Winter Domain Camp Day 2

Name: Divya Sharma **UID: 22BCS14192**
Branch: BE-CSE **Section:**22BCS_IOT_603-B
Semester: 5th **Date of Performance:**20thDecember,2024

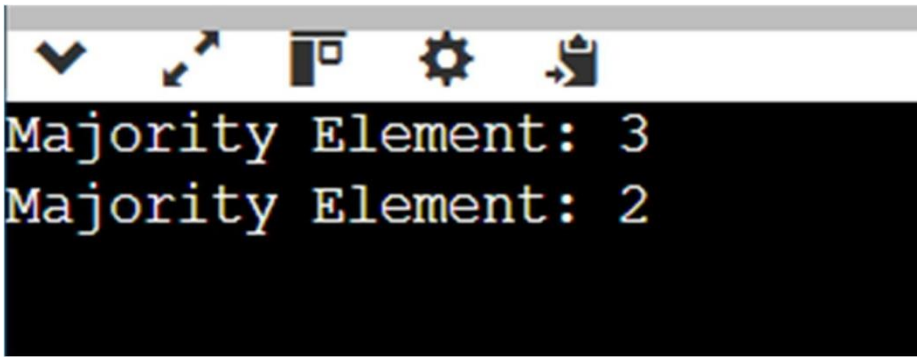
Problem1: Majority Elements
Given an array nums of size n, return the majority element. The majority elementistheelementthatappears morethan ⌊ n/2⌋ times.Youmayassume that the majority element always exists in the array.

Solution:

```
#include<iostream>
#include <vector>

intmajorityElement(conststd::vector<int>&nums){
int candidate = 0, count = 0;
//Phase1:Findthecandidate for
(int num : nums) {
if (count == 0) {
   candidate=num;
   count = 1;
}elseif(num==candidate){
   count++;
} else{
   count--;
}
}

//Phase2:Verifythecandidate(Optionalsinceproblemguaranteesmajority element)
count =0;
for (int num : nums) {
if(num==candidate){
count++;
   }
```

```
    }

    if(count>nums.size()/2){ return
        candidate;
    }

    throwstd::runtime_error("Nomajorityelementfound.");
}

intmain(){
    std::vector<int>nums={2,2,1,1,1,2,2}; try {
        std::cout<<"MajorityElement: "<<majorityElement(nums)<<std::endl;
    } catch (const std::exception& e) {
        std::cerr<<e.what()<<std::endl;
    }
    return0;
}
```

Output:



Problem2: Pascal'sTriangle

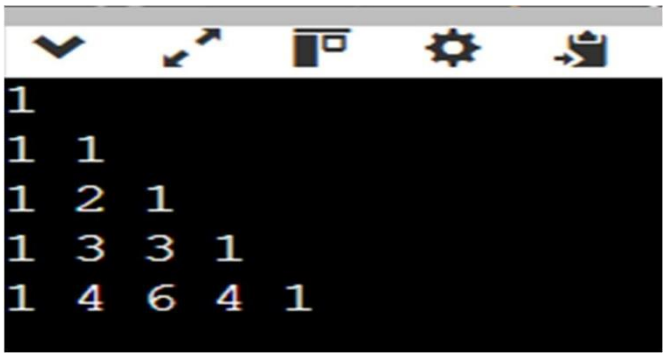
Given an integer numRows, return the first numRows of Pascal's triangle. In Pascal'striangle,eachnumberisthesumofthetwonumbersdirectlyaboveit

Solution:

```
#include<iostream>
#include <vector>
std::vector<std::vector<int>>generate(intnumRows){
    std::vector<std::vector<int>> triangle(numRows);
```

```
    for(inti=0;i<numRows;++i){
        triangle[i].resize(i + 1, 1);
        for(intj=1; j<i;++j){
            triangle[i][j]=triangle[i-1][j-1]+triangle[i -1][j];
        }}
    returntriangle;}
int main() {
    intnumRows=5;//Exampleinput auto
    result = generate(numRows); for
    (const auto& row : result) {
        for(intnum:row)std::cout <<num<<"";
        std::cout <<"\n";}
    return0;}
```

Output:



Problem3:SingleNumber

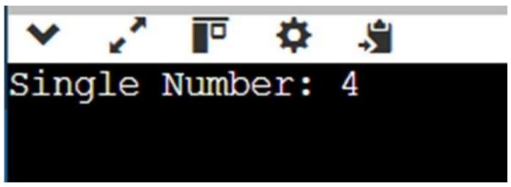
Given anon-emptyarrayofintegersnums,everyelementappearstwiceexcept for one. Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space.

Solution:

```
#include<iostream>
#include <vector>

intsingleNumber(conststd::vector<int>&nums){
    int result = 0;
    for(intnum:nums){ result
        ^= num;
    }
    returnresult;
```

```
}

intmain(){
   std::vector<int>nums={4,1,2,1,2};//Exampleinput
   std::cout<<"SingleNumber:"<<singleNumber(nums)<<std::endl;
   return 0;
}
```

Output:



Problem4: MergeTwoSortedLists

You are given the heads of two sorted linked lists list1 and list2. Merge the two listsintoonesortedlist.Thelistshouldbemadebysplicingtogetherthenodesof the first two lists. Return the head of the merged linked list.

