



Experiment 4

Student Name: Sabir Ali

UID: 22BET10033

Branch: BE-IT

Section/Group: 22BET-IOT-701/A

Semester: 6

Date of Performance: 14/02/2025

Subject Name: AP2

Subject Code: 22ITP-351

1. Aim: Implement the following problem:- Longest Nice Substring, Reverse Bits, Number of 1 bits, Max Subarray, Search 2d matrix 2, Super Pow, Beautiful Array, The Skyline Problem, Reverse Pairs, Longest increasing subsequence 2.

2. Objective: To develop a deep understanding and proficiency in solving a variety of algorithmic challenges, including string manipulation, bit manipulation, dynamic programming, array operations, matrix manipulation, and mathematical computations. Focus will be on improving skills in problem-solving, algorithm design, and optimizing solutions for efficiency.

3. Implementation/Code:

(A) Longest Nice Substring

```
public class Solution {  
    public String longestNiceSubstring(String s) {  
        if (s.length() <= 1) return "";  
        for (int i = 0; i < s.length(); i++) {  
            if (!isNice(s.charAt(i), s)) {  
                String left = longestNiceSubstring(s.substring(0, i));  
                String right = longestNiceSubstring(s.substring(i + 1));  
                return left.length() > right.length() ? left : right;  
            }  
        }  
        return s;  
    }  
    private boolean isNice(char ch, String s) {  
        return s.contains(String.valueOf(Character.toLowerCase(ch))) &&  
            s.contains(String.valueOf(Character.toUpperCase(ch)));  
    }  
}
```

(B) Reverse Bits

```
public class Solution {  
    public int reverseBits(int n) {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
int result = 0;
for (int i = 0; i < 32; i++) {
    result = result << 1;
    result |= (n & 1);
    n >>= 1;
}
return result;
}
```

(C) Number of 1 Bits

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int count = 0;
        while (n) {
            n &= (n - 1);
            count++;
        }
        return count;
    }
};
```

(D) Maximum Subarray

```
public class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }
        return maxSum;
    }
}
```

(E) Search a 2D Matrix II

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {  
    return false;  
}  
int row = 0;  
int col = matrix[0].length - 1;  
while (row < matrix.length && col >= 0) {  
    if (matrix[row][col] == target) {  
        return true;  
    } else if (matrix[row][col] < target) {  
        row++;  
    } else {  
        col--;  
    }  
}  
return false;  
}
```

(F) Super Pow

```
public class Solution {  
    private static final int MOD = 1337;  
    public int superPow(int a, int[] b) {  
        a = a % MOD;  
        int result = 1;  
        for (int i = b.length - 1; i >= 0; i--) {  
            result = (result * modExp(a, b[i])) % MOD;  
            a = modExp(a, 10) % MOD;  
        }  
        return result;  
    }  
    private int modExp(int base, int exp) {  
        int result = 1;  
        base = base % MOD;  
        while (exp > 0) {  
            if (exp % 2 == 1) {  
                result = (result * base) % MOD;  
            }  
            base = (base * base) % MOD;  
            exp /= 2;  
        }  
    }  
}
```

```
}  
return result;  
}  
}
```

(G) Beautiful Array

```
public class Solution {  
    public int[] beautifulArray(int N) {  
        List<Integer> res = new ArrayList<>();  
        res.add(1);  
        while (res.size() < N) {  
            List<Integer> temp = new ArrayList<>();  
            for (int x : res) {  
                if (x * 2 - 1 <= N) {  
                    temp.add(x * 2 - 1);  
                }  
            }  
            for (int x : res) {  
                if (x * 2 <= N) {  
                    temp.add(x * 2);  
                }  
            }  
            res = temp;  
        }  
        return res.stream().mapToInt(i -> i).toArray();  
    }  
}
```

(H) The Skyline Problem

```
public class Solution {  
    public List<List<Integer>> getSkyline(int[][] buildings) {  
        List<List<Integer>> result = new ArrayList<>();  
        List<int[]> heights = new ArrayList<>();  
        for (int[] b : buildings) {  
            heights.add(new int[]{b[0], -b[2]});  
            heights.add(new int[]{b[1], b[2]});  
        }  
        Collections.sort(heights, (a, b) -> a[0] == b[0] ? a[1] - b[1] : a[0] - b[0]);  
    }  
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
pq.add(0);
int prevMaxHeight = 0;
for (int[] h : heights) {
    if (h[1] < 0) {
        pq.add(-h[1]);
    } else {
        pq.remove(h[1]);
    }
    int currentMaxHeight = pq.peek();
    if (currentMaxHeight != prevMaxHeight) {
        result.add(Arrays.asList(h[0], currentMaxHeight));
        prevMaxHeight = currentMaxHeight;
    }
}
return result;
}
```

