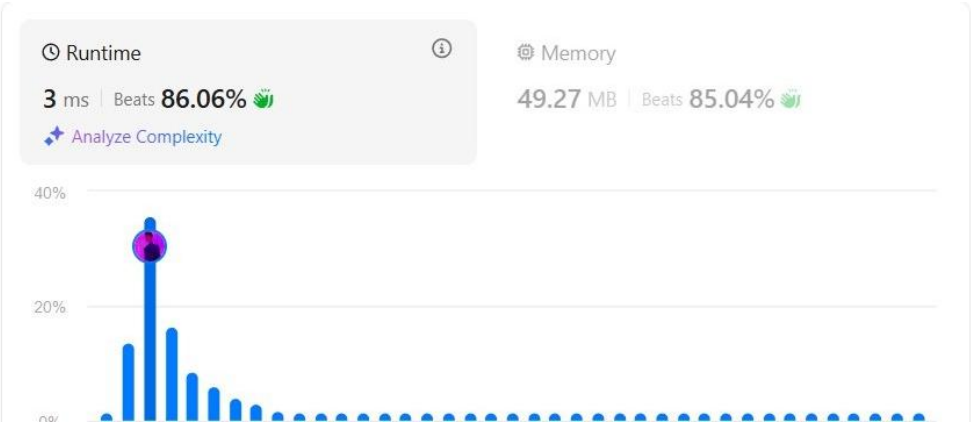
Experiment 9

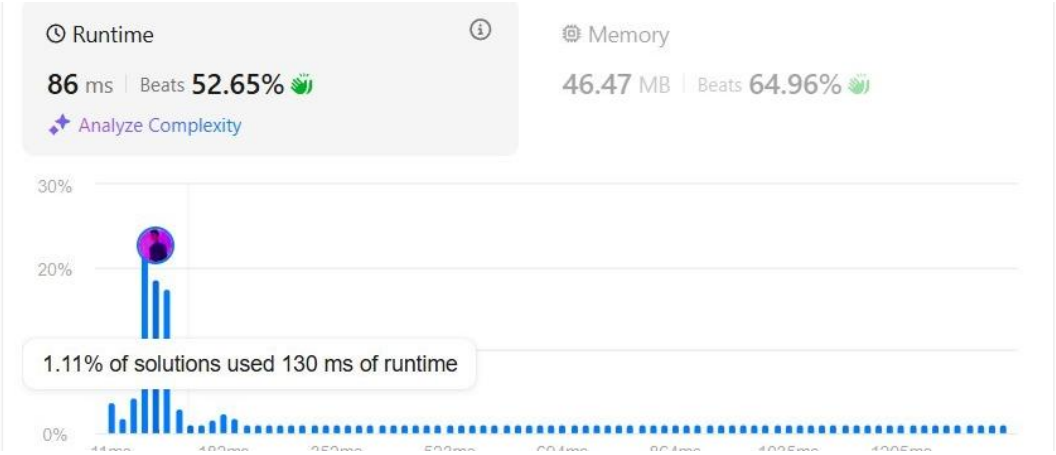
Number of Islands

```
class Solution {
    boolean [][]dp;
    public int numIslands(char[][] grid) {
        int c=0;
        for(int i=0; i<grid.length; i++){
            for(int j=0; j<grid[0].length; j++){
                if(grid[i][j]=='1'){
                    dfs(grid, i, j);
                    c++;
                }
            }
        }
        return c;
    }
    void dfs(char [][]grid, int i, int j){
        if(i<0 || j<0 || i>=grid.length || j>=grid[0].length || grid[i][j]!='1'){
            return;
        }
        grid[i][j]= '0';
        dfs(grid, i-1, j);
        dfs(grid, i+1, j);
        dfs(grid, i, j-1);
        dfs(grid, i, j+1);
    }
}
```



Word Ladder

```
class Solution {
    class Pair{
        String first;
        int second;
        Pair(String first, int second){
            this.first= first;
            this.second= second;
        }
    }
    public int ladderLength(String beginWord, String endWord,
List<String> wordList) {
        Set<String> set= new HashSet<>();
        for(String str: wordList){
            set.add(str);
        }
        if(!set.contains(endWord)) return 0;
        Queue<Pair> queue= new LinkedList<>();
        queue.offer(new Pair(beginWord, 1));
        while(!queue.isEmpty()){
            Pair temp= queue.poll();
            int step= temp.second;
            String word= temp.first;
            set.remove(word);
            if(word.equals(endWord)) return step;
            for(int i=0; i<word.length(); i++){
                for(char ch='a'; ch<='z'; ch++){
                    char []arr= word.toCharArray();
                    arr[i]=ch;
                    String replW= new String(arr);
                    if(set.contains(replW)){
                        queue.offer(new Pair(replW, step+1));
                        set.remove(replW);
                    }
                }
            }
        }
        return 0;
    }
}
```