Solution:

```
#include<iostream>

struct ListNode {
   int val;
   ListNode*next;
   ListNode(intx):val(x),next(nullptr){}
};

ListNode*mergeTwoLists(ListNode*list1,ListNode*list2){ if
   (!list1) return list2;
   if(!list2)return list1;
   if(list1->val <list2->val){
      list1->next=mergeTwoLists(list1->next,list2); return
      list1;
   }else{
```

```
        list2->next=mergeTwoLists(list1,list2->next); return
        list2;
    }
}

voidprintList(ListNode*head){
    while (head) {
        std::cout<<head->val<<"";
        head = head->next;
    }
}

intmain(){
    ListNode*list1=newListNode(1);
    list1->next = new ListNode(2);
    list1->next->next=newListNode(4);

    ListNode*list2=newListNode(1);
    list2->next = new ListNode(3);
    list2->next->next=newListNode(4);

    ListNode*mergedList=mergeTwoLists(list1,list2);
    printList(mergedList);

    return0;
}
```

Output:

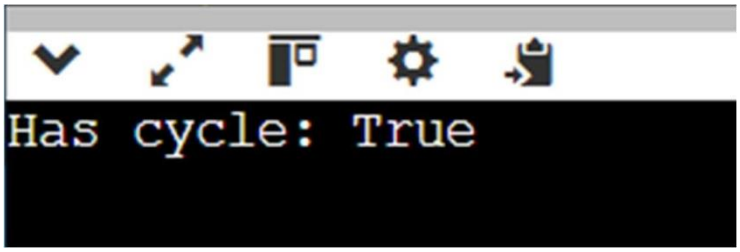
Problem5: LinkedListCycle.
Given head, the head of a linked list, determine if the linked list has a cycle in it. Thereisacycleinalinked listifthereissomenodeinthelistthatcanbereached againbycontinuouslyfollowingthenextpointer.Internally,posisusedtodenote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter. Return true if there is a cycle in the linked list. Otherwise, return false.

Solution:

```
#include<iostream>

struct ListNode {
   int val;
   ListNode*next;
   ListNode(intx):val(x),next(nullptr){}
};

boolhasCycle(ListNode*head){ if
   (!head) return false;
   ListNode*slow=head, *fast =head;

   while(fast&&fast->next){
      slow = slow->next;          // Move slow pointer by1 step
      fast=fast->next->next;       //Movefastpointerby2steps

      if (slow == fast) {          //Cycledetected
         return true;}}
   returnfalse;//Nocycle} int
main() {
   //Example1:Creatingacycleinthelist
   ListNode* head = new ListNode(3);
   head->next = new ListNode(2);
   head->next->next=newListNode(0);
   head->next->next->next=newListNode(-4);
   head->next->next->next->next=head->next;//Cyclestartsatnodewith value 2
```

```
    std::cout<<"Hascycle:"<<(hasCycle(head)?"True":"False")<<std::endl;

    return0;
}
```

Output:

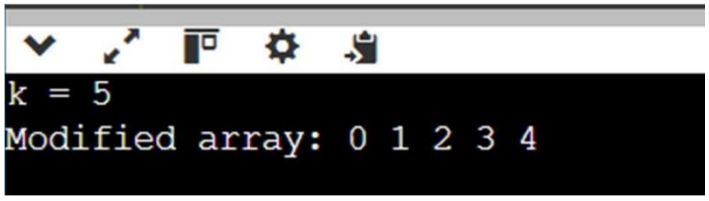


Problem6: RemoveElement
Givenanintegerarraynumssortedinnon-decreasingorder,removetheduplicates in-placesuchthateachuniqueelementappearsonlyonce.Therelativeorderofthe elements shouldbekeptthesame.Thenreturnthe numberofunique elementsin nums.Consider thenumber ofuniqueelementsofnumstobek,to getaccepted, youneedtodothefollowing things: Changethearraynums such thatthefirstk elements of nums containthe uniqueelementsintheordertheywerepresentin numsinitially.Theremainingelements of nums are notimportantaswellasthe size of nums. Return k.

Solution:

```
#include<iostream>
#include <vector>

intremoveDuplicates(std::vector<int>&nums){ if
   (nums.empty()) return 0;

   intk=1;//Pointerforthenextuniqueelement for (int i
   = 1; i < nums.size(); ++i) {
      if(nums[i]!=nums[i-1]){
         nums[k++]=nums[i];//Movetheuniqueelementtothefront}} return k;//
   Number of unique elements}
intmain(){
   std::vector<int>nums={0,0,1,1,1,2,2,3,3,4}; int
   k = removeDuplicates(nums);

   std::cout<<"k="<<k<<std::endl;
   std::cout <<"Modified array: ";
   for (int i = 0; i < k; ++i) {
      std::cout<<nums[i]<<"";
   }
   std::cout<<std::endl;
   return 0;}
```

Output:

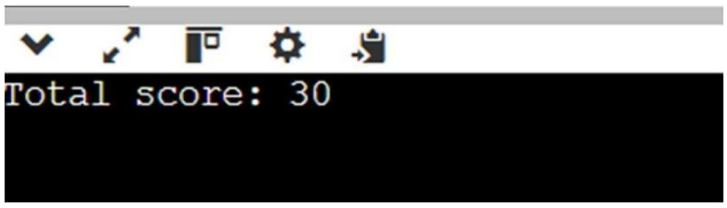


Problem7: BaseballGame:
You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record. You are given a list of stringsoperations,whereoperations[i]istheithoperationyoumustapplytothe record and is one of the following: An integer x. Record a new score of x. '+'. Recordanewscorethatisthe sumof theprevious two scores.'D'. Record anew 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 applyingalltheoperations.Thetestcasesaregeneratedsuchthattheanswerand all intermediate calculations fit in a 32-bit integer and that all operations are valid.

Solution:

```
#include<iostream>
#include <vector>
#include <string>

intcalPoints(std::vector<std::string>&ops){
   std::vector<int> record;
   for(constauto&op:ops){ if
      (op == "C") {
         record.pop_back();//Removethelastscore
      }else if(op=="D"){
         record.push_back(2*record.back());// Doublethelastscore
      }else if(op=="+"){
         record.push_back(record[record.size()-1]+record[record.size()-2]);// Sum
of last two scores
      } else{
         record.push_back(std::stoi(op));//Addtheintegerscore
      }
   }

   int total=0;
   for(intscore:record){ total
      += score;
   }

   returntotal;
}

intmain(){
   std::vector<std::string>ops ={"5","2","C","D","+"};
```

```
    std::cout<<"Totalscore:"<<calPoints(ops)<<std::endl;//Output:30 return 0;
}Output:
```



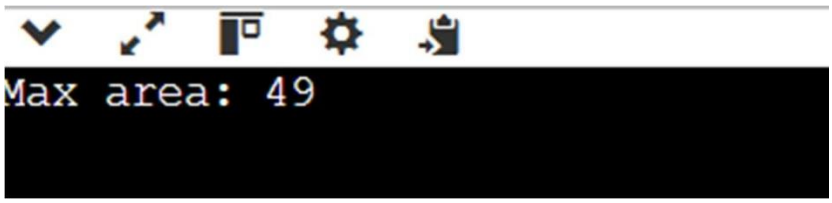
Problem8: ContainerWithMostWater
You are given an integer array height of length n. There are n vertical lines drawnsuchthatthetwoendpointsoftheithlineare(i,0)and(i,height[i]).Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store.

