**Sắp xếp**

void InsertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++)

    {

        key = arr[i];

        j = i - 1;

        while (j >= 0 && arr[j] > key)

        {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

void BubbleSort(int arr[], int n)

{

    int i, j;

    for (i = 0; i < n - 1; i++)

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

            if (arr[j] > arr[j + 1])

                swap(arr[j], arr[j + 1]);

}

void SelectionSort(int arr[], int n)

{

    int i, j, min\_idx;

    for (i = 0; i < n - 1; i++)

    {

        min\_idx = i;

        for (j = i + 1; j < n; j++)

            if (arr[j] < arr[min\_idx])

                min\_idx = j;

        if (min\_idx != i)

            swap(arr[min\_idx], arr[i]);

    }

}

void QuickSort(int arr[], int start, int end)

{

    int pivot, i, j, mid;

    if (start < end)

    {

        mid = (start + end) / 2;

        swap(arr[start], arr[mid]);

        pivot = arr[start];

        i = start + 1;

        j = end;

        while (i <= j)

        {

            while (i <= end && arr[i] <= pivot)

                i++;

            while (j >= start && arr[j] > pivot)

                j--;

            if (i < j)

                swap(arr[i], arr[j]);

        }

        swap(arr[start], arr[j]);

        QuicksortV2(arr, start, j - 1);

        QuicksortV2(arr, j + 1, end);

    }

}

**Tìm kiếm**

int BinarySearch(int \*a, int n, int x)

{

    int left = 0, right = n - 1;

    while (left <= right)

    {

        int mid = (left + right) / 2;

        if (a[mid] == x)

        {

            return mid;

        }

        else if (a[mid] > x)

        {

            right = mid - 1;

        }

        else if (a[mid] < x)

        {

            left = mid + 1;

        }

    }

    return -1;

}

int SequentialSearch(int \*a, int n, int x)

{

    for (int i = 0; i < n - 1; i++)

    {

        if (x == a[i])

        {

            return i;

        }

    }

    return -1;

}

**Bài toán với dãy số**

#include <iostream>

using namespace std;

class DaySo

{

public:

    int \*arr, size;

    DaySo()

    {

        this->arr = nullptr;

        this->size = 0;

    }

    DaySo(int \*arr, int size)

    {

        this->size = size;

        this->arr = arr;

    }

    ~DaySo()

    {

        this->size = 0;

        if (this->arr != nullptr)

            delete this->arr;

    }

    void Input()

    {

        int n;

        cin >> n;

        int \*arr;

        arr = (int \*)calloc(n, sizeof(int));

        for (int i = 0; i < n; i++)

        {

            cin >> \*(arr + i);

        }

        this->size = n;

        this->arr = arr;

    }

    void Display()

    {

        for (int i = 0; i < this->size; i++)

            cout << this->arr[i] << " ";

        cout << endl;

    }

    int Partition(int arr[], int start, int end)

    {

        int pivot = arr[start];

        int count = 0;

        for (int i = start + 1; i <= end; i++)

            if (arr[i] <= pivot)

                count++;

        int pivotIndex = start + count;

        swap(arr[pivotIndex], arr[start]);

        int i = start, j = end;

        while (i < pivotIndex && j > pivotIndex)

        {

            while (arr[i] <= pivot)

                i++;

            while (arr[j] > pivot)

                j--;

            if (i < pivotIndex && j > pivotIndex)

                swap(arr[i++], arr[j--]);

        }

        return pivotIndex;

    }

    void QuickSortExec(int arr[], int start, int end)

    {

        if (start >= end)

            return;

        int p = Partition(arr, start, end);

        QuickSortExec(arr, start, p - 1);

        QuickSortExec(arr, p + 1, end);

    }

void QuickSort()

    {

        QuickSortExec(this->arr, 0, this->size);

    }

    void BinarySearch(int x)

    {

        int left = 0, right = size - 1;

        while (left <= right)

        {

            int mid = (left + right) / 2;

            if (arr[mid] == x)

                cout << "Phan tu dang o vi tri thu: " << mid + 1 << endl;

            else if (arr[mid] > x)

                right = mid - 1;

            else if (arr[mid] < x)

                left = mid + 1;

        }

        cout << "Khong tim thay phan tu nao" << endl;

    }

};

int main()

{

    int arr[] = {85, 76, 35, 43, 14, 121, 67};

    int size = 7;

    DaySo ds = DaySo(arr, size);

    cout << "Day so ban dau: " << endl;

    ds.Display();

    ds.BinarySearch(35);

    cout << "Day so sau khi sap xep: " << endl;

    ds.QuickSort();

    ds.Display();

}

**Bài toán với bảng băm**

#include <iostream>

#include <vector>

using namespace std;

class HashTableEntry

{

public:

    int key;

    int value;

    HashTableEntry(int key, int value)

    {

        this->key = key;

        this->value = value;

    }

};

class HashMapTable

{

public:

    vector<HashTableEntry> \*table;

    int N;

    HashMapTable(int N)

    {

        this->N = N;

        table = new vector<HashTableEntry>[N];

    }

    int Hash(int key)

    {

        return (key \* key + key + 1) % N;

    }

    void Insert(int key, int value)

    {

        int hKey = Hash(key);

        vector<HashTableEntry>::iterator hte;

        for (hte = table[hKey].begin(); hte != table[hKey].end(); hte++)

        {

            if (hte->key == key)

            {

                cout << "Key da ton tai, vui long them key khac!" << endl;

                break;

            }

        }

        if (hte == table[hKey].end())

        {

            HashTableEntry hte = HashTableEntry(key, value);

            table[hKey].push\_back(hte);

            cout << "Them thanh cong!" << endl;

        }

    }

    void Display()

    {

        for (int i = 0; i < N; i++)

        {

            cout << i;

            for (auto hte : table[i])

                cout << " --> "

                     << "key: " << hte.key << " value: " << hte.value;

            cout << endl;

        }

    }

    void Search(int key)

    {

        int hKey = Hash(key);

        vector<HashTableEntry>::iterator hte;

        for (hte = table[hKey].begin(); hte != table[hKey].end(); hte++)

        {

            if (hte->key == key)

            {

                cout << hte->value << endl;

                cout << "Tim kiem thanh cong!" << endl;

                break;

            }

        }

        if (hte == table[hKey].end())

        {

            cout << "Tim kiem that bai!" << endl;

        }

    }

    void Remove(int key)

    {

        int hKey = Hash(key);

        vector<HashTableEntry>::iterator hte;

        for (hte = table[hKey].begin(); hte != table[hKey].end(); hte++)

        {

            if (hte->key == key)

            {

                table[hKey].erase(hte);

                cout << "Xoa thanh cong!" << endl;

                break;

            }

        }

        if (hte == table[hKey].end())

        {

            cout << "Xoa that bai!" << endl;

        }

    }

};

**Bài toán với danh sách liên kết**

#include <iostream>

using namespace std;

class Employee

{

public:

    int id;

    string name;

    int yearOfBirth;

    float salary;

    Employee()

    {

    }

    Employee(int id, string name, int yearOfBirth, float salary)

    {

        this->id = id;

        this->name = name;

        this->yearOfBirth = yearOfBirth