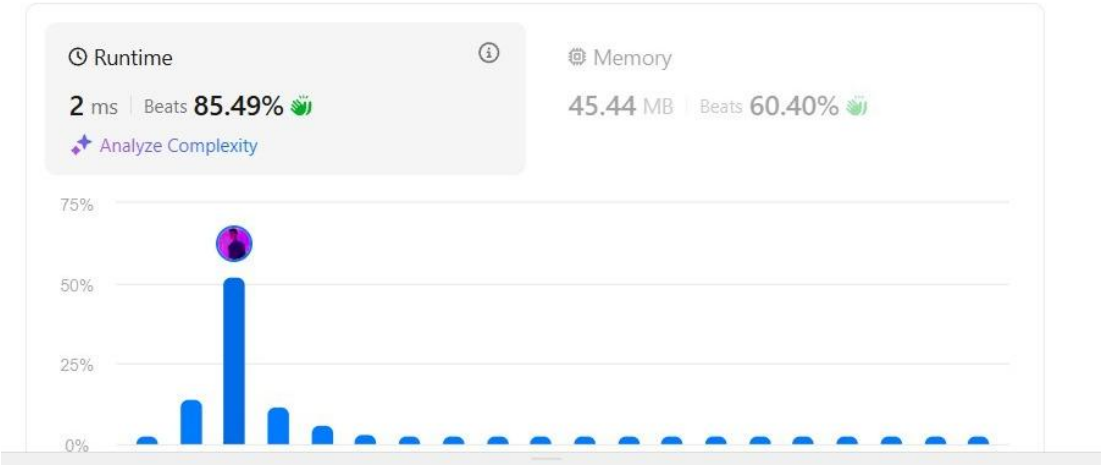


Surrounded Regions

```
class Solution {
    public void solve(char[][] board) {
        if (board.length == 0 || board[0].length == 0) return;
        if (board.length <= 2 || board[0].length <= 2) return;

        int[][] dir = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
        int[][] visited = new int[board.length][board[0].length];

        for (int j = 0; j < board[0].length; j++) {
            if (board[0][j] == 'O' && visited[0][j] == 0) {
                dfs(0, j, board, visited, dir);
            }
            if (board[board.length - 1][j] == 'O' && visited[board.length -
1][j] == 0) {
                dfs(board.length - 1, j, board, visited, dir);
            }
        }

        for (int i = 0; i < board.length; i++) {
            if (board[i][0] == 'O' && visited[i][0] == 0) {
                dfs(i, 0, board, visited, dir);
            }
            if (board[i][board[0].length - 1] == 'O' &&
visited[i][board[0].length - 1] == 0) {
                dfs(i, board[0].length - 1, board, visited, dir);
            }
```

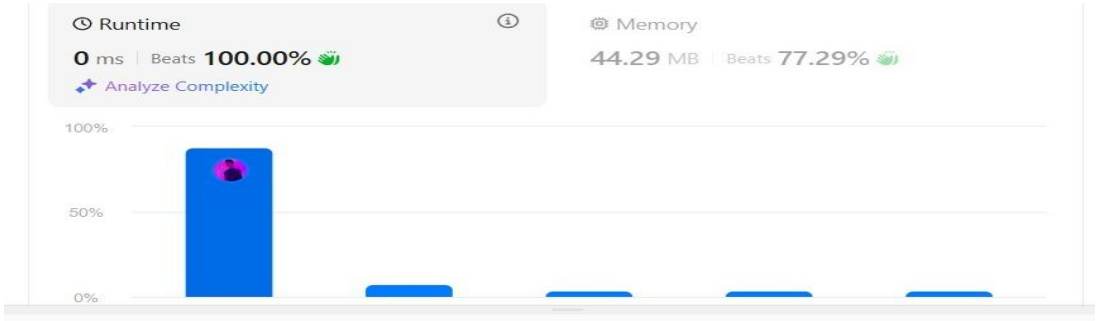
```
        }

        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                if (board[i][j] == 'O' && visited[i][j] == 0) {
                    board[i][j] = 'X';
                }
            }
        }
    }

    void dfs(int i, int j, char[][] board, int[][] visited, int[][] dir) {
        visited[i][j] = 1;
        for (int k = 0; k < 4; k++) {
            int r = i + dir[k][0];
            int c = j + dir[k][1];
            if (r >= 0 && r < board.length && c >= 0 && c <
board[0].length
                && board[r][c] == 'O' && visited[r][c] == 0) {
                dfs(r, c, board, visited, dir);
            }
        }
    }
}
```