Solution:

```
#include<iostream>
#include <vector>

intmaxArea(std::vector<int>&height){
    intleft= 0,right =height.size() -1,maxArea= 0;

    while(left<right){
        intwidth=right- left;
        inth=std::min(height[left],height[right]);
        maxArea=std::max(maxArea,width*h);

        if(height[left]<height[right]) {
            ++left;
        } else{
            --right;
        }
    }

    returnmaxArea;
}
```

```
intmain(){
   std::vector<int>height={1,8,6,2,5,4,8,3,7};
   std::cout<<"Maxarea:"<<maxArea(height)<<std::endl;//Output:49 return
   0;
}
```

Output:



Problem9: JumpGameII

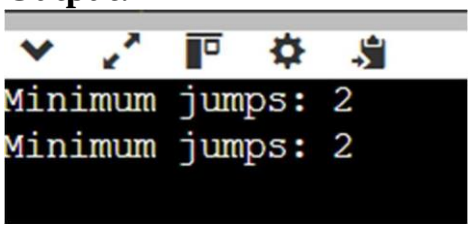
You are given a 0-indexed array of integers nums of length n. You are initially positionedatnums[0].Eachelementnums[i]representsthemaximumlengthofa forwardjumpfromindexi.Inotherwords,ifyouareatnums[i],youcanjumpto any nums[i + j] where: 0 <= j <= nums[i] and i + j < n Return the minimum number of jumps to reachnums[n -1].The test cases aregenerated such that you can reach nums[n - 1].

Solution:

```
#include<iostream>
#include <vector>

intjump(std::vector<int>&nums){ int
   n = nums.size();
   intjumps=0,farthest =0,currentEnd=0;

   for (int i= 0; i <n-1; ++i){
      farthest =std::max(farthest,i+nums[i]);//Updatethefarthestpointwecan reach

      if(i == currentEnd){
         jumps++;          //Wemakeajumpwhenwereachthecurrentend
         currentEnd = farthest;// Move to the farthest point we can reach

         if(currentEnd>= n - 1)break;// Ifwecan reachtheend,stop
      }
   }
```

```
    returnjumps;
}

intmain(){
    std::vector<int>nums1= {2,3,1,1,4};
    std::cout<<"Minimumjumps:"<<jump(nums1)<<std::endl;//Output:2

    std::vector<int>nums2= {2,3,0,1,4};
    std::cout<<"Minimumjumps:"<<jump(nums2)<<std::endl;//Output:2

    return0;
}
```

Output:



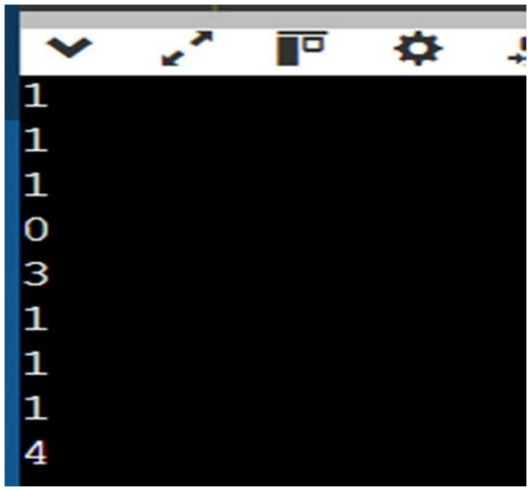
Problem10: DesignCircularQueue
Calculatethesumofthedigitsofagivennumbern.Forexample,forthenumber 12345, the sum of the digits is 1+2+3+4+5=15. To solve this, you will need to extract each digit from the number and calculate the total sum.

Solution:

```
#include<iostream>
#include <vector>
classMyCircularQueue{
private:
    std::vector<int>queue;
    intfront,rear,size,capacity;
public:
    MyCircularQueue(intk):queue(k),front(-1),rear(-1),size(0),capacity(k){} bool
    enQueue(int value) {
        if(size==capacity)returnfalse; if
        (size == 0) front = 0;
        rear=(rear+1)%capacity;
        queue[rear] = value;size++;
        returntrue;}
```

```
    bool deQueue() {
        if (size ==0) returnfalse;
        if (front == rear) front = rear = -1;
        elsefront=(front+1)%capacity;
        size--;
        returntrue;}
    int Front() {
        returnsize==0?-1:queue[front];} int
    Rear() {
        returnsize==0?-1:queue[rear];} bool
    isEmpty() {
        returnsize==0;}
    bool isFull() {
        returnsize==capacity;}}; int
main() {
    MyCircularQueueq(3);
    std::cout << q.enQueue(1) << std::endl;// True
    std::cout << q.enQueue(2) << std::endl;// True
    std::cout << q.enQueue(3) << std::endl;// True
    std::cout<<q.enQueue(4)<<std::endl;//False(full)
    std::cout << q.Rear() << std::endl;      // 3
    std::cout << q.isFull() << std::endl;    // True
    std::cout << q.deQueue() << std::endl;   // True
    std::cout<<q.enQueue(4)<<std::endl;//True
    std::cout << q.Rear() << std::endl;      // 4return
    0;}
```

Output:



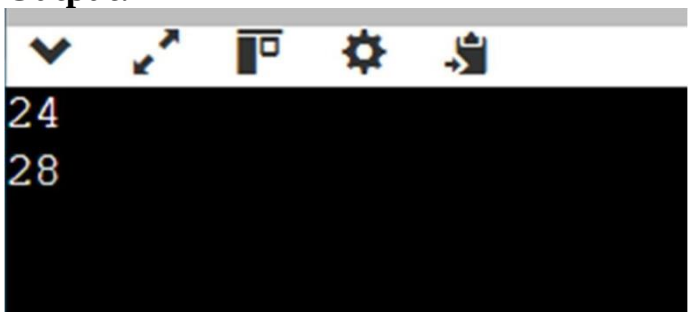
Problem11: CherryPickup II
You are given a rows x cols matrix grid representing a field of cherries where grid[i][j] represents the number of cherries that you can collect from the (i, j) cell.Youhavetworobotsthatcancollectcherriesforyou:Robot#1islocatedat thetop-leftcorner(0,0),and Robot#2 is located atthetop-right corner (0, cols - 1). Return the maximum number of cherries collection using both robots by following the rules below: Fromacell (i, j),robots can move to cell (i +1, j- 1), (i + 1, j), or (i + 1, j + 1). When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell. When both robots stay in the same cell, only one takes the cherries. Both robots cannot move outside of the grid at any moment. Both robots should reach the bottom row in grid.

Solution:

```
#include <iostream>
#include <vector>
usingnamespacestd;

classSolution{
public:
   intcherryPickup(vector<vector<int>>&grid){
      int rows = grid.size(), cols = grid[0].size();
      vector<vector<vector<int>>>dp(rows,vector<vector<int>>(cols,
vector<int>(cols, -1)));
      returndfs(0,0, cols-1,grid,dp);
   }

private:
   intdfs(introw,intcol1,intcol2,vector<vector<int>>&grid,
vector<vector<vector<int>>>& dp) {
      int rows =grid.size(),cols=grid[0].size();
      if(col1<0 ||col2<0 ||col1>=cols||col2>=cols)return0; if
      (dp[row][col1][col2] != -1) return dp[row][col1][col2];

      intcherries=grid[row][col1];
      if(col1!=col2)cherries+=grid[row][col2]; if
      (row < rows - 1) {
         intmaxCherries=0;
         for(int d1=-1;d1 <=1;d1++){
```

```
                for(intd2 =-1;d2 <=1; d2++){
                    maxCherries=max(maxCherries,dfs(row+1,col1+d1,col2+d2,
grid,dp));
                }
            }
            cherries += maxCherries;
        }
        returndp[row][col1][col2]= cherries;
    }
};

int main() {
    Solutionsol;
    vector<vector<int>>grid1={{3,1,1},{2,5,1},{1,5,5},{2,1,1}};
    cout << sol.cherryPickup(grid1) << endl; // Output: 24

    vector<vector<int>>grid2=
{{1,0,0,0,0,0,1},{2,0,0,0,0,3,0},{2,0,9,0,0,0,0},{0,3,0,5,4,0,0},{1,0,2,3,0,0,6}};
    cout<<sol.cherryPickup(grid2)<<endl;//Output:28
    return 0;
}
```

Output:



Problem 12: Maximum Number of Darts Inside of a Circular Dartboard

Aliceisthrowing ndartsonaverylargewall.You aregivenanarraydarts wheredarts[i]=[xi,yi]is thepositionoftheithdartthatAlicethrewonthe wall. Bobknows thepositionsofthendartsonthewall.Hewantstoplacea dartboard ofradius ron the wall so that themaximumnumber of dartsthat Alicethrowslie on the dartboard. Given the integer r, return the maximum number of darts that can lie on the dartboard.

Solution:

```
#include <iostream>
#include <vector>
#include <cmath>
usingnamespacestd;

classSolution{
public:
   intnumPoints(vector<vector<int>>&darts,intr){ int
      maxCount = 1, n = darts.size();
      for(inti = 0;i <n; ++i){
         for(int j= i + 1; j<n; ++j){
            doubledx=darts[j][0]-darts[i][0],dy=darts[j][1]-darts[i][1]; double
            dist = sqrt(dx * dx + dy * dy);
            if (dist>2* r)continue;
            doublemidX=(darts[i][0]+darts[j][0])/2.0,midY=(darts[i][1]+ darts[j][1]) /
2.0;
            double angle =sqrt(r* r-(dist/2) *(dist /2)),norm=dist ? r/dist :
0;
            maxCount= max(maxCount,count(darts,midX-norm* dy,midY+
norm*dx,r));
         }
      }
      returnmaxCount;
   }

private:
   intcount(constvector<vector<int>>&darts,doublecx,doublecy,intr){ int c =
      0;
      for(auto&dart:darts)
         if(pow(dart[0]-cx,2)+pow(dart[1]-cy,2)<=r* r+1e-7)++c; return c;
   }
};

int main() {
   Solutionsol;
   vector<vector<int>>darts={{-2,0},{2,0},{0,2},{0,-2}}; cout <<
   sol.numPoints(darts, 2) << endl; // Output: 4
}
```

Problem13: DesignSkiplist

DesignaSkiplistwithoutusinganybuilt-inlibraries.Askiplistis adatastructure thattakesO(log(n))timetoadd,eraseandsearch.Comparingwith treapandred- black tree which has the same function and performance, the code length of Skiplist can be comparatively short and the idea behind Skiplists is just simple linked lists.

