# Experiment 4

**Student Name:** Tarun Baliyan **UID:** 22BET10184

**Branch:** IT **Section/Group:** 22BET\_IOT-702/A

**Semester:** 5th **Date of Performance:** 14/02/25

**Subject Name:** Advance Programming-II **Subject Code:** 22ITP-367

**Problem: 1.4.1: Longest Nice Substring**

**Problem Statement:** A string s is considered **nice** if, for every character in the string, the character’s uppercase and lowercase forms both exist in the string.

1. **Objective:** Find the longest contiguous substring where every character has both its uppercase and lowercase counterpart present.
2. **Code:**

## class Solution:

## def longestNiceSubstring(self, s: str) -> str:

## # Base case: if the string is empty or has only one character, return ""

## if len(s) < 2:

## return ""

## 

## # Check for invalid characters

## for i, ch in enumerate(s):

## if ch.swapcase() not in s:

## # Split around the invalid character and check both parts

## left = self.longestNiceSubstring(s[:i])

## right = self.longestNiceSubstring(s[i+1:])

## # Return the longer substring

## return left if len(left) >= len(right) else right

## 

## # If all characters are valid, return the entire string

## return s

## 3. Result:

## 

**Problem 1.4.2: Reverse Bits**

**Problem Statement:** You are given a 32-bit unsigned integer n. Your task is to reverse the bits of n and return the result as an unsigned integer.

1. **Objective:** Reverse the order of bits in the given 32-bit unsigned integer.
2. **Code:**

class Solution:

def reverseBits(self, n: int) -> int:

result = 0

for i in range(32):

# Shift result left by 1 to make room for the next bit

result <<= 1

# Extract the last bit of n and add it to result

result |= n & 1

# Shift n right by 1 to process the next bit

n >>= 1

return result

1. **Result:**

## 

**Problem 1.4.3: Number of 1 bits**

**Problem Statement:** You are given a 32-bit unsigned integer n. Your task is to return the number of '1' bits it has, also known as the **Hamming Weight**.

1. **Objective:** Count the number of '1' bits in the 32-bit binary representation of n.
2. **Code:**

class Solution:

def hammingWeight(self, n: int) -> int:

# Step 1: Sum bits in pairs

n = (n & 0x55555555) + ((n >> 1) & 0x55555555)

# Step 2: Sum bits in nibbles (4-bit groups)

n = (n & 0x33333333) + ((n >> 2) & 0x33333333)

# Step 3: Sum bits in bytes (8-bit groups)

n = (n & 0x0F0F0F0F) + ((n >> 4) & 0x0F0F0F0F)

# Step 4: Sum bits in 16-bit groups

n = (n & 0x00FF00FF) + ((n >> 8) & 0x00FF00FF)

# Step 5: Sum bits in the final 32-bit integer

n = (n & 0x0000FFFF) + ((n >> 16) & 0x0000FFFF)

return n

1. **Result:**

## 

**Problem 1.3.4: Max Subarray**

**Problem Statement:** Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

1. **Objective:** Identify the contiguous subarray within the given array that has the maximum sum.
2. **Code:**

class Solution:

def maxSubArray(self, nums: List[int]) -> int:

# Helper function that returns a tuple of:

# (maxSub, maxPrefix, maxSuffix, total) for the subarray nums[l:r+1]

def helper(l, r):

if l == r:

# Base case: for a single element, all values are the same.

return (nums[l], nums[l], nums[l], nums[l])

mid = (l + r) // 2

left = helper(l, mid)

right = helper(mid + 1, r)

# Calculate the total sum for the current segment.

total = left[3] + right[3]

# Maximum prefix sum: either the left prefix or the entire left plus the right prefix.

maxPrefix = max(left[1], left[3] + right[1])

# Maximum suffix sum: either the right suffix or the entire right plus the left suffix.

maxSuffix = max(right[2], right[3] + left[2])

# Maximum subarray sum for the current segment is the best among:

# left.maxSub, right.maxSub, or a subarray crossing the mid-point.

maxSub = max(left[0], right[0], left[2] + right[1])

return (maxSub, maxPrefix, maxSuffix, total)

return helper(0, len(nums) - 1)[0]

1. **Result:**

## 

**Problem 1.4.5: The Skyline Problem**

**Problem Statement:** Given a list of buildings, where each building is represented as a triplet [L,R,H][L, R, H][L,R,H] (with LLL as the left x-coordinate, RRR as the right x-coordinate, and HHH as the height), your task is to output the skyline formed by these buildings. The skyline is a list of "key points" [x,y][x, y][x,y] that represent where the height of the skyline changes. Key points should be output in sorted order by the x-coordinate.

1. **Objective:** Determine the key points that form the outer contour (skyline) when the buildings are viewed from a distance.
2. **Code:**

import heapq

from typing import List

class Solution:

def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:

# Create events: start events and end events

events = []

for L, R, H in buildings:

# Start event: use negative height so that higher buildings come first

events.append((L, -H, R))

# End event: height 0 to mark the end of a building

events.append((R, 0, 0))

# Sort events by x-coordinate, then by height

events.sort()

# Max-heap with a dummy building (height 0 lasting "forever")

result = []

live = [(0, float('inf'))] # (negative height, end)

for x, negH, R in events:

# Remove expired buildings from the heap

while live[0][1] <= x:

heapq.heappop(live)

# If it's a start event, add the building into the heap

if negH:

heapq.heappush(live, (negH, R))

# Current maximum height is the negative of the top of the heap

currentHeight = -live[0][0]

# If the current maximum height changes, add a key point

if not result or result[-1][1] != currentHeight:

result.append([x, currentHeight])

return result

1. **Result:**

## 