Binary Tree Maximum Path Sum



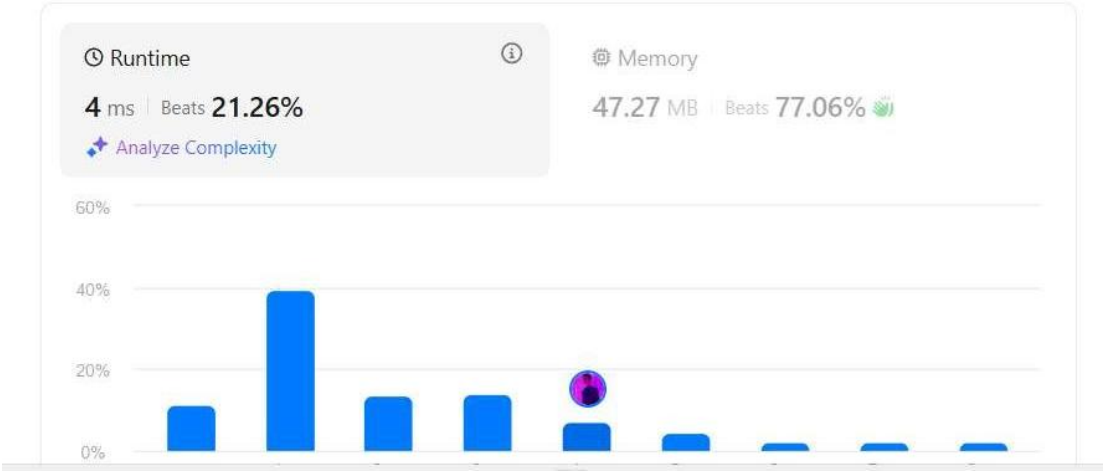

```
/**
```

```
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    int max= Integer.MIN_VALUE;
    public int maxPathSum(TreeNode root) {
        if(root==null){
            return 0;
        }
        dfs(root);
        return max;
    }
    int dfs(TreeNode root){
        if(root==null){
            return 0;
        }
        int lN= Math.max(0, dfs(root.left));
        int rN= Math.max(0, dfs(root.right));
        max= Math.max(max, lN+rN+ root.val);
        return Math.max(lN, rN)+ root.val;
    }
}
```



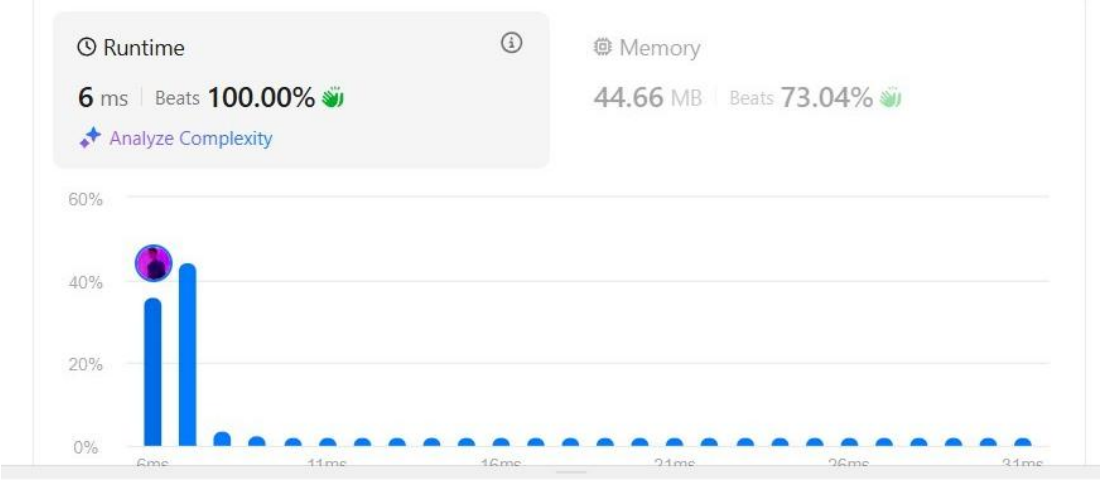
Friend Circles

```
class Solution {
    public int findCircleNum(int[][] isConnected) {
        int[] visited = new int[isConnected.length];
        int c=0;

        for (int i = 0; i < isConnected.length; i++) {
            if (visited[i] == 0) {
                bfs(isConnected, visited, i);
                c++;
            }
        }

        return c++;
    }

    private void bfs(int[][] isConnected, int[] visited, int start) {
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(start);

        while (!queue.isEmpty()) {
            int ind = queue.poll();
            visited[ind] = 1;

            for (int i = 0; i < isConnected[ind].length; i++) {
                if (isConnected[ind][i] == 1 && visited[i] == 0) {
                    queue.offer(i);
                }
            }
        }
    }
}
```