Solution:

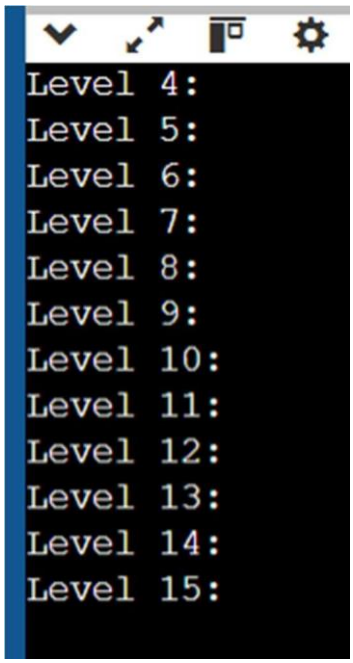
```
#include<iostream>
#include <cstdlib>
#include <ctime>
#include <vector>

usingnamespace std;

//NodestructureforSkiplist
struct Node {
    intvalue;
    vector<Node*>forward;// Pointerstonextnodesateach level

    Node(intvalue,intlevel):value(value),forward(level,nullptr){}
};

//Skiplistclass
classSkiplist{
private:
    intmaxLevel;            //Maximumlevel for theskiplist
    floatprobability;//Probabilityofpromotinganodetoahigherlevel      Node*
    header;                 // Header node

    //Randomlevelgeneratorfunction
    int randomLevel() {
        int level = 1;
        while((rand()%2)<probability&&level<maxLevel){
```

```
            level++;
        }
        returnlevel;
    }

public:
    Skiplist(intmaxLevel=16,floatprobability=0.5)
        :maxLevel(maxLevel),probability(probability){
        header=newNode(-1,maxLevel);//Headernode withadummyvalue
    }

    //SearchforavalueintheSkiplist bool
    search(int target) {
        Node* current=header;
        for(inti =maxLevel -1; i>=0; --i) {
            while(current->forward[i]!=nullptr&&current->forward[i]->value<
target) {
                current=current->forward[i];
            }
        }
        current=current->forward[0];
        return(current!=nullptr&&current->value== target);
    }

    //InsertavalueintotheSkiplist
    void insert(int value) {
        vector<Node*>update(maxLevel,nullptr);
        Node* current = header;

        for(inti =maxLevel -1; i>=0; --i) {
            while(current->forward[i]!=nullptr&&current->forward[i]->value<
value) {
                current=current->forward[i];
            }
            update[i]=current;
        }

        current=current->forward[0];
        if(current==nullptr||current->value!=value){ int
            level = randomLevel();
            Node*newNode= newNode(value,level);
```

```
            for(int i =0;i <level; ++i){
                newNode->forward[i]=update[i]->forward[i];
                update[i]->forward[i] = newNode;
            }
        }
    }

    //EraseavaluefromtheSkiplist void
    erase(int value) {
        vector<Node*>update(maxLevel,nullptr);
        Node* current = header;

        for(inti =maxLevel -1; i>=0; --i) {
            while(current->forward[i]!=nullptr&&current->forward[i]->value<
value) {
                current=current->forward[i];
            }
            update[i]=current;
        }

        current=current->forward[0];
        if(current!=nullptr&&current->value==value){ for
            (int i = 0; i < maxLevel; ++i) {
                if (update[i]->forward[i] != current) break;
                update[i]->forward[i]=current->forward[i];
            }
            deletecurrent;
        }
    }

    //PrinttheSkiplist void
    print() {
        for (int i = 0; i < maxLevel; ++i) {
            Node*current=header->forward[i];
            cout <<"Level "<< i <<": ";
            while(current!=nullptr) {
                cout<<current->value<<"";
                current = current->forward[i];
            }
            cout<<endl;
        }
    }
```

```
};

intmain(){
    srand(time(0));//Seedforrandomlevelgeneration Skiplist

    skiplist;

    //InsertvaluesintotheSkiplist
    skiplist.insert(30);
    skiplist.insert(40);
    skiplist.insert(50);
    skiplist.insert(60);
    skiplist.insert(70);
    skiplist.insert(90);

    //Printthe Skiplist
    cout<<"Skiplistafterinsertions:"<<endl;
    skiplist.print();

    //Insertadditionalvalues
    skiplist.insert(80);
    skiplist.insert(45);

    cout<<"Skiplistafteradding80and45:"<<endl;
    skiplist.print();

    //Searchforsome values
    cout<<"Searchingfor45:"<<(skiplist.search(45)? "Found":"NotFound")
<<endl;
    cout<<"Searchingfor100:"<<(skiplist.search(100)?"Found":"Not Found")
<< endl;

    // Erase a value
    skiplist.erase(50);
    cout<<"Skiplistafterremoving50:"<<endl;
    skiplist.print();

    return0;
}
```

Ouput:



Problem14: AllO`one Data Structure

Design a data structure to store the strings' count with the ability to return the strings with minimum and maximum counts. Implement the AllOne class: ● AllOne() Initializes the object ofthe data structure.● inc(String key) Increments the count of the string key by 1. If key does not exist in the data structure, insert it with count 1.● dec(String key) Decrements the count ofthe stringkeyby1. If the count of key is 0 after the decrement, remove it from the data structure. It is guaranteed that key exists in the data structure before the decrement. ● getMaxKey() Returns one of the keys with the maximal count. If no element exists,returnanemptystring"".●getMinKey()Returnsoneofthekeys withthe minimum count. If no element exists, return an empty string "".

