

DAY 5

Ques 1. Given an integer k and array arr. Your task is to return the position of the first occurrence of k in the given array and if element k is not present in the array then return -1.

Program code:

```
#include <iostream> #include <vector> using
namespace std; int findFirstOccurrence(int k, const
vector<int>& arr) {
    for (int i = 0; i < arr.size(); i++)
    { if (arr[i] == k) {      return i + 1;
        }    }
    return -1;
} int main() {    int k = 16;
vector<int> arr = {9, 7, 16, 16, 4};
```

```
    int result = findFirstOccurrence(k, arr);
cout << result << endl; // Output: 3

    return 0;
}
```

Output:

Output

3

=== Code Execution Successful ===

Ques 2. 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.

Program code:

```
#include <iostream> #include <vector> using namespace std;
vector<int> searchRange(const vector<int>& nums, int target)
{ auto findBound = [&](bool isFirst) -> int {    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 (isFirst)
{ right = mid - 1;
        } else
{ left = mid + 1;
        }
    }
```

```

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

};

int start = findBound(true);
int end = findBound(false);
return {start, end};
} 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:



The screenshot shows a terminal window with a title bar labeled "Output". The output text is "[3, 4]" on the first line. Below it, on a second line, is the message "=== Code Execution Successful ===" followed by a cursor. The background of the terminal is light gray.

Ques 3 You are given an integer array `arr[]`. Your task is to find the smallest positive number missing from the array.

Program Code:

```
#include <iostream> #include <vector> using
namespace std; int
findSmallestMissingPositive(vector<int>& arr) {
    int n = arr.size();    for (int i = 0; i < n; i++) {        while (arr[i]
> 0 && arr[i] <= n && arr[arr[i] - 1] != arr[i]) {
        swap(arr[i], arr[arr[i] - 1]);
    }
}
    for (int i = 0; i < n; i++) {
if (arr[i] != i + 1) return i + 1;
    }
    return n + 1;
}

int main() {    vector<int> arr = {2, -3, 4, 1, 1, 7};
cout << findSmallestMissingPositive(arr) << endl;
    return 0;
}
```

Output:

```
Output
3

=== Code Execution Successful ===
```

Ques 4. You are given an $m \times n$ matrix `mat` that has its rows sorted in nondecreasing order and an integer `k`. You are allowed to choose exactly one element from each row to form an array. Return the k th smallest array sum among all possible arrays.

Program Code:

```
#include <iostream>

#include <vector> #include <queue> using
namespace std; int
kthSmallest(vector<vector<int>>& mat, int k)
{ priority_queue<int, vector<int>, greater<>>
minHeap; minHeap.push(0); for (const auto&
row : mat) { priority_queue<int, vector<int>,
greater<>> nextHeap; while
(!minHeap.empty()) { int sum =
minHeap.top(); minHeap.pop(); for
(int num : row) { nextHeap.push(sum +
num); if (nextHeap.size() > k)
nextHeap.pop();
}
}
minHeap.swap(nextHeap);
}
return minHeap.top();
} int main() { vector<vector<int>> mat = {{1, 3,
11}, {2, 4, 6}};
int k = 5;
```

```

    cout << kthSmallest(mat, k) << endl;
return 0;
}

```

Output:



```

Output
7

=== Code Execution Successful ===

```

Ques 5. You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Program Code:

```

#include <iostream>
#include <vector>
#include <queue> using
namespace std; struct
ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};
class Solution { public:

```

```

ListNode* mergeKLists(vector<ListNode*>& lists) {      auto compare =
[](ListNode* a, ListNode* b) { return a->val > b->val; };

    priority_queue<ListNode*, vector<ListNode*>, decltype(compare)>
minHeap(compare);

    for (ListNode* list : lists)
{ if (list) minHeap.push(list);    }

    ListNode dummy(0);      ListNode* current =
&dummy;      while (!minHeap.empty())
{ ListNode* node = minHeap.top();
minHeap.pop();      current->next = node;
current = current->next;      if (node->next)
minHeap.push(node->next);
    }
    return dummy.next;
}
};

```

```

int main() {
    ListNode* l1 = new ListNode(1, new ListNode(4, new ListNode(5)));
    ListNode* l2 = new ListNode(1, new ListNode(3, new ListNode(4)));
    ListNode* l3 = new ListNode(2, new ListNode(6));    vector<ListNode*>
lists = {l1, l2, l3};
}

```

Solution solution;

ListNode* result = solution.mergeKLists(lists);

```
    while (result) {    cout <<  
result->val << " ";    result =  
result->next;  
    }  
    return 0;  
}
```

Output:

```
Output  
1 1 2 3 4 4 5 6  
  
=== Code Execution Successful ===
```