Lowest Common Ancestor of a Binary Tree

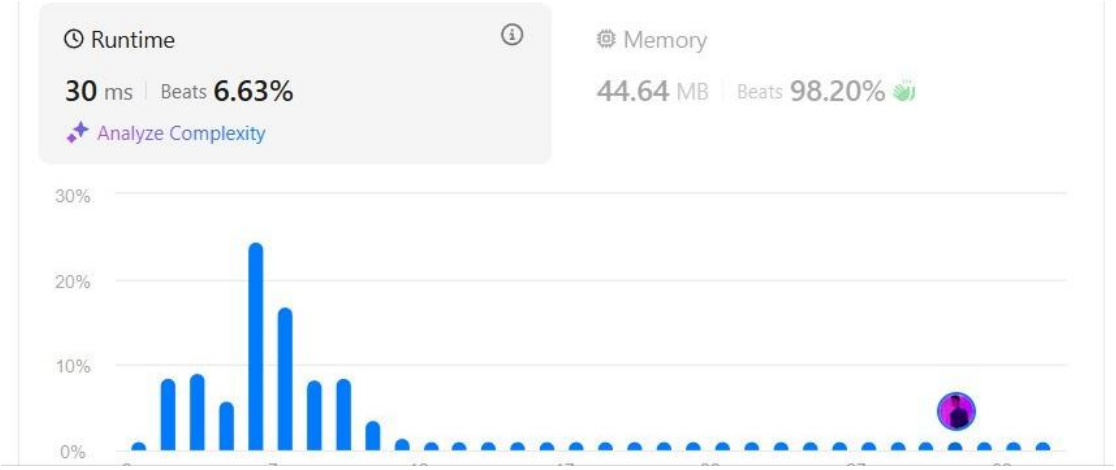
```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root,
TreeNode p, TreeNode q) {
        if(root==null || root==p || root==q){
            return root;
        }
        TreeNode left= lowestCommonAncestor(root.left, p, q);
        TreeNode right= lowestCommonAncestor(root.right, p, q);
        return left==null?right:right==null?left:root;
    }
}
```



Course Schedule

```
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
     if (numCourses == 0) return true;

        int[] indegree = new int[numCourses];
        for (int i = 0; i < prerequisites.length; i++) {
            indegree[prerequisites[i][0]]++;
        }

        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < numCourses; i++) {
            if (indegree[i] == 0) {
                queue.offer(i);
            }
        }

        int index = 0;
        while (!queue.isEmpty()) {
            int prerequisite = queue.poll();
            index++;
            for (int i = 0; i < prerequisites.length; i++) {
                if (prerequisites[i][1] == prerequisite) {
                    indegree[prerequisites[i][0]]--;
                    if (indegree[prerequisites[i][0]] == 0) {
                        queue.offer(prerequisites[i][0]);
                    }
```

```
            }
        }
    }

    return index == numCourses;
  }
}
```



Longest Increasing Path in a Matrix

```
class Solution {
    public static final int[][] dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

public int longestIncreasingPath(int[][] matrix) {
    if(matrix.length == 0) return 0;
    int m = matrix.length, n = matrix[0].length;
    int[][] cache = new int[m][n];
    int max = 1;
    for(int i = 0; i < m; i++) {
        for(int j = 0; j < n; j++) {
            int len = dfs(matrix, i, j, m, n, cache);
            max = Math.max(max, len);
        }
    }
    return max;
}

public int dfs(int[][] matrix, int i, int j, int m, int n, int[][] cache) {
```

```
    if(cache[i][j] != 0) return cache[i][j];
    int max = 1;
    for(int[] dir: dirs) {
        int x = i + dir[0], y = j + dir[1];
        if(x < 0 || x >= m || y < 0 || y >= n || matrix[x][y] <= matrix[i][j])
continue;
        int len = 1 + dfs(matrix, x, y, m, n, cache);
        max = Math.max(max, len);
    }
    cache[i][j] = max;
    return max;
}
}
```

Course Schedule II

```
class Solution {
    public int[] findOrder(int numCourses, int[][] prerequisites) {
        if (numCourses == 0) return null;
    int indegree[] = new int[numCourses], order[] = new
int[numCourses], index = 0;
    for (int i = 0; i < prerequisites.length; i++)
        indegree[prerequisites[i][0]]++;

    Queue<Integer> queue = new LinkedList<Integer>();
    for (int i = 0; i < numCourses; i++)
        if (indegree[i] == 0) {
            order[index++] = i;
            queue.offer(i);
```

```
        }

    while (!queue.isEmpty()) {
        int prerequisite = queue.poll();
        for (int i = 0; i < prerequisites.length; i++)  {
            if (prerequisites[i][1] == prerequisite) {
                indegree[prerequisites[i][0]]--;
                if (indegree[prerequisites[i][0]] == 0) {
                    order[index++] = prerequisites[i][0];
                    queue.offer(prerequisites[i][0]);
                }
            }
        }
    }

    return (index == numCourses) ? order : new int[0];
    }
}
```