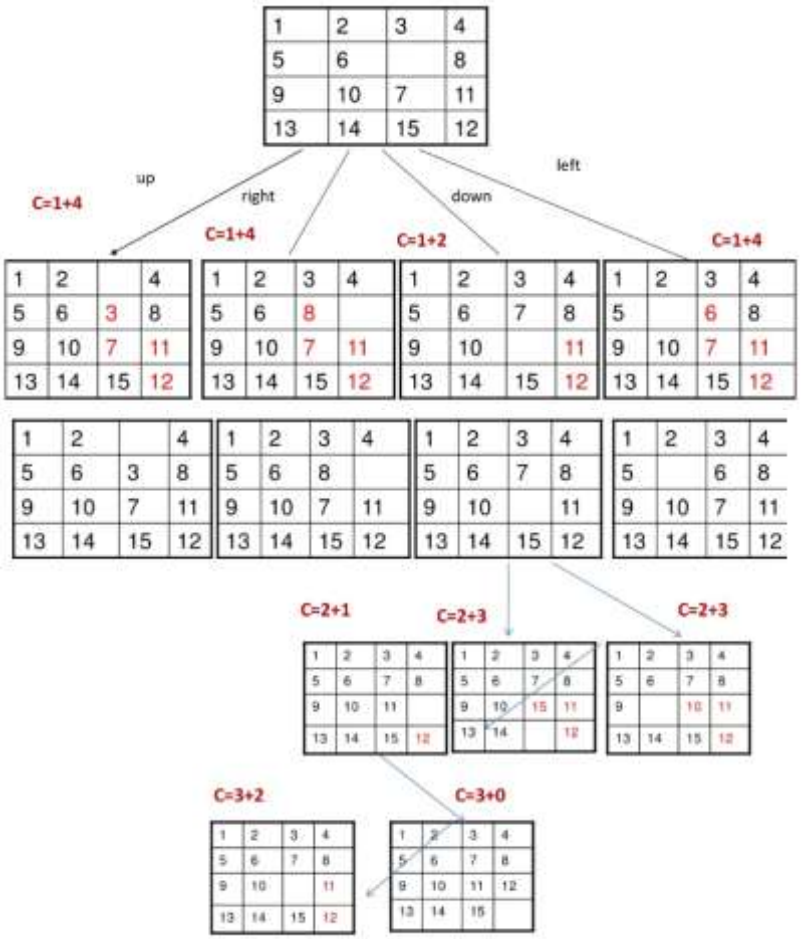
Goal arrangement

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



PSEUDOCODE:

```
struct list_node
{
list_node *next;
list_node *parent;
float cost;
};
algorithm LCSearch(list_node *t)
{
        if (*t is an answer node)
      {
         print(*t);
         return;
}
        E =t;
Initialize the list of live nodes to be empty;
while (true) {
for each child x of E
{
      if x is an answer node {
        print the path from x to t;
        return; }
      Add (x);
      x->parent = E; }
     if there are no more live nodes
   {
        print ("No answer node");
        return;
   }
      E = Least();
   }
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	
CODE:	import java.util.*; class Node { int[][] array; int misplaced; int recent; *// 1 up| 2 right| 3 down| 4 left|* } class branchBounds { int blankRow; int blankRowIndex;*// row index* int blackColIndex;*// column index* int totalCost; String isChoosen = "None"; int[][] targetMatrix = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 0 } }; boolean isSolvable(int[][] arr, int row_len) { int inv = inversions(arr); *// method that counts the number of inversions (i<j, arr[i]>arr[j])* if (arr.length % 2 != 0) {*// checks if n is odd* if (inv % 2 == 0) {*// if the length is odd, and inversions are even the puzzle is solvable* return true; } *// row: even AND inversion: odd =>solvable* *// row: odd AND inversion: even =>solvable* } else {*// if n is even* if (this.blankRow % 2 == 0 && inv % 2 != 0 || this.blankRow % 2 != 0 && inv % 2 == 0) { return true; } } return false;