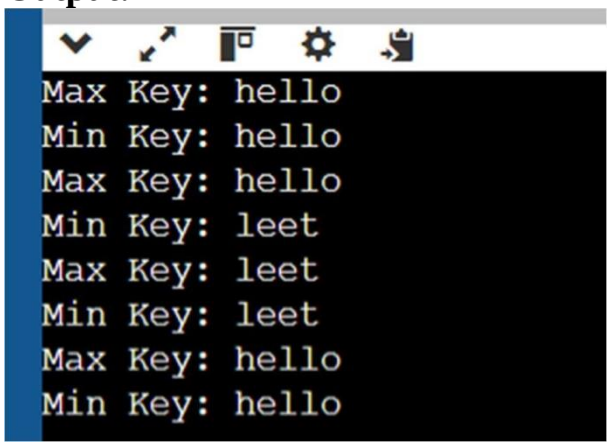
Solution:

```
#include <iostream>
#include<unordered_map>
#include <string>
#include <list>
#include <map>
usingnamespacestd;
class AllOne {
private:
    //Hashmaptostorethecountofeachkey
    unordered_map<string, int> keyCount;
```

```
    //Maptostorethestringsbytheircounts,orderedbycounts map<int,
    list<string>> countKeys;
public:
    AllOne() {}
    //Incrementthecountofastringby1 void
    inc(string key) {
        intcount=keyCount[key];
        //Removethekeyfromitsoldcountlist if
        (count > 0) {
            countKeys[count].remove(key);
             if(countKeys[count].empty()){
                countKeys.erase(count);}}
        //Incrementthecountandaddthekeytothenewcountlist keyCount[key] =
        count + 1;
        countKeys[count+ 1].push_back(key);}
    //Decrementthecountofastringby1 void
    dec(string key) {
        intcount=keyCount[key];
        //Removethekeyfromitscurrentcountlist countKeys[count].remove(key);
        if(countKeys[count].empty()){
            countKeys.erase(count); }
        //Ifcountbecomes0,removethekeyfromkeyCount if
        (count == 1) {
            keyCount.erase(key);
        } else{
            keyCount[key] = count - 1;
            countKeys[count-1].push_back(key);}}
    //Getthekeywiththemaximumcount
    string getMaxKey() {
        if(countKeys.empty())return"";
        // Get the last element (maximum count)
        returncountKeys.rbegin()->second.front();
    }

    //Getthekeywiththeminimumcount
    string getMinKey() {
        if(countKeys.empty())return"";
        // Get the first element (minimum count)
        returncountKeys.begin()->second.front();
    }
```

```
};

int main() {
    AllOneallOne;

    allOne.inc("hello");
    allOne.inc("hello");
    cout<<"MaxKey:"<<allOne.getMaxKey()<<endl;//"hello" cout
    <<"Min Key: "<< allOne.getMinKey() << endl; // "hello"

    allOne.inc("leet");
    cout<<"MaxKey:"<<allOne.getMaxKey()<<endl;//"hello" cout
    <<"Min Key: "<< allOne.getMinKey() << endl; // "leet"

    allOne.dec("hello");
    cout<<"MaxKey:"<<allOne.getMaxKey()<<endl;//"hello" cout
    <<"Min Key: "<< allOne.getMinKey() << endl; // "leet"

    allOne.dec("leet");
    cout<<"MaxKey:"<<allOne.getMaxKey()<<endl;//"hello" cout
    <<"Min Key: "<< allOne.getMinKey() << endl; // "hello"

    return0;
}
```

Output:



Problem15: FindMinimumTimetoFinish All Jobs

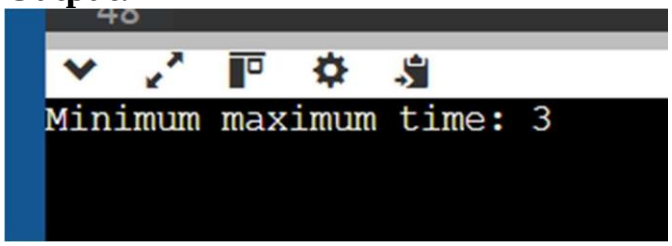
You are given an integer arrayjobs, where jobs[i] is the amount of time it takes tocompletetheithjob.Therearekworkersthatyoucanassignjobsto.Eachjob should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the maximum working time of any worker is minimized. Return the minimum possible maximum working time of any assignment.

Solution:

```
#include <iostream>
#include <vector>
#include <numeric>
#include<algorithm>

usingnamespace std;

// Helperfunctiontocheckifwecandistributejobswithmaxworkload<=mid bool
canAssignJobs(const vector<int>& jobs, int k, int mid) {
    int currentSum=0;
    int workersUsed =1;// Start withoneworker

    for(intjob :jobs){
        if(currentSum+job>mid) {
            //Needanewworkersincecurrentworkercannottakethisjob
            workersUsed++;
            currentSum=job;
            if(workersUsed>k)returnfalse;//Moreworkersthan allowed
        } else{
            currentSum+= job;
        }
    }
    returntrue;
}

intminimumTimeRequired(vector<int>&jobs,intk){
    int left = *max_element(jobs.begin(), jobs.end()); // Max job time
    intright=accumulate(jobs.begin(),jobs.end(),0);//Sumofalljobs

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

```
        if(canAssignJobs(jobs,k,mid)){
            right= mid;//Tryfor asmallermax workload
        } else{
            left=mid+1;//Increasethe allowedmaxworkload
        }
    }

    returnleft;
}

intmain(){
    vector<int>jobs={3,2,3}; int
    k = 3;
    cout<<"Minimummaximumtime:"<<minimumTimeRequired(jobs,k)<<endl;
    return0;
}
```

Output:



Problem16: MinimumNumberofPeople toTeach:
On a social network consisting of m users and some friendships between users, two users can communicate with each other if they know a common language. You are given an integer n, an array languages, and an array friendships where: Therearenlanguagesnumbered1throughn,languages[i]is thesetoflanguages the ith user knows, and friendships[i] = [ui, vi] denotes a friendship between the users ui and vi. You can choose one language and teach it to some users so that all friends can communicate with each other. Return the minimum number of usersyouneed toteach.Note thatfriendshipsarenottransitive,meaningifxis a friend of y and y is a friend of z, this doesn't guarantee that x is a friend of z.

Solution:

```
#include <iostream>
#include <vector>
#include<unordered_set>
#include<unordered_map>

using namespace std;

classUnionFind{
public:
   UnionFind(intn){
      parent.resize(n);
      size.resize(n,1);
      for(inti = 0; i <n; ++i)parent[i]=i;
   }

   intfind(intu) {
      if(parent[u]!=u){
         parent[u]=find(parent[u]);//Path compression
      }
      return parent[u];
   }

   voidunionSets(intu,intv){ int
      rootU = find(u);
      int rootV = find(v);
      if(rootU!=rootV){
         if(size[rootU]<size[rootV])swap(rootU,rootV);
         parent[rootV] = rootU;
         size[rootU]+= size[rootV];
      }
   }

private:
   vector<int>parent;
   vector<int> size;
};

intminimumTeachings(intn,vector<vector<int>>&languages,
vector<vector<int>>& friendships) {
   intm= languages.size();// Number of users
```

```
//Step1:Union-Findinitializationforusers
UnionFind uf(m);

//Step2:Unionuserswhoshareacommonlanguage
unordered_map<int, vector<int>> languageUsers;
for (int i = 0; i < m; ++i) {
    for (int lang : languages[i]) {
        languageUsers[lang].push_back(i);
    }
}