(I) Reverse Pairs

```
public class Solution {
    public int reversePairs(int[] nums) {
        if (nums == null || nums.length <= 1) {
            return 0;
        }
        return mergeSort(nums, 0, nums.length - 1);
    }
    private int mergeSort(int[] nums, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);
        int j = mid + 1;
        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2L * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
merge(nums, left, mid, right);
return count;
}
private void merge(int[] nums, int left, int mid, int right) {
int[] temp = new int[right - left + 1];
int i = left, j = mid + 1, k = 0;
while (i <= mid && j <= right) {
if (nums[i] <= nums[j]) {
temp[k++] = nums[i++];
} else {
temp[k++] = nums[j++];
}
}
while (i <= mid) {
temp[k++] = nums[i++];
}
while (j <= right) {
temp[k++] = nums[j++];
}
System.arraycopy(temp, 0, nums, left, temp.length);
}
}
```

(J) Longest Increasing Subsequence II

```
class Solution {
class SegmentTree {
int[] tree;
int n;
SegmentTree(int size) {
n = size;
tree = new int[4 * n];
}
void update(int index, int value, int left, int right, int node) {
if (left == right) {
tree[node] = value;
return;
}
int mid = (left + right) / 2;
if (index <= mid) update(index, value, left, mid, 2 * node + 1);
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
else update(index, value, mid + 1, right, 2 * node + 2);
tree[node] = Math.max(tree[2 * node + 1], tree[2 * node + 2]);
}
int query(int ql, int qr, int left, int right, int node) {
if (ql > right || qr < left) return 0;
if (ql <= left && qr >= right) return tree[node];
int mid = (left + right) / 2;
return Math.max(query(ql, qr, left, mid, 2 * node + 1), query(ql, qr, mid + 1, right, 2 * node + 2));
}
void update(int index, int value) {
update(index, value, 0, n - 1, 0);
}
int query(int ql, int qr) {
return query(ql, qr, 0, n - 1, 0);
}
}
public int lengthOfLIS(int[] nums, int k) {
int maxVal = 0;
for (int num : nums) maxVal = Math.max(maxVal, num);
SegmentTree segTree = new SegmentTree(maxVal + 1);
int result = 0;
for (int num : nums) {
int maxLen = segTree.query(Math.max(0, num - k), num - 1) + 1;
segTree.update(num, maxLen);
result = Math.max(result, maxLen);
}
return result;
}
}
```



4. Output:

Testcase

Test Result

Accepted Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

s =
"c"

Output

""

Expected

""

[illegible]



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

(C) Number of 1 Bits

✓ Testcase | > Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

n =
128

Output

1

Expected

1

♥ Contribute a testcase

(D) Maximum Subarray

✓ Testcase | > Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[-2,1,-3,4,-1,2,1,-5,4]

Output

6

Expected

6

♥ Contribute a testcase



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

(E) Search a 2D Matrix II

☒ Testcase ☒ Test Result

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]

target =
20

Output

false

Expected

false

(F) Super Pow

☒ Testcase ☒ Test Result

Accepted Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

a =
1

b =
[4,3,3,8,5,2]

Output

1

Expected

1

Contribute a testcase



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

(G) Beautiful Array

☒ Testcase | [Test Result](#)

Accepted Runtime: 1 ms

- Case 1
- Case 2

Input

n =
5

Output

[1,5,3,2,4]

Expected

[3,1,2,5,4]

[Contribute a testcase](#)

(H) The Skyline Problem

☒ Testcase | [Test Result](#) ⌵

Accepted Runtime: 1 ms

- Case 1
- Case 2

Input

buildings =
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]

Output

[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]

Expected

[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]

[Contribute a testcase](#)



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

(I) Reverse Pairs

☒ Testcase | [Test Result](#)



Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

```
nums =  
[2,4,3,5,1]
```

Output

3

Expected

3

[Contribute a testcase](#)

(J) Longest Increasing Subsequence II

☒ Testcase | [Test Result](#)



Accepted Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

```
nums =  
[1,5]
```



```
k =  
1
```

Output

1

Expected

1



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

5. Learning Outcomes:-

- Understanding how binary representation and bitwise operations (&, |, ^, >>, <<) optimize problem-solving.
- Learning how to make optimal choices at each step (like Kadane's Algorithm) to improve efficiency.
- Reducing brute-force approaches ($O(n^2)$ or worse) to more efficient ones ($O(n)$ or $O(\log n)$) for better performance.
- Breaking problems into smaller parts, identifying patterns, and applying the right algorithm.
- Writing clean, efficient code, avoiding logical errors, and testing with edge cases.