```
    }

    int inversions(int[][] arr) {
        // count the number of inversions
        int no_inversions = 0;
        int[] arr2 = new int[arr.length * arr.length]; // 1d
array
        int k = 0;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                arr2[k] = arr[i][j]; // converting 2d array into 1d
array
                if (arr[i][j] == 0) {// blank tile(finding the index
of the blank tile)
                    this.blankRow = arr.length - i; // findimg the
index according to the convention (bottom->top):
                                                    // 1..2 3.
                    this.blankRowIndex = i;
                    this.blackColIndex = j;
                }
                k++;
            }
        }
        printArray(arr);
        System.out.println();
        System.out.println("----------------------");
        System.out.println("X Mark is at -> " +
(blankRowIndex + 1) + ", " + (blackColIndex + 1));
        System.out.println("----------------------");

        System.out.println();
        for (int i = 0; i < arr2.length; i++) {
            for (int j = i + 1; j < arr2.length; j++) {
                if (arr2[i] > arr2[j] && arr2[j] != 0) { // not
considering the blank tile while finding out the
                    // inversions
                    no_inversions++;
                }
            }
```

```
        }
        System.out.println("Total inversions: " +
no_inversions);
        return no_inversions;
    }

    boolean isMatched(int[][] arr, int[][] sel) {
        // check if the selected array is the target array
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                if (arr[i][j] != sel[i][j]) {// checks the array with
the target array, as soon as it matches the while
                    // loop exits
                    return false;
                }
            }
        }
        return true;
    }

    int mismatch(int[][] arr) {
        // misplaced tiles
        int mislocations = 0;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                if (arr[i][j] != this.targetMatrix[i][j] && arr[i][j]
!= 0) {// checks the number of elements that dont
                                                                //
match the target array
                    mislocations++;
                }
            }
        }
        return mislocations;
    }

    void solve(int[][] arr) {
        // Solving the puzzle
        int cost = Integer.MAX_VALUE;
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        int level = 0;
        int[][] temp_array = new int[arr.length][arr.length];

        while (!isMatched(arr, targetMatrix)) {
            level++;
            for (int i = 0; i < arr.length; i++) {
                for (int j = 0; j < arr.length; j++) {
                    // blank tile index
                    // checking where the x mark is
                    if (arr[i][j] == 0) {
                        this.blankRow = arr.length - i;
                        this.blankRowIndex = i;
                        this.blackColIndex = j;
                    }

                }
            }
            System.out.print("\nCosts->\n");
            int left[][] = leftShift(arr, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Left shift: " + ((int)
mismatch(left) + (int) level));

            int up[][] = upShift(arr, left, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Up shift: " + ((int)
mismatch(up) + (int) level));

            int right[][] = rightShift(arr, up, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Right shift: " + ((int)
mismatch(right) + (int) level));

            int[][] down = downShift(arr, right, temp_array,
blankRowIndex, blackColIndex, level, cost);
            System.out.println("Down shift: " + ((int)
mismatch(down) + (int) level));
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
        for (int i = 0; i < down.length; i++) { // storing the array for down shift
            for (int j = 0; j < down.length; j++) {
                down[i][j] = arr[i][j];
            }
        }

        if (blankRowIndex != arr.length - 1) {// checks if the down shift is possible and doesnt go out of bounds
            int temp = down[blankRowIndex + 1][blackColIndex];
            down[blankRowIndex + 1][blackColIndex] = down[blankRowIndex][blackColIndex];
            down[blankRowIndex][blackColIndex] = temp;
        }

        if (mismatch(down) + level <= cost) {// checking if the cost is lower
            cost = mismatch(down) + level;
            for (int i = 0; i < left.length; i++) {
                for (int j = 0; j < left.length; j++) {
                    temp_array[i][j] = down[i][j];
                }
            }

        }
        System.out.print("\nMinimum possible cost: " + ((int) mismatch(down) + (int) level) + "\n");

        System.out.print("\nOperation performed: " + isChoosen + "\n\n");

        // after filtering through the whole level printing the current
        for (int i = 0; i < down.length; i++) {
            for (int j = 0; j < down.length; j++) {
                arr[i][j] = temp_array[i][j];
                // status of the matrix
            }
```

```
            }
            printArray(arr);

            totalCost = totalCost + cost;

