Name: Aryan Kumar Singh
Branch : BE_CSE
UID: 22BCS13577
Section: IOT_615_B
Date:24/12/2024

DAY 5 (Tuesday)

Q1.Search in 2D Matrix.

You are given an m x n integer matrix matrix with the following two properties:

Each row is sorted in non-decreasing order.

The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true if target is in matrix or false otherwise.

You must write a solution in O(log(m * n)) time complexity.

ANSWER:

```
    #include <iostream> #include <vector> using namespace std;

bool searchMatrix(vector<vector<int>>& matrix, int target) { if

(matrix.empty() || matrix[0].empty()) {

return false;

}

int m = matrix.size(); int n =

matrix[0].size(); int left = 0, right

= m * n - 1; while (left <= right) {

int mid = left + (right - left) / 2;

int midValue = matrix[mid / n][mid % n]; if

(midValue == target) { return true;

} else if (midValue < target) { left = mid + 1;
} else {
right = mid - 1;
}
}

return false;
}

int main() {
vector<vector<int>> matrix = {
{1, 3, 5, 7},
{10, 11, 16, 20},
{23, 30, 34, 60}
};
int target = 3;

if (searchMatrix(matrix, target)) { cout <<
   "true" << endl;
   } else {
   cout << "false" << endl;
   }
   return 0;
   }
```

OUTPUT:

```
true
```

Q2.Find First and Last Position of Element in Sorted Array.

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

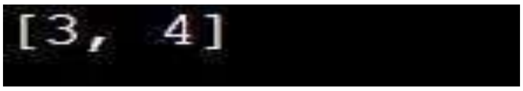
If target is not found in the array, return [-1, -1].

You must write an algorithm with O(log n) runtime complexity.

ANSWER:

```
#include <iostream> #include

<vector>  using  namespace

std;



vector<int> searchRange(vector<int>& nums, int target) { vector<int> result = {-

1, -1};



auto findBound = [&](bool findFirst) { int left = 0,

right = nums.size() - 1; int bound = -1;


while (left <= right) {
int mid = left + (right - left) / 2;



if (nums[mid] == target) { bound = mid;

if (findFirst) { right = mid - 1;

} else {
left = mid + 1;

}
} else if (nums[mid] < target) { left = mid + 1;

    } else {
right = mid - 1;

    }

    }



    return bound;

    };



    result[0]  =  findBound(true);  if  (result[0]

    != -1) {

    result[1] = findBound(false);

    }



    return result;

    }


      int main() {

      vector<int> nums = {5, 7, 7, 8, 8, 10};
      int target = 8;
vector<int> result = searchRange(nums, target);

      cout << "[" << result[0] << ", " << result[1] << "]" << endl; return 0;

      }
```

OUTPUT:



```
[3, 4]
```

Q3. Find Minimum in Rotated Sorted Array.

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- [4,5,6,7,0,1,2] if it was rotated 4 times.
- [0,1,2,4,5,6,7] if it was rotated 7 times.
-

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n- 1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of unique elements, return the minimum element of this array.

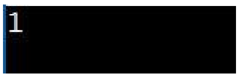
You must write an algorithm that runs in O(log n) time.

ANSWER:

```
 #include  <iostream>

 #include     <vector>

 using namespace std;



int  findMin(vector<int>&  nums)  {  int

 left = 0, right = nums.size() - 1; while

 (left < right) {


        int mid = left + (right - left) / 2;



if (nums[mid] > nums[right]) { left = mid

 + 1;

 } else {
```

```
right = mid;
}
}

return nums[left];
}

int main() {
vector<int> nums = {3, 4, 5, 1, 2};
cout << findMin(nums) << endl;
return 0;
}
```

OUTPUT:



```
1
```

Q4. Sort Items by Groups Respecting Dependencies

There are n items each belonging to zero or one of m groups where group[i] is the group that the i-th item belongs to and it's equal to -1 if the i-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it.

