# **Experiment 4**

Student Name: Arun UID: 22BET10320

**Branch:** Information Technology Section/Group: 22BET\_IOT-701/A

Semester: 6<sup>th</sup> Subject Code: 22ITP-351

#### Problem 1

#### Aim:

Longest Nice Substring

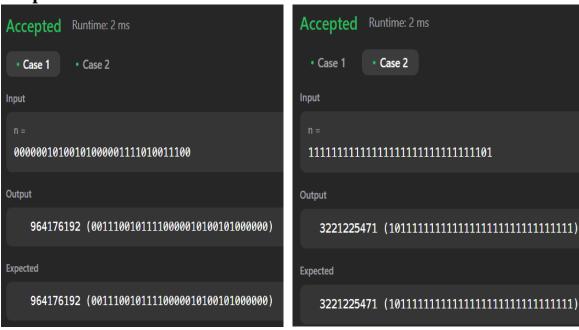
```
Code:
class Solution {
public:
  string longestNiceSubstring(string s) {
     string output = "";
     int count = 0;
     for(int i = 0; i < s.length(); i++){
        int smallMask=0;
        int largeMask = 0;
        char ch = s[i];
        int chint = 0;
        if(ch>=65 && ch<=90){
          chint = ch-'A';
          largeMask = 1<<chint;</pre>
        }
        else{
          chint = ch-'a';
          smallMask = 1 << chint;
        for(int j = i+1; j < s.length(); j++){
          ch = s[i];
          if(ch>=65 && ch<=90){
             chint = ch-'A';
             largeMask |= 1<<chint;</pre>
          }
          else{
             chint = ch-'a';
             smallMask |= 1<<chint;
          }
          //checking for nice
          if((smallMask^largeMask) == 0){
             if(count<j-i+1){
               count = i-i+1;
                string temp(s.begin()+i,s.begin()+j+1);
               output = temp;
             }
          }
     return output;
```

# Output:



#### Aim:

### **Output:**



Test Case 1 Test Case 2

#### Aim:

Number of 1 bits

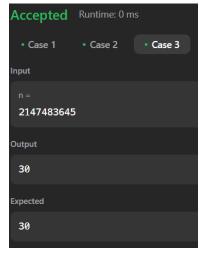
#### **Code:**

```
class Solution {
public:
    int hammingWeight(int n) {
        bitset<32> b(n);
    int ans=0;
        for(size_t i=0;i<b.size();i++){
        if(b[i]==1) ans++;
        }
        return ans;
    }
};</pre>
```

# **Output:**







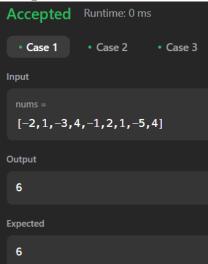
Case 1 Case 2

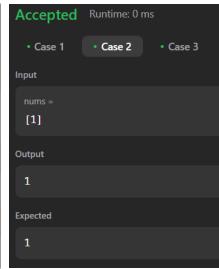
Case 3

# Aim: Max Subarray **Code:** class Solution { public: int maxSubArray(vector<int>& nums) { int res = nums[0]; int total = 0; for (int n : nums) { if (total < 0) { total = 0;total += n; res = max(res, total);} return res; }

# **Output:**

**}**;





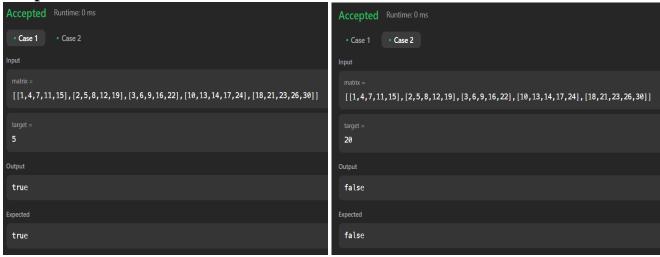


Case 1 Case 2

```
Aim:
Search 2d matrix 2
Code:
class Solution {
public:
  bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int n = matrix.size(), m = matrix[0].size();
    int row = 0, col = m - 1;

    while (row < n && col >= 0) {
        if (matrix[row][col] == target) return true;
        else if (matrix[row][col] < target) row++;
        else col--;
     }
     return false;
}
};</pre>
```

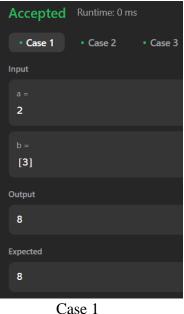
### **Output:**

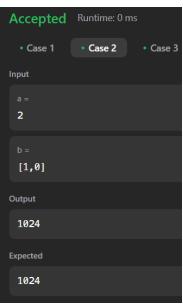


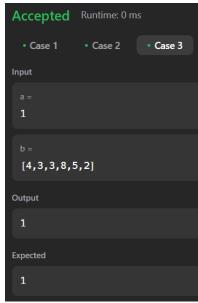
Case 1 Case 2

# Aim: Super Pow **Code:** class Solution { private: int solve(int base, int power, int mod) { int ans = 1; while (power > 0) { if (power & 1) { ans = (ans \* base) % mod; base = (base \* base) % mod; power >>= 1; } return ans; public: int superPow(int a, vector<int>& b) { a% = 1337;int n = b.size(); int m = 1140; int expi = 0; for(int i : b){ expi = (expi\*10+i)%m;} if $(\exp i == 0)$ { expi = m;return solve(a,expi,1337); **}**;