        }
        System.out.println("Total cost: " + totalCost);
    }

    public void printArray(int arr[][]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.println("----------------------------");
            for (int j = 0; j < arr.length; j++) {
                System.out.print(String.format("| %3d  ",
arr[i][j]));
            }
            System.out.println("|");
        }
        System.out.println("----------------------------");
    }

    public int[][] leftShift(int[][] arr, int[][] temp_array, int
blankRowIndex, int blackColIndex, int level,
            int cost) {
        // left shift
        int[][] left = new int[arr.length][arr.length];
        // storing the array for left shift
        for (int i = 0; i < left.length; i++) {
            for (int j = 0; j < left.length; j++) {
                left[i][j] = arr[i][j];
            }
        }
        // checks if the left shift is possible and doesnt go out
of bounds
        if (blackColIndex != 0) {
            int temp = left[blankRowIndex][blackColIndex];
            left[blankRowIndex][blackColIndex] =
left[blankRowIndex][blackColIndex - 1];
            left[blankRowIndex][blackColIndex - 1] = temp;
```

```
        }
        // checking if the cost is minimum
        if (mismatch(left) + level <= cost) {
            isChoosen = "Shifting left";
            cost = mismatch(left) + level; // assigning lower
cost
            for (int i = 0; i < left.length; i++) {
                for (int j = 0; j < left.length; j++) {
                    temp_array[i][j] = left[i][j]; // potential
candidate
                }
            }
        }
        return temp_array;
    }

    public int[][] rightShift(int[][] arr, int[][] up, int[][]
temp_array, int blankRowIndex, int blackColIndex,
            int level,
            int cost) {
        // right shift
        int[][] right = new int[arr.length][arr.length];
        // storing the array for right shift
        for (int i = 0; i < right.length; i++) {
            for (int j = 0; j < right.length; j++) {
                right[i][j] = arr[i][j];
            }
        }
        // checks if the right shift is possible and doesnt go
out of bounds
        if (blackColIndex != arr.length - 1) {
            int temp = right[blankRowIndex][blackColIndex];
            right[blankRowIndex][blackColIndex] =
right[blankRowIndex][blackColIndex + 1];
            right[blankRowIndex][blackColIndex + 1] = temp;
        }
        // checking if the cost is minimum
        if (mismatch(right) + level <= cost) {
            isChoosen = "Shifting right";
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
            cost = mismatch(right) + level; // assigning lower
cost
            for (int i = 0; i < right.length; i++) {
               for (int j = 0; j < right.length; j++) {
                  temp_array[i][j] = right[i][j]; // potential
candidate
               }
            }
         }
         return temp_array;
      }

      public int[][] upShift(int arr[][], int left[][], int
temp_array[][], int blankRowIndex, int blankColIndex,
            int level,
            int cost) {
         int[][] up = new int[arr.length][arr.length];

         for (int i = 0; i < up.length; i++) {
            for (int j = 0; j < up.length; j++) {
               up[i][j] = arr[i][j]; // storing the array for up
shift
            }
         }
         if (blankRowIndex != 0) { // checks if the up shift is
possible and doesn't go out of bounds

            int temp = up[blankRowIndex - 1][blankColIndex];
            up[blankRowIndex - 1][blankColIndex] =
up[blankRowIndex][blankColIndex];
            up[blankRowIndex][blankColIndex] = temp;
         }