// Union users based on shared languages
for(auto&[lang,users]:languageUsers){
    for(inti = 0;i <users.size(); ++i){
        for(intj=i+1;j<users.size();++j){
            uf.unionSets(users[i], users[j]);
        }
    }
}

//Step3:Unionusersbasedonfriendships for
(auto& friendship : friendships) {
    intu=friendship[0]-1;
    intv=friendship[1]-1;
    if(uf.find(u)!=uf.find(v)){
        uf.unionSets(u, v);
    }
}

//Step4:Findtheconnectedcomponentsandcheckiftheycancommunicate
unordered_map<int, unordered_set<int>> components;
for(inti=0;i<m;++i){ int
    root = uf.find(i);
    for (int lang : languages[i]) {
        components[root].insert(lang);
    }
}

// Step5:Determinehowmanyusersneedtobetaughtanewlanguage int
result = 0;
for(auto&[component,langs]:components){
```

```
        //Ifnolanguageis sharedamongusers inthiscomponent,weneedtoteach at least
one user
        if(langs.empty()){
            result++;
        }
    }

    returnresult;
}

intmain(){
    vector<vector<int>>languages1={{1},{2},{1, 2}};
    vector<vector<int>>friendships1={{1,2},{1,3},{2,3}}; int
    n1 = 2;
    cout<<"Minimumnumberofuserstoteach:"<<minimumTeachings(n1, languages1,
friendships1) << endl;

    vector<vector<int>>languages2= {{2},{1,3},{1,2},{3}};
    vector<vector<int>>friendships2={{1,4},{1,2},{3,4},{2,3}}; int n2
    = 3;
    cout<<"Minimumnumberofuserstoteach:"<<minimumTeachings(n2, languages2,
friendships2) << endl;

    return0;
}
```

Output:

Output

```
Minimum number of users to teach: 0
Minimum number of users to teach: 0
```

Problem17: CountWaystoMakeArrayWithProduct

Youaregivena2Dintegerarray,queries.Foreachqueries[i],wherequeries[i]= [ni,ki],findthenumberof different ways you can placepositiveintegers into an arrayof sizeni suchthatthe productof theintegersis ki. Asthenumber of ways may be too large, the answer to the ith query is the number of ways modulo 109 +7.Returnanintegerarrayanswerwhereanswer.length==queries.length,and answer[i] is the answer to the ith query.

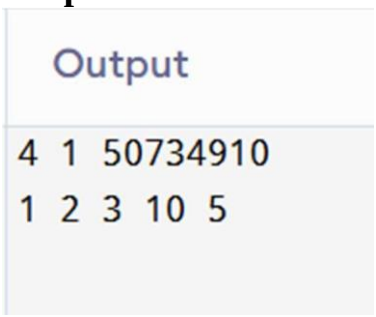
Solution:

```
#include<iostream>
#include <vector>
#include <cmath>

const int MOD = 1e9 + 7;
constintMAXN=10000;

//PrecomputefactorialsandmodularinversesusingFermat'sLittleTheorem
std::vector<long long> factorial(MAXN + 1), inv_factorial(MAXN + 1);

//Functiontocomputex^y% MOD
longlongmod_exp(longlongx,longlongy,longlongmod){ long long
   result = 1;
   while(y>0){
      if(y%2==1)result=(result*x)%mod; x = (x
      * x) % mod;
      y /=2;
   }
   returnresult;
}

//Functiontoprecomputefactorialsandtheirinverses void
precompute() {
   factorial[0]=inv_factorial[0]=1; for
   (int i = 1; i <= MAXN; ++i) {
      factorial[i]= (factorial[i- 1]*i) %MOD;
   }
   inv_factorial[MAXN]=mod_exp(factorial[MAXN],MOD-2,MOD);//
Using Fermat's little theorem
   for(int i=MAXN- 1; i>= 1; --i) {
      inv_factorial[i]= (inv_factorial[i+1]*(i +1))%MOD;
   }
```

```
}

//Functiontocomputebinomialcoefficient C(n,k)%MOD long
long binomial(int n, int k) {
    if(k > n ||k<0)return0;
    return(factorial[n]*inv_factorial[k]%MOD)*inv_factorial[n-k]%MOD;
}

// Functiontogettheprimefactorizationofanumber
std::vector<std::pair<int,int>>prime_factors(intk){
    std::vector<std::pair<int,int>>factors;
    for (int i = 2; i * i <= k; ++i) {
        if(k%i==0){
            intcount=0;
            while(k%i==0){ k
                /= i;
                count++;
            }
            factors.push_back({i,count});
        }
    }
    if (k > 1) {
        factors.push_back({k,1});
    }
    returnfactors;
}

//Functiontosolveeachquery
long long solve(int n, int k) {
    //Primefactorizek
    autofactors=prime_factors(k);

    longlongresult=1;
    for(constauto&factor:factors){ int
        prime = factor.first;
        int exponent = factor.second;
        //Calculatenumberofwaystosplitexponentofthisprimeintonparts result =
        (result * binomial(exponent + n - 1, n - 1)) % MOD;
    }

    returnresult;
}
```

```
std::vector<int>waysToPlaceIntegers(std::vector<std::vector<int>>&queries){
    precompute(); // Precompute factorials and inverses
    std::vector<int>result;

    for(auto&query:queries){ int n
        = query[0];
        int k = query[1];
        result.push_back(solve(n,k));
    }

    returnresult;
}

intmain(){
    //Exampleusage
    std::vector<std::vector<int>>queries1={{2,6},{5,1},{73,660}}; std::vector<int>
    result1 = waysToPlaceIntegers(queries1);

    for (int r : result1) {
        std::cout<<r<<"";
    }
    std::cout<< std::endl;

    std::vector<std::vector<int>>queries2={{1,1},{2,2},{3, 3},{4,4},{5,
5}};
    std::vector<int>result2=waysToPlaceIntegers(queries2);

    for (int r : result2) {
        std::cout<<r<<"";
    }
    std::cout<< std::endl;