## **Output:**







e 1 Case 2

Case 3

```
Aim:
Beautiful Array
Code:
class Solution {
public:
  int partition(vector<int> &v, int start, int end, int mask)
     int j = start;
     for(int i = start; i \le end; i++)
        if((v[i] \& mask) != 0)
          swap(v[i], v[j]);
          j++;
     }
     return j;
  void sort(vector<int> & v, int start, int end, int mask)
  {
     if(start >= end) return;
     int mid = partition(v, start, end, mask);
     sort(v, start, mid - 1, mask << 1);
     sort(v, mid, end, mask << 1);
  }
  vector<int> beautifulArray(int N) {
     vector<int> ans;
     for(int i = 0; i < N; i++) ans.push_back(i + 1);
     sort(ans, 0, N - 1, 1);
     return ans;
```

# **Output:**

**}**;



Case 1



Case 2

# **Problem 8** Aim: The Skyline Problem Code: class Solution { public: vector<vector<int>>> getSkyline(vector<vector<int>>& buildings) { int edge\_idx = 0; vector<pair<int, int>> edges; priority\_queue<pair<int, int>> pq; vector<vector<int>> skyline; for (int i = 0; i < buildings.size(); ++i) { const auto &b = buildings[i]; edges.emplace\_back(b[0], i); edges.emplace\_back(b[1], i); std::sort(edges.begin(), edges.end()); while (edge\_idx < edges.size()) { int curr\_height; const auto &[curr\_x, \_] = edges[edge\_idx]; while (edge\_idx < edges.size() && curr\_x == edges[edge\_idx].first) { const auto &[\_, building\_idx] = edges[edge\_idx]; const auto &b = buildings[building\_idx]; if $(b[0] == curr_x)$ pq.emplace(b[2], b[1]); ++edge idx; while (!pq.empty() && pq.top().second <= curr\_x) pq.pop(); curr\_height = pq.empty() ? 0 : pq.top().first; if (skyline.empty() || skyline.back()[1] != curr\_height) skyline.push\_back({curr\_x, curr\_height});

**Output:** 

**}**;

}

return skyline;

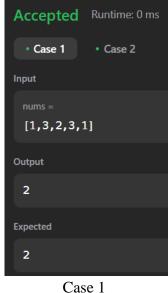


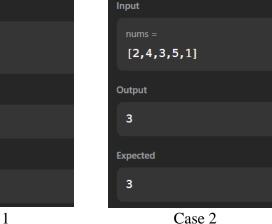
Case 1 Case 2

```
Aim:
Reverse Pairs
Code:
class SegTree {
private:
  int tree_size;
  vector<int> tree;
  void update(int lx, int rx, int ni, int idx) {
     if (rx - lx == 1) {
        tree[ni]++;
        return;
     }
     int m = (lx + rx) >> 1;
     if (idx < m)
        update(lx, m, ni * 2 + 1, idx);
     else
        update(m, rx, ni *2 + 2, idx);
     tree[ni] = tree[ni * 2 + 1] + tree[ni * 2 + 2];
  int query(int l, int r, int lx, int rx, int ni) {
     if (1 >= rx || r <= lx)
        return 0;
     if (1 \le lx \&\& r \ge rx)
        return tree[ni];
     int m = (lx + rx) >> 1;
     return query(l, r, lx, m, ni * 2 + 1) + query(l, r, m, rx, ni * 2 + 2);
   }
public:
  SegTree(int n) {
     tree\_size = 1;
     while (tree_size < n)
        tree_size <<= 1;
     tree = vector<int>(tree_size * 2);
   }
  void update(int idx) {
     update(0, tree_size, 0, idx);
  int query(int l, int r) {
```

```
return query(1, r + 1, 0, tree\_size, 0);
};
class Solution {
public:
  int reversePairs(vector<int>& nums) {
     int n = nums.size();
     set<long long> values;
     for(const auto& num : nums) {
       values.insert(num);
       values.insert(2LL * num);
     }
     int last index = 0;
     unordered_map<long long, int> values_indices;
     for(const auto& val : values)
       values_indices[val] = last_index++;
     SegTree seg_tree(last_index);
     int ans = 0;
     for(int i = 0; i < n; ++i) {
       ans += seg_tree.query(values_indices[2LL * nums[i]] + 1, last_index);
       seg_tree.update(values_indices[nums[i]]);
     }
     return ans;
};
```

### **Output:**





Accepted

• Case 1

Case 2

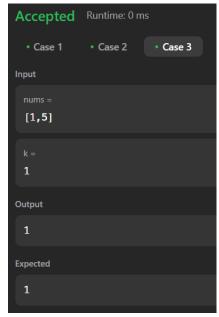
#### Aim:

```
Longest increasing subsequence 2
Code:
class Solution {
public:
  vector<int>tree;
  void update(int node,int st,int end,int i,int val){
     if(st==end)
       tree[node]=max(tree[node],val);
       return;
     int mid=(st+end)/2;
     if(i \le mid)
       update(node*2,st,mid,i,val);
     }else{
       update(node*2+1,mid+1,end,i,val);
     tree[node]=max(tree[node*2],tree[node*2+1]);
  int query(int node,int st,int end,int x,int y){
     if(x>end \parallel y<st) return -1e9;
     if(st >= x && end <= y)
       return tree[node];
     int mid=(st+end)/2;
     int left=query(2*node,st,mid,x,y);
     int right=query(2*node+1,mid+1,end,x,y);
     return max(left,right);
  int lengthOfLIS(vector<int>& nums, int k) {
     int n=nums.size();
     if(n==1) return 1;
     int m=*max_element(nums.begin(),nums.end());
     tree.clear();
     tree.resize(4*m+10);
     for(int i=n-1;i>=0;i--)
       int l=nums[i]+1,r=min(nums[i]+k,m);
       int x=query(1,0,m,l,r);
       if(x==-1e9) x=0;
       update(1,0,m,nums[i],x+1);
     return tree[1];
};
```

# **Output:**







Case 1 Case 2 Case 3