         if (mismatch(up) + level <= cost) { // checking if the
cost is lower
            isChoosen = "Shifting Up";
            cost = mismatch(up) + level;
            for (int i = 0; i < left.length; i++) {
               for (int j = 0; j < left.length; j++) {
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

```
                temp_array[i][j] = up[i][j];
            }
        }
    }
    return temp_array;
}

public int[][] downShift(int arr[][], int left[][], int
temp_array[][], int blankRowIndex, int blankColIndex,
        int level,
        int cost) {
    int[][] down = new int[arr.length][arr.length];

    for (int i = 0; i < down.length; i++) {
        for (int j = 0; j < down.length; j++) {
            down[i][j] = arr[i][j]; // storing the array for
down shift
        }
    }
    if (blankRowIndex != arr.length - 1) { // checks if the
down shift is possible and doesn't go out of bounds
        int temp = down[blankRowIndex +
1][blankColIndex];
        down[blankRowIndex + 1][blankColIndex] =
down[blankRowIndex][blankColIndex];
        down[blankRowIndex][blankColIndex] = temp;
    }

    if (mismatch(down) + level <= cost) { // checking if
the cost is lower
        isChoosen = "Shifting Down";
        cost = mismatch(down) + level;
        for (int i = 0; i < left.length; i++) {
            for (int j = 0; j < left.length; j++) {
                temp_array[i][j] = down[i][j];
            }
        }
    }
    return temp_array;
```

```
   }
}

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

      branchBounds obj = new branchBounds();
      System.out.println("--------------15 puzzle solve-------
---------");
      System.out.println("Input Matrix: ");
      System.out.print("\nEnter the size of the matrix: ");
      int size = sc.nextInt();
      int[][] table = new int[size][size];
      System.out.print("\nEnter the elements of the matrix:
");
      for (int i = 0; i < size; i++) {
         for (int j = 0; j < size; j++) {
            table[i][j] = sc.nextInt();
         }
      }

      System.out.println("The Length of the puzzle is: " +
table.length);
      if (obj.isSolvable(table, table[0].length)) {
         System.out.println("\nPuzzle is solvable");
         obj.solve(table);// solving the matrix

      } else {
         System.out.println("\n Puzzle is not solvable");
      }
      sc.close();

   }
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

SCREENSHOT:

Not Solvable

```
--------------15 puzzle solve----------------
Input Matrix:

Enter the size of the matrix: 4
The Length of the puzzle is: 4
----------------------------
|   3  |   9  |   1  |  15  |
----------------------------
|  14  |  11  |   4  |   6  |
----------------------------
|  13  |   0  |  10  |  12  |
----------------------------
|   2  |   7  |   8  |   5  |
----------------------------

----------------------
X Mark is at -> 3, 2
----------------------
Total inversions: 56

 Puzzle is not solvable
```

Solvable

```
Input Matrix:

Enter the size of the matrix: 4

Enter the elements of the matrix: 1 2 3 0 5 6 7 4 9 10 11 8 13 14 15 12
The Length of the puzzle is: 4
----------------------------
|   1  |   2  |   3  |   0  |
----------------------------
|   5  |   6  |   7  |   4  |
----------------------------
|   9  |  10  |  11  |   8  |
----------------------------
|  13  |  14  |  15  |  12  |
----------------------------

----------------------
X Mark is at -> 1, 4
----------------------
Total inversions: 9

Puzzle is solvable
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

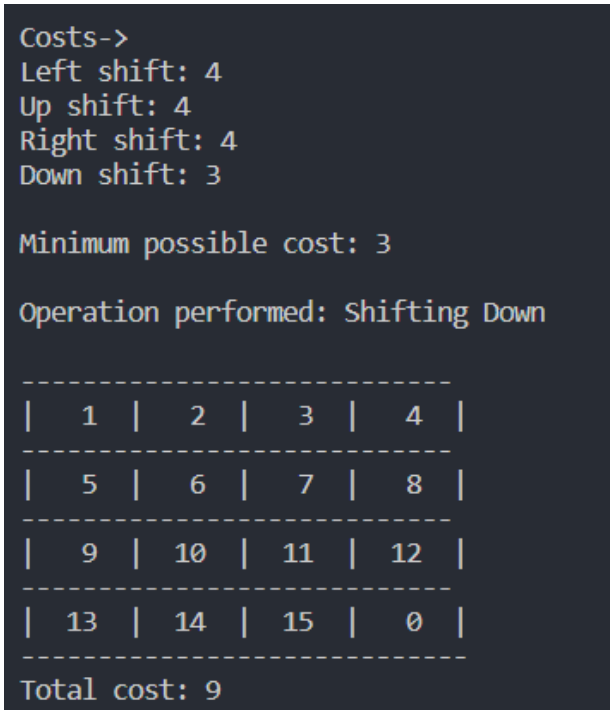
```
Costs->
Left shift: 5
Up shift: 4
Right shift: 4
Down shift: 3

Minimum possible cost: 3

Operation performed: Shifting Down

-----------------------------
|   1  |   2  |   3  |   4  |
-----------------------------
|   5  |   6  |   7  |   0  |
-----------------------------
|   9  |  10  |  11  |   8  |
-----------------------------
|  13  |  14  |  15  |  12  |
-----------------------------
```

```
Costs->
Left shift: 4
Up shift: 4
Right shift: 4
Down shift: 3

Minimum possible cost: 3

Operation performed: Shifting Down

-----------------------------
|   1  |   2  |   3  |   4  |
-----------------------------
|   5  |   6  |   7  |   8  |
-----------------------------
|   9  |  10  |  11  |   0  |
-----------------------------
|  13  |  14  |  15  |  12  |
-----------------------------
```