    return0;
}
```

Output:



Problem18: MaximumTwin Sumof aLinked List

In a linked list of size n, where n is even, the ith node (0-indexed) of the linked list is known as the twin of the (n-1-i)th node, if 0 <= i <= (n / 2) - 1. ● For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node2.Thesearetheonlynodes withtwins forn=4.Thetwinsumis definedas the sum of a node and its twin. Given the head of a linked list with even length, return the maximum twin sum of the linked list.

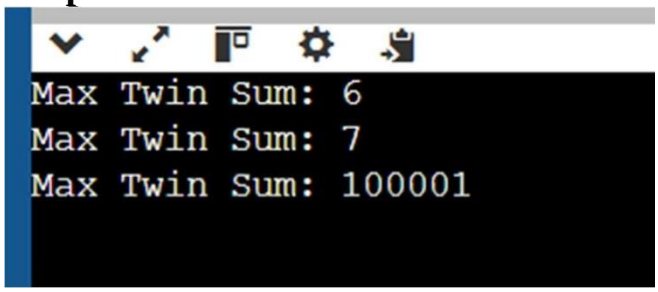
Code:

```
#include <iostream>
#include <vector>
#include<algorithm>

usingnamespace std;

//Definitionforsingly-linkedlist.
struct ListNode {
   int val;
   ListNode*next;
   ListNode(intx):val(x),next(nullptr){}
};

classSolution{
public:
   intpairSum(ListNode*head){
      //Step1:Storethevalues ofthelinkedlistinavector vector<int>
      values;
      ListNode*current=head;

      //Traversethelinkedlistandaddeachnode'svaluetothevector while
      (current != nullptr) {
         values.push_back(current->val);
```

```
            current=current->next;
        }

        //Step2:Calculatethemaximumtwinsum int n
        = values.size();
        intmaxTwinSum=0;

        //Iterateoverthefirsthalfofthelistandcalculatetwinsums for (int i
        = 0; i < n / 2; ++i) {
            int twinSum = values[i] + values[n - 1 - i];
            maxTwinSum=max(maxTwinSum,twinSum);
        }

        returnmaxTwinSum;
    }
};

//Helperfunctiontocreatealinkedlistfromavector
ListNode* createList(const vector<int>& nums) {
    ListNode*head=newListNode(nums[0]);
    ListNode* current = head;
    for (int i = 1; i < nums.size(); ++i) {
        current->next=newListNode(nums[i]);
        current = current->next;
    }
    returnhead;
}

intmain(){
    //Example1
    vector<int> input1 = {5, 4, 2, 1};
    ListNode*head1=createList(input1);
    Solution sol;
    cout<<"MaxTwinSum:"<<sol.pairSum(head1)<<endl;//Output:6

    //Example2
    vector<int> input2 = {4, 2, 2, 3};
    ListNode*head2=createList(input2);
    cout<<"MaxTwinSum:"<<sol.pairSum(head2)<<endl;//Output:7

    //Example3
    vector<int>input3={1,100000};
```

```
    ListNode* head3=createList(input3);
    cout<<"MaxTwinSum:"<<sol.pairSum(head3)<<endl;//Output: 100001

    return0;
}
```

Output:



Problem19: InsertGreatestCommonDivisorsinLinkedList
Given the head of a linked list head, in which each node contains an integer value. Between every pair of adjacent nodes, insert a new node with a value equal to the greatest common divisor of them. Return the linked list after insertion.Thegreatestcommondivisoroftwonumbersisthelargestpositive integer that evenly divides both numbers.
Solution:

```
#include<iostream>
#include<vector>//Includethevectorheader #include
<algorithm>
usingnamespace std;

//Definitionforsingly-linkedlist.
struct ListNode {
    int val;
    ListNode*next;
    ListNode(intx):val(x),next(nullptr){}
};

classSolution{
public:
    //FunctiontocomputeGCDoftwonumbers int
    gcd(int a, int b) {
        while(b!=0){
```

```
            inttemp=b; b
            = a % b;
            a= temp;
        }
        returna;
    }

    //FunctiontoinsertGCDnodesbetweeneachpairofadjacentnodes ListNode*
    insertGreatestCommonDivisors(ListNode* head) {
        //Edgecase: Ifthelistisemptyorhasonlyonenode,noinsertionisneeded if (!head
        || !head->next) return head;

        ListNode*current=head;

        //Traversethelist
        while(current&&current->next){
            intgcdValue=gcd(current->val,current->next->val);//Calculatethe
GCD
            ListNode*newNode=newListNode(gcdValue); //Createa newnode
withtheGCDvalue
            newNode->next=current->next;//Linkthenewnodetothenextnode
            current->next = newNode; // Link the current node to the new node
            current = newNode->next; // Move to the next pair of nodes
        }

        return head;
    }
};

//Helperfunctiontocreatealinkedlistfromavector
ListNode* createList(const vector<int>& values) {
    if (values.empty()) return
    nullptr;ListNode*head=newListNode(value
    s[0]); ListNode* current = head;
    for (int i = 1; i < values.size(); ++i) {
        current->next=newListNode(values[i]);
        current = current->next;
    }
    returnhead;
}

//Helperfunctiontoprintthelinkedlist
```

```
voidprintList(ListNode*head){
    while (head) {
        cout<<head->val;
        if(head->next)cout<<"->"; head
        = head->next;
    }
    cout<<endl;
}

int main() {
    Solutionsol;

    // Testcase1
    vector<int> values1 = {18, 6, 10, 3};
    ListNode*head1=createList(values1);
    ListNode*result1=sol.insertGreatestCommonDivisors(head1);
    printList(result1);//Expected:18->6->6 ->2->10->1->3

    // Testcase2
    vector<int>values2={7};
    ListNode* head2=createList(values2);
    ListNode*result2=sol.insertGreatestCommonDivisors(head2);
    printList(result2);// Expected: 7

    return0;
}
```

Output:

Output

```
18 -> 6 -> 6 -> 2 -> 10 -> 1 -> 3
7
```