PROVA IN ITINERE 01/12/2022

Zevola Armando

M63001514

**Esercizio 1**

*Codice*:

#include <bits/stdc++.h>

using namespace std;

int algorithm(vector<int> array, int k) {

    vector<int>::iterator it\_lower, it\_upper;

    int index\_lower, index\_upper;

    // Riordino l'array con sort

    sort(array.begin(), array.end());

    it\_lower = lower\_bound(array.begin(), array.end(), k);

    index\_lower = it\_lower - array.begin();

    //cout << index\_lower << endl;

    it\_upper = upper\_bound(array.begin(), array.end(), k);

    index\_upper = it\_upper - array.begin();

    //cout << index\_upper << endl;

    return (index\_upper - index\_lower);

}

int main() {

    //Inserisci test case

    int tc;

    int k;

    int value;

    int dim\_array;

    int index = 1;

    cout << "Test case: ";

    cin >> tc;

    cout << "\n\n";

    vector<int> array;

    vector<int> dimensioni(tc);

    vector<int> output;

    while(tc) {

        cout << "TEST CASE " << index << endl;

        cout << "Valore K: ";

        cin >> k;

        //metto la dimensione dell'array

        cout << "Dimensione array: ";

        cin >> dim\_array;

        //Riempio l'array

        cout << "Valori array: ";

        for(int i=0; i<dim\_array; i++) {

            cin >> value;

            array.push\_back(value);

        }

        output.push\_back(algorithm(array, k));

        cout << "\n\n";

        index++;

        tc --;

    }

    cout << "Output: " << endl;

    for(int i=0; i<output.size(); i++) {

        cout << output[i] << endl;

    }

    return 0;

}

*Complessità*

La funzione sort() riordina il vettore con complessità O(nlogn), mentre le funzioni upper\_bound() e lower\_bound() sono degli algoritmi di ricerca binaria modificati che hanno complessità O(logn)

In generale, la complessità è O(nlogn)

NB: in alternativa alla funzione sort avrei potuto implementare una funzione di merge sort

**Esercizio 2**

*Codice*

#include <bits/stdc++.h>

using namespace std;

/\*

    S: somma

    P: un numero primo

    N: elementi strettamente maggiori di P tale che la loro somma è uguale a S

\*/

vector<vector<int>> solutions;

// La isSafe deve valutare se il numero è primo

bool isSafe(int num) {

    for(int i=2; i<num; i++) {

        // se il resto della divisione col numero i-esimo è ritorna falso

        if(num % i == 0) return false;

    }

    return true;

}

void printSolutions() {

    for(int i=0; i<solutions.size(); i++) {

        for(int j=0; j<solutions[0].size(); j++) {

            cout << solutions[i][j] << " ";

        }

        cout << "\n";

    }

}

void clearAll(vector<int> & elem, int & count) {

    elem.clear();

    count = 0;

    solutions.clear();

    solutions[0].clear();

}

void algorithm(int s, int n, int p, vector<int> elem, int & count) {

    if(elem.size()==n) {

        if(count == s) {

            solutions.push\_back(elem);

            return;

        }

        else return;

    }

    for(int i = p+1; i<=s; i++) {

        if(isSafe(i) && count+i<=s) {

            count += i;

            elem.push\_back(i);

            algorithm(s, n, i, elem, count);

            //backtracking

            count -= i;

            elem.pop\_back();

        }

    }

    return;

}

int main() {

    vector<int> elem;

    int count = 0;

    int index = 1;

    int tc;

    int s, n, p;

    cin >> tc;

    while(tc) {

        cin >> s;

        cin >> n;

        cin >> p;

        algorithm(s, n, p, elem, count);

        cout << "\n\nCASO DI TEST " << index << endl;

        printSolutions();

        cout << "\n\n";

        clearAll(elem, count);

        index++;

        tc--;

    }

}

*Complessità*

La complessità è esponenziale.