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

	
TIME COMPLEXITY	The Time complexity of the branch and Bound algorithm is given below: **Time Complexity: O(n^2)** The Space Complexity of the branch and Bound Algorithm is given below: **Space Complexity: O(n)**

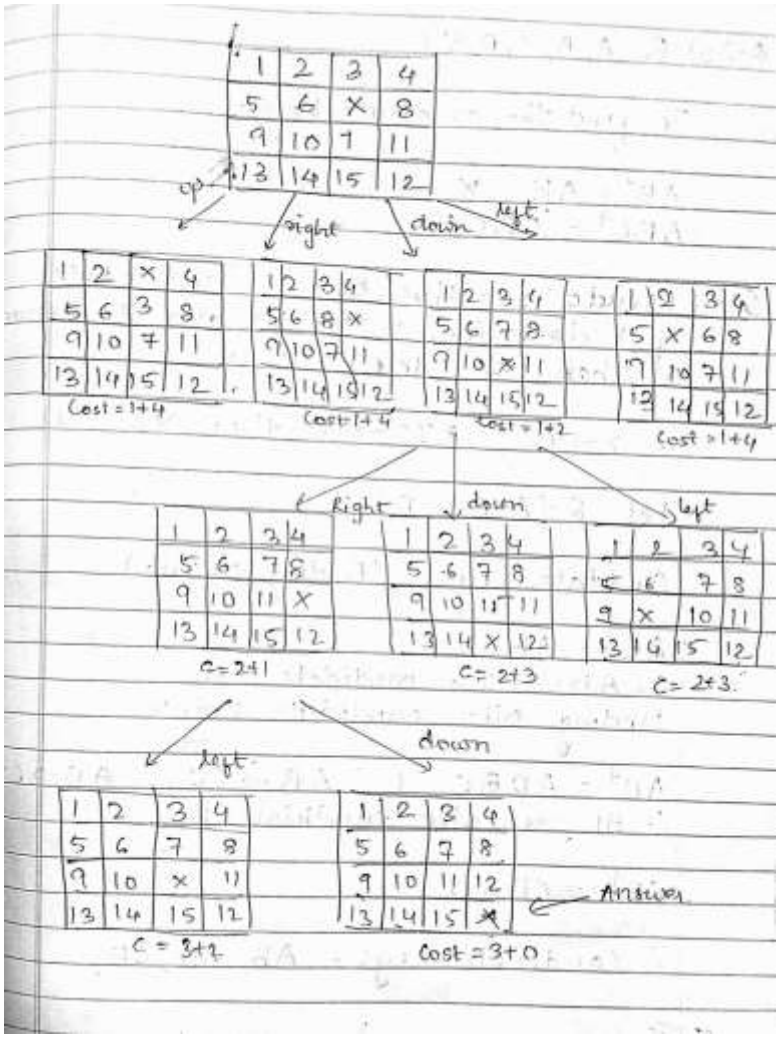
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

OUTPUT:



CONCLUSION: Things learnt during the procedural programming of the problem of branch and bound

- Learnt one of the most popular algorithms used in the optimization problem is the branch and bound algorithm.
- Learnt about some advantages and disadvantages of the branch and bound algorithm
- Also noted how and when a branch and bound algorithm would be the right choice for a user to use.
- Used branch and bound based algorithm for solving the 15 puzzle problem.