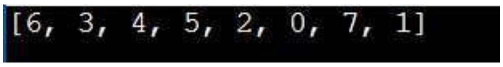
Return a sorted list of the items such that:

The items that belong to the same group are next to each other in the sorted list. There are some relations between these items where beforeItems[i] is a list containing all the items that should come before the i-th item in the sorted array (to the left of the i-th item).
Return any solution if there is more than one solution and return an empty list if there is no solution.

ANSWER:

```
#include <iostream>
#include <vector>
          #include <queue>
          #include <unordered_map> #include
          <unordered_set>
          using namespace std;

          vector<int> topologicalSort(int n, unordered_map<int, vector<int>>&
           graph, vector<int>& indegree) { queue<int> q; vector<int> order;
for (int i = 0; i < n; ++i) { if (indegree[i] == 0)
          {
          q.push(i);
          }
          }

        while (!q.empty()) { int node =
          q.front(); q.pop();
          order.push_back(node);

        for (int neighbor : graph[node]) { --
          indegree[neighbor]; if (indegree[neighbor] ==
          0) {
          q.push(neighbor);
          }
          }
          }

          if (order.size() == n) {
          return order;
          } return {};
          }

          vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>& beforeItems)
          { for (int i = 0; i < n; ++i) { if (group[i] == -1) {
          group[i] = m++;
          }
          }

          unordered_map<int, vector<int>> itemGraph; unordered_map<int,
          vector<int>> groupGraph;
          vector<int> itemIndegree(n, 0);
          vector<int> groupIndegree(m, 0);
for (int i = 0; i < n; ++i) { for (int before : beforeItems[i])
          { itemGraph[before].push_back(i);
          ++itemIndegree[i];

        if (group[before] != group[i]) { groupGraph[group[before]].push_back(group[i]);
          ++groupIndegree[group[i]];
          }
          }
        }

        vector<int> itemOrder = topologicalSort(n, itemGraph, itemIndegree); vector<int> groupOrder
        = topologicalSort(m, groupGraph, groupIndegree);

      if (itemOrder.empty() || groupOrder.empty()) { return {};
        }

        unordered_map<int, vector<int>> groupedItems; for (int
        item : itemOrder) {
        groupedItems[group[item]].push_back(item); }

        vector<int> result; for (int grp : groupOrder) { for
        (int item : groupedItems[grp]) {
        result.push_back(item);
        }
        }

        return result;
        }

      int main() { int n = 8, m = 2;
        vector<int> group = {-1, -1, 1, 0, 0, 1, 0, -1};
        vector<vector<int>> beforeItems = {{}, {6}, {5}, {6}, {3, 6}, {}, {}, {}};

        vector<int> result = sortItems(n, m, group, beforeItems); if
        (result.empty()) { cout << "[]" << endl;
      } else { cout << "[";
      for (size_t i = 0; i < result.size(); ++i) { cout << result[i];
        if (i < result.size() - 1) { cout << ", ";
        }
        }
        cout << "]" << endl;
        }
return 0;
        }
```

OUTPUT:



```
[6, 3, 4, 5, 2, 0, 7, 1]
```

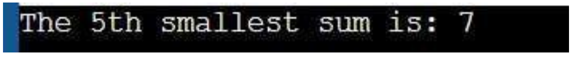
Q5. Find the Kth Smallest Sum of a Matrix With Sorted Rows.

ANSWER:

```
#include <iostream>
```

```
#include <vector>
#include    <queue>    #include
<set>  using  namespace  std;
struct Node
{ int sum; vector<int> indices;
bool operator>(const Node &other) const
{
return sum > other.sum;
} };
int kthSmallest(vector<vector<int>> &mat, int k) {
int m = mat.size(), n = mat[0].size();
priority_queue<Node, vector<Node>, greater<Node>> minHeap;
vector<int> initialIndices(m, 0); int
initialSum = 0; for (int i = 0; i < m; ++i)
{
initialSum += mat[i][0];
}
minHeap.push(Node{initialSum,                  initialIndices});
set<vector<int>> visited; visited.insert(initialIndices);
for (int count = 0; count < k - 1; ++count)
{
Node currentNode = minHeap.top(); minHeap.pop();
for (int i = 0; i < m; ++i)
{
if (currentNode.indices[i] + 1 < n)
{
vector<int> newIndices = currentNode.indices; newIndices[i]++;
if (visited.find(newIndices) == visited.end())
{
visited.insert(newIndices);
int newSum = currentNode.sum - mat[i][currentNode.indices[i]] + mat[i][newIndices[i]];
minHeap.push(Node{newSum, newIndices});
}
}
}
}
return minHeap.top().sum;
} int main() { vector<vector<int>> mat = {{1, 3, 11}, {2, 4,
6}}; int k = 5;
int result = kthSmallest(mat, k);
 cout << "The " << k << "th smallest sum is: " << result << endl; return 0;
 }
```

Output:



```
The 5th smallest sum is: 7
```

Q6. Merge k Sorted Lists.

```
}
```

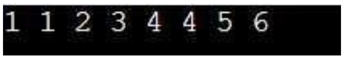
ANSWER:

```
#include <iostream>

#include           <vector>

#include <queue> using

namespace  std;  struct

ListNode

{

int val;

ListNode *next;

ListNode(int x) : val(x), next(nullptr) {}

};

struct compare

{

bool operator()(ListNode *a, ListNode *b)

{

return a->val > b->val;

}

};

ListNode *mergeKLists(vector<ListNode *> &lists)

{

priority_queue<ListNode *, vector<ListNode *>, compare> minHeap;

for (ListNode *list : lists)

{

if (list != nullptr)

{

minHeap.push(list);

}

}
```

```
ListNode *dummy = new ListNode(0);
ListNode *current = dummy; while
(!minHeap.empty())
{
ListNode *node = minHeap.top();
minHeap.pop(); current->next =
node; current = current->next; if
(node->next != nullptr)
{
minHeap.push(node->next);
}
}
return dummy->next;
}
void printList(ListNode *head)
{
while (head != nullptr)
{
cout << head->val << " "; head =
head->next;
}
cout << endl;
}
int main()
{
ListNode *l1 = new ListNode(1); l1->next =
new ListNode(4); l1->next->next = new
ListNode(5); ListNode *l2 = new
ListNode(1); l2->next = new ListNode(3); l2-
```

```
>next->next = new ListNode(4); ListNode
*l3 = new ListNode(2); l3->next = new
ListNode(6);
vector<ListNode *> lists = {l1, l2, l3}; ListNode
*mergedList          =          mergeKLists(lists);
printList(mergedList); return 0;
}
```

Output:



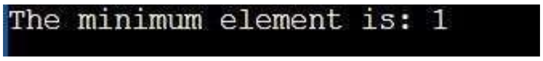
```
1 1 2 3 4 4 5 6
```

Q7. Find Minimum in Rotated Sorted Array II.

ANSWER:

```
#include     <iostream>     #include
<vector> using namespace std; int
findMin(vector<int> &nums)
{
int left = 0, right = nums.size() - 1; while
(left < right)
{
int mid = left + (right - left) / 2; if
(nums[mid] > nums[right])
{
left = mid + 1;
}
else if (nums[mid] < nums[right])
{ right = mid; } else
{ right--; } } return
nums[left];
    } int
    main() {
    vector<int> nums = {1, 3, 5};
    cout << "The minimum element is: " << findMin(nums) << endl; return
    0;
    }
```

Output:



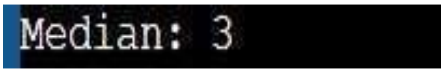
```
The minimum element is: 1
```

Q8. Median of Two Sorted Arrays.

ANSWER:

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;
double findMedianSortedArrays(vector<int> &nums1, vector<int> &nums2)
{ if (nums1.size() > nums2.size())
{
swap(nums1, nums2);
}
int m = nums1.size(); int
n = nums2.size(); int left
= 0, right = m; while (left
<= right)
{ int partition1 = left + (right - left) / 2; int
partition2 = (m + n + 1) / 2 - partition1;
int maxLeft1 = (partition1 == 0) ? INT_MIN : nums1[partition1 - 1]; int
minRight1 = (partition1 == m) ? INT_MAX : nums1[partition1]; int
maxLeft2 = (partition2 == 0) ? INT_MIN : nums2[partition2 - 1]; int
minRight2 = (partition2 == n) ? INT_MAX : nums2[partition2]; if
(maxLeft1 <= minRight2 && maxLeft2 <= minRight1)
{ }
if ((m + n) % 2 == 1)
{
return max(maxLeft1, maxLeft2);
} else
{
return (max(maxLeft1, maxLeft2) + min(minRight1, minRight2)) / 2.0;
}
if (maxLeft1 > minRight2)
{
right = partition1 - 1;
} else {
left = partition1 + 1;
} }
throw invalid_argument("Input arrays are not sorted");
} int main()
{
vector<int>  nums1  =  {1,  3};  vector<int>
nums2 = {2};
cout << "Median: " << findMedianSortedArrays(nums1, nums2) << endl; return 0;
}
```

Output:



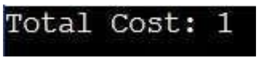
```
Median: 3
```

Q9. Create Sorted Array through Instructions.

ANSWER:

```
#include <iostream>
#include <vector> #include
<algorithm> using
namespace std; const int
MOD = 1e9 + 7; class
FenwickTree
{ public:
FenwickTree(int size) : bit(size + 1, 0) {} void
update(int index, int value)
{
for (; index < bit.size(); index += index & -index)
{ bit[index] += value;
} } int query(int
index)
{ int sum = 0;
for (; index > 0; index -= index & -index)
{ sum += bit[index];
} return sum; }
private:
vector<int> bit;
};
int createSortedArray(vector<int> &instructions)
{ int max_val = 100000; FenwickTree
fenwick(max_val); long long total_cost =
0; for (int i = 0; i < instructions.size(); ++i)
{
int current = instructions[i];
int less_than_current = fenwick.query(current - 1); int
greater_than_current = i - less_than_current; total_cost +=
min(less_than_current, greater_than_current); total_cost %=
MOD; fenwick.update(current, 1);
}
return total_cost;
} int main()
{
vector<int> instructions = {1, 5, 6, 2}; cout << "Total Cost: " <<
createSortedArray(instructions) << endl; // Output: 1 return 0;
}
```

Output:



```
Total Cost: 1
```

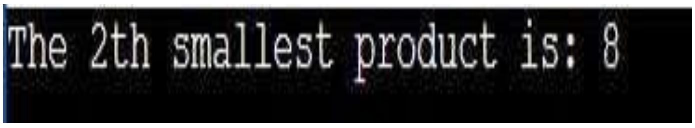
Q10. Kth Smallest Product of Two Sorted Arrays.

ANSWER:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
struct Product
  { int value, i,
  j;
  Product(int v, int x, int y) : value(v), i(x), j(y) {}
  bool operator>(const Product &other) const
  {
  return value > other.value;
  } };
```

```
int kthSmallestProduct(vector<int> &nums1, vector<int> &nums2, int k)
{
int m = nums1.size(), n = nums2.size();
priority_queue<Product, vector<Product>, greater<Product>> minHeap; for
(int j = 0; j < n; ++j)
{
}
minHeap.push(Product(nums1[0] * nums2[j], 0, j)); for
(int count = 1; count < k; ++count)
{
Product p = minHeap.top(); minHeap.pop();
int i = p.i, j = p.j; if
(i + 1 < m)
{
minHeap.push(Product(nums1[i + 1] * nums2[j], i + 1, j));
} }
return minHeap.top().value;
} int
main() {
vector<int> nums1 = {2, 5};
vector<int> nums2 = {3, 4};
int k = 2;
int result = kthSmallestProduct(nums1, nums2, k); cout << "The "
<< k << "th smallest product is: " << result << endl; return 0;
}
```

Output:



```
The 2th smallest product is: 8
```