CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

DOMAIN WINTER CAMP

Department of Computer Science and Engineering

Name:- Aditya Sharma   UID:- 22BCS10722   Section:- 22KPIT-901-B

DAY-2

Q.1. Majority Elements

Given an array nums of size n, return the majority element.
The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Program Code:-

```cpp
int majorityElement(vector<int>&
    nums){ int count=0;
    int el;
    for(int
        i=0;i<nums.size();i++){ if(c
        ount==0){
            count=1;
            el=nums[i];
        }
        else
            if(nums[i]==el){ count++;
            }
        else{
            count--;}
    }
    int count1 = 0;
    for(int i=0;
        nums.size();i++){ if(nums[i]
        ==el){
            count1++;}
    }
    if(count1 > (nums.size() /
        2)){ return el;}
    return -1;
```

```
        }

    };
```

Output:-

Input

nums =
[3,2,3]

Output

3

Q.2. Pascal's Triangle

Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Program Code:-

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

vector<vector<int>> generate(int numRows)
  { vector<vector<int>> result(numRows);
  for (int i = 0; i < numRows; ++i)
    { result[i].resize(i + 1, 1); // Initialize row with
    1s for (int j = 1; j < i; ++j)
      result[i][j] = result[i - 1][j - 1] + result[i - 1][j];
  }
  return result;
}

int main() {
  int numRows = 5;
  auto pascal = generate(numRows);
  cout << "[\n";
  for (const auto& row : pascal)
    { cout << " [";
    for (int i = 0; i < row.size(); ++i)
      cout << row[i] << (i < row.size() - 1 ? "," : "");
    cout << "]\n";
  }
```
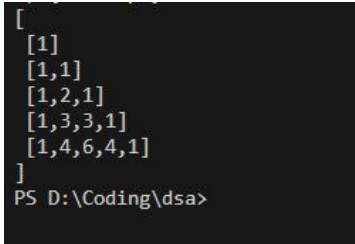
```
        cout << "]\n";
        return 0;
    }
```

Output:-


```
[
 [1]
 [1,1]
 [1,2,1]
 [1,3,3,1]
 [1,4,6,4,1]
]
PS D:\Coding\dsa>
```

Q.3. Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.

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

struct ListNode
    { int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};
ListNode* removeElements(ListNode* head, int val)
    { ListNode dummy(0);
    dummy.next = head;
    ListNode* current = &dummy;
    while (current->next) {
        if (current->next->val == val)
            current->next = current->next->next;
        else
            current = current->next;
    }
    return dummy.next;
}

void printList(ListNode* head)
    { while (head) {
        cout << head->val << " ";
        head = head->next;
```

```cpp
    }
    cout << endl;
}

int main()
    { int n, val;
    cout << "Enter the number of nodes: ";
    cin >> n;
    ListNode *head = nullptr, *tail = nullptr;
    cout << "Enter the elements of the linked list: ";
    while (n--) {
        int nodeVal;
        cin >> nodeVal;
        ListNode* newNode = new ListNode(nodeVal);
        if (!head)
            head = tail = newNode;
        else
            tail = tail->next = newNode;
    }
    cout << "Enter the value to remove: ";
    cin >> val;
    head = removeElements(head, val);
    cout << "Linked list after removal: ";
    printList(head);
    return 0;
}
```

Output:-

```
Enter the number of nodes: 5
Enter the elements of the linked list: 8 5 10 6 1
Enter the value to remove: 6
Linked list after removal: 8 5 10 1
PS D:\Coding\dsa>
```

Q.4. Baseball Game :

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the ith operation you must apply to the record and is one of the following:

An integer x.
Record a new score of x.
'+'.
Record a new score that is the sum of the previous two scores.
'D'.
Record a new score that is the double of the previous score.
'C'.
Invalidate the previous score, removing it from the record.
Return the sum of all the scores on the record after applying all the operations.

The test cases are generated such that the answer and all intermediate calculations fit in a 32-bit integer and that all operations are valid.

Program Code:-

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <bits/stdc++.h>

using namespace std;

int calPoints(vector<string>& ops)
    { vector<int> record;
    for (const string& op : ops)
        { if (op == "+")
            record.push_back(record.back() + record[record.size() - 2]);
        else if (op == "D")
            record.push_back(2 * record.back());
        else if (op == "C")
            record.pop_back();
        else
            record.push_back(stoi(op));
    }
    return accumulate(record.begin(), record.end(), 0);
}

int main() {
    vector<string> ops = {"5", "2", "C", "D", "+"};
    cout << calPoints(ops) << endl; // Output: 30
    return 0;
}
```

Output:-

```
30
PS D:\Coding\dsa> []
```

Q.5. **You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete the ith job.There are k workers that you canassign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that themaximum working time of any worker is minimized. Return the minimum possible maximum working time**

Program Code:-

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
using namespace std;

bool canAssign(const vector<int>& jobs, vector<int>& workers, int maxTime, int index) {
    if (index == jobs.size()) return true;
    for (int i = 0; i < workers.size(); ++i) {
        if (workers[i] + jobs[index] <= maxTime) { workers[i]
            += jobs[index];
            if (canAssign(jobs, workers, maxTime, index + 1)) return true; workers[i]
            -= jobs[index];
        }
        if (workers[i] == 0) break;
    }
    return false;
}

int minimumTimeRequired(vector<int>& jobs, int k)
    { int left = *max_element(jobs.begin(), jobs.end());
    int right = accumulate(jobs.begin(), jobs.end(), 0);
    sort(jobs.rbegin(), jobs.rend());
```

```cpp
    while (left < right) {
        int mid = left + (right - left) / 2;
        vector<int> workers(k, 0);
        if (canAssign(jobs, workers, mid, 0)) right = mid;
        else left = mid + 1;
    }
    return left;
}

int main() {
    vector<int> jobs = {3, 2, 3};
    int k = 3;
    cout << "Minimum possible maximum working time: " <<
minimumTimeRequired(jobs, k) << endl;
    return 0;
}
```

Output:-

```
Minimum possible maximum working time: 3
PS D:\Coding\dsa>
```