SEARCHING -EXERCISE

1) *Given an array arr[], check whether it is sorted in non-decreasing order. Return true if it is sorted otherwise false.*

ANSWER:

```
class Solution {
  public:
    bool isSorted(vector<int>& arr) {
        int n = arr.size();
        for (int i = 0; i < n - 1; i ++){
            if ( arr[i] > arr[i+1])
            return false;
        }
        return true;
    }
};
```

2) *Hãy viết chương trình nhập vào một danh sách A gồm tọa độ n điểm trong không gian 3 chiều. Sau đó, nhập hai giá trị b và e, với b ≤ e, để chọn tập R chứa các điểm A[i] sao cho khoảng cách từ gốc tọa độ O(0,0,0) đến điểm A[i] là d(O, A[i]) ∈ [b, e]. Thứ tự dò tìm từ đầu danh sách A đến cuối danh sách A.*
Đầu vào:
- Dòng đầu là số n (1 ≤ n ≤ 10000).
- n dòng kế tiếp, mỗi dòng 3 số có phần thập phân x, y và z là giá trị các chiều của một điểm trong danh sách A.
- Sau cùng là b với e
Đầu ra: Nếu tập R rỗng thì xuất ra "KHONG". Nếu R không rỗng thì liệt kê các giá trị x, y và z của tọa độ các điểm trong R theo từng dòng, mỗi dòng là một điểm.

Phân tích đề:

- Nhập n điểm trong không gian 3 chiều: chạy vòng lặp < n, tạo struct gồm (x, y, z) → 1 vector sẽ có cấu trúc của 1 struct.
- Nhập b và e: Cho 1 khoảng giới hạn
- Tính khoảng cách giữa các vector sao cho d thuộc khoảng [b,e]

ANSWER:

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

struct Point{
    float x, y, z; // Tạo 1 struct có cấu tạo x, y, z. Đây là những thành phần của điểm 3D
};

void input(vector <Point>& a, int n) { // Tạo 1 vector chứa các điểm 3D
    for (int i = 0; i < n; i++) {
        cin >> a[i].x >> a[i].y >> a[i].z; // Nhập lần lượt các giá trị x, y, z của vector 3D đó
    }
}

float calculate(Point a) { // Tính khoảng cách của điểm
    return sqrt(a.x * a.x + a.y *a.y + a.z * a.z);
}
```

```cpp
int main() {
    vector<Point> a;
    int n;
    cin >> n;
    a.resize(n); // cấp cho vector a có size n

    float b, e;
    input(a, n);
    cin >> b >> e; // Nhập khoảng giá trị

    bool flag = false;
    for (int i = 0; i < n; i++) {
        if (calculate(a[i]) >= b && calculate(a[i]) <= e) {
            cout << a[i].x << " " << a[i].y << " " << a[i].z << endl;
            flag = true;
        }
    }
    if (!flag) {
        cout << "KHONG";
    }
    return 0;
}
```

3) *Hãy viết chương trình nhập vào một danh sách A gồm tọa độ n điểm trong không gian 3 chiều. Sau đó, nhập m điểm pi và kiểm tra xem các điểm pi này có xuất hiện trong danh sách A hay không.*

Đầu vào:

- Dòng đầu là số n (1 ≤ n ≤ 10000)

- n dòng kế tiếp, mỗi dòng 3 số có phần thập phân lần lượt là giá trị x, y và z của điểm trong danh sách.

- Dòng tiếp theo là số m (1 ≤ m ≤ 3500).

- m dòng tiếp theo, mỗi dòng có 3 số có phần thập phân lần lượt là giá trị x, y và z của điểm pi.

Đầu ra: Gồm m dòng, dòng thứ i là kết quả tìm điểm pi trong A. Kết quả tìm là

- "KHONG" nếu không tìm thấy,

- Chỉ số của điểm đầu tiên được tìm thấy trong A theo hướng tìm từ đầu danh sách đến cuối danh sách.

Phân tích đề:

- Nhập n điểm vào danh sách A → nhập m vào danh sách B và kiểm tra thuộc A
- Sử dụng linear search hoặc thư viện STL.
- "điểm pi này có xuất hiện trong danh sách A" → Phải duyệt các phần A lần lượt qua B.

ANSWER:

```
int main() {
    int n;
    int m;
    cin >> n;

    vector<Point> a(n); // khai báo vector a có khoảng n
    for (int i = 0; i < n; ++i) {
        cin >> a[i].x >> a[i].y >> a[i].z;
    }

    cin >> m;
    vector<Point> b(m); // khai báo vector b có khoảng m
    for (int i = 0; i < m; ++i) {
        cin >> b[i].x >> b[i].y >> b[i].z;
    }

    for (int i = 0; i < m; i++) { // duyệt từng điểm b[i] để tìm trong a
        int pos = -1; // gọi pos = -1

        for (int j = 0; j < n; j++) {   // tìm trong a
            if (a[j].x == b[i].x &&
                a[j].y == b[i].y &&
                a[j].z == b[i].z) {
                pos = j; // cập nhật giá trị pos
                break; // chỉ lấy giá trị pos đầu tiên rồi chuyển sang giá trị khác
            }
        }
```

```
        if (pos == -1) {
            cout << "KHONG\n";
        }
        else {
            cout << pos << "\n";
        }
    }

    return 0;
}
```

4) *Hãy viết chương trình nhập vào một danh sách A gồm tọa độ n điểm trong không gian 3 chiều đã được sắp xếp tăng dần theo giá trị chiều x, nếu giá trị chiều x bằng nhau thì xét tăng dần theo giá trị chiều y, nếu giá trị chiều y bằng nhau thì xét tăng dần theo giá trị chiều z. Sau đó, nhập m điểm pi và kiểm tra*

bằng thuật toán tìm nhị phân xem các điểm pi này có xuất hiện trong danh sách A hay không.
Đầu vào:
- Dòng đầu là số n (1 ≤ n ≤ 10000)
- n dòng kế tiếp, mỗi dòng 3 số có phần thập phân lần lượt là giá trị x, y và z của điểm trong danh sách.
- Dòng tiếp theo là số m (1 ≤ m ≤ 3500).
- m dòng tiếp theo, mỗi dòng có 3 số có phần thập phân lần lượt là giá trị x, y và z của điểm pi.
Đầu ra: Gồm m dòng, dòng thứ i là kết quả tìm điểm pi trong A. Kết quả tìm là
- "KHONG" nếu không tìm thấy,
- Số bước chia danh sách A theo thuật toán tìm nhị phân để xác định điểm đầu tiên được tìm thấy trong A.

Phân tích đề:

- Nhập n điểm vào danh sách A → nhập m vào danh sách B và kiểm tra thuộc A
- Sử dụng linear search hoặc thư viện STL.
- “điểm pi này có xuất hiện trong danh sách A” → Phải duyệt các phần A lần lượt qua B.

SORTING - LT

CÁC DẠNG SORTING

Selection Sort is a comparison-based sorting algorithm. It sorts an array by repeatedly selecting the **smallest (or largest)** element from the unsorted portion and swapping it with the first unsorted element. This process continues until the entire array is sorted.

DỊCH: Selection Sort là dạng sắp xếp bằng cách **chọn ra giá trị nhỏ nhất hoặc lớn nhất trong mảng** sau đó swap nó với phần tử đầu tiên chưa được sắp xếp trong mảng.

Cú pháp:

```
void selectionSort(vector<int> &arr) {
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i) {
// có nghĩa là mảng {3,2,1} int i chạy từ 0 -> 1
// i < n - 1 là vì mình phải chọn ra 1 giá trị để làm mốc so sánh, VD: mảng 9 phần tử, lấy đi phần tử a[0] thì còn lại 8 phần tử.
        int min_idx = i;
        for (int j = i + 1; j < n; ++j) {
          // vòng lặp j + 1 để chọn phần tử đằng i, lúc này j sẽ là 8 phần tử còn lại
          // chạy j = i + 1, tức nếu i = 0 thì j = 1, j chạy từ 0 -> 2
            if (arr[j] < arr[min_idx]) {
              // nếu giá trị tại vị trí a[j] < a[min] (== a[i]) thì đổi chỗ
                min_idx = j;
              // cho mảng {3,2 1}: a[i] = 3 và a[j] = 2. Sau so sánh thì a[min] = 2 và min = 1. Tương tự a[min] = 2 > arr[2] = 1 => min = 2 và arr[min] = 1
            }
        }
        swap(arr[i], arr[min_idx]);
      // swap arr[0] == 3 với arr[min] = 1
    }
}
```

Bubble Sort is the simplest sorting algorithm that works by **repeatedly swapping the adjacent elements** if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity are quite high.

- We sort the array using multiple passes. After the first pass, the maximum element goes to end (its correct position). Same way, after second pass, the second largest element goes to second last position and so on.

DỊCH: Bubble Sort là một dạng sắp xếp cơ bản nhất, hoạt động dựa trên nguyên lí **lặp lại liên tục việc swap các phần tử khi nó đang nằm sai vị trí.**

- Ta sắp xếp mảng dựa vào nhiều lần truyền nhau. Sau lần đầu tiên, phần tử lớn nhất được xếp ở cuối mảng (vị trí đúng thứ tự). Làm tương tự với các phần tử khác

Cú pháp:

```
void bubbleSort(vector<int>& arr) {

    int n = arr.size();

    bool swapped;

    for (int i = 0; i < n - 1; i++) {

        swapped = false;

      // đặt cờ: cho swap = false

        for (int j = 0; j < n - i - 1; j++) {

          // VD: 1 mảng gồm 3 phần tử. Ở vòng lặp i: Chọn phần tử i = 0  để so sánh với 2 phần tử còn (n-1)

            // Sau khi chạy i vòng ngoài, thì đã có i phần tử cuối cùng của mảng được đặt đúng vị trí (đã sorted)→ Vì vậy vòng j KHÔNG cần đụng tới các phần tử đó
```

```
        if (arr[j] > arr[j + 1]) {
            swap(arr[j], arr[j + 1]);
            swapped = true;
        }
      }
      if (!swapped)
        break;
   }
}
```

Insertion sort is a simple sorting algorithm that works by iteratively **inserting each element of an unsorted list** into its correct position in a sorted portion of the list. It is like sorting playing cards in your hands. You split the cards into two groups: the sorted cards and the unsorted cards. Then, you pick a card from the unsorted group and put it in the right place in the sorted group.

DỊCH: Insertion Sort là một dạng sắp xếp đơn giản hoạt động dựa trên việc **"nhập" những phần tử chưa được sắp xếp vào đúng vị trí trong danh sách.**

Cú pháp:

```
void insertionSort(int arr[], int n)
{
   for (int i = 1; i < n; ++i) {
     // chạy vòng lặp từ 1 đến n
      int key = arr[i];
      int j = i - 1;
```

```
        while (j >= 0 && arr[j] > key) {
          // xét điều kiện để j >= 0 và a[j] lớn hơn a[i]
            arr[j + 1] = arr[j];
            j = j - 1;
        }
       // trả về vị trí được sắp xếp
        arr[j + 1] = key;
    }
}
// vd cho mảng [3, 4, 1, 6]
// vòng lặp i = 1: key = 4. Xét a[j] = 3 (j = 0) > key = 4? --> không đúng nên thoát vòng while. Trả về a[j+1] = key = a[i] = 4
// vòng lặp i = 2: key = 1. Xét a[j]= 4 (j = 1) > key = 1?--> đúng nên chạy vòng lặp j: a[j+1] = a[2] = 4, j - 1 = 0--> chạy tiếp đến hết
```

QuickSort is a sorting algorithm based on the **Divide and Conquer** that **picks an element as a pivot** and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

DỊCH: Quick Sort là thuật toán sắp xếp dựa trên nguyên lí **chia để trị**. Chọn 1 phần tử làm pivot sau đó sắp xếp các mảng dựa trên các pivot.

Chia làm 3 bước chính:

1. **Chọn Pivot**
2. **Phân hoạch mảng (Partition)**
3. **Gọi đệ quy**

Cú pháp: (LOMUTO)

```
//VD: Cho 1 mảng gồm các giá trị [2,8,7,1,5,3,6,4]
int partition(vector<int>& arr, int low, int high) {
  // chọn 1 pivot bằng giá trị chỉ số cuối
    int pivot = arr[high]; // pivot = 4, high = 7

  // cho i = low - 1: Tức là chỉ số phần phía bên tay trái -1
    int i = low - 1;
```

```
  // chạy vòng lặp bên phải: Từ low -> high - 1
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) { // nếu a[j] < pivot thì tăng i
            i++; // VD: // a[j] = a[0] = 2 < pivot = 4 → i = 0
            swap(arr[i], arr[j]); // swap (a[0], a[0])
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(vector<int>& arr, int low, int high) {
   if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

SORTING - EXCERCISE

1) *Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.*

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Phân tích đề:

- Chạy 2 vòng lặp: Trong đó i là giá trị mốc, j là các giá trị còn lại.

ANSWER:

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        for (int i = 0; i < nums.size(); i++){// chọn 1 giá trị làm mốc
            for (int j = i +1; j <= nums.size() - 1; j++){// chạy lần lượt các giá trị còn lại qua vòng lặp j
                if ( target - nums[i] == nums[j] ){// kiểm tra điều kiện
                return vector<int>{i, j}; // trả về 2 phần tử đầu tiên thỏa điều kiện
                break;// dừng lại ngay khi tìm được
                }
            }
        }
        return vector<int>{-1, -1};// return phòng hờ không có giá trị trả về
    }
};
```

2) *Given an integer x, return true if x is a **palindrome**, and false otherwise.*

ANSWER:

```cpp
class Solution {
public:
    bool isPalindrome(int x) {
        int rev; // gọi 1 biến để chứa các số dư
        long long res =0;// gọi 1 hàm đựng kết quả của số đã đảo ngược
        int temp = x; // gọi temp để sao chép giá trị x
        if (x < 0) return false;
        while (temp > 0){ // cho temp chay
            rev = temp %10;
            res = res * 10 + rev;
            temp = temp/10;
        }
        return res == x; // Khi trả về phải so sánh với giá trị ban đầu (== x)
    }
};
```

3) *Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1.*

Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2. Elements that do not appear in arr2 should be placed at the end of arr1 in ascending order.

Input: arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

Output: [2,2,2,1,4,3,3,9,6,7,19]

ANSWER:

```cpp
class Solution {
public:
    vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
        vector <int> remaining; // cho 1 vector chứa các phần tử không tồn tại trong arr2
        vector <int> result;// cho 1 vector chứa kết quả các số đã tồn tại

        for (int i = 0; i < arr2.size(); i++){ // chạy vòng lặp để xét từng phần tử arr2 so với tất cả các phần tử arr1
            for ( int j = 0; j < arr1.size(); j ++){
                if (arr2[i] == arr1[j]){ // xét phần tử đã tồn tại
                    int res = arr1[j];
                    result.push_back(res); // push_back vào vector result
                }
            }
        }
```

```
        // vòng lặp thứ 2 tìm những phần tử không trùng nhau
        for (int j = 0; j < arr1.size(); j++){// chạy vòng lặp để xét từng phần tử arr1 so với tất cả các phần tử arr2
            bool flag = false; // đặt cờ để kiểm tra tìm thấy phần tử chưa
            for (int i = 0; i < arr2.size(); i++ ){
                if ( arr1[j]== arr2[i]){
                    flag = true; // cờ trả về = tìm thấy
                    break;// bỏ qua vòng lặp luôn vì giá trị đã bị trùng
                }
            }
            if (!flag){
                remaining.push_back(arr1[j]); // push dô mảng
            }
        }
        sort(remaining.begin(), remaining.end()); // sắp xếp lại theo thứ tự
        for (int x : remaining) result.push_back(x); // chạy vòng lặp để đưa các phần tử thuộc remaining vào result
        return result;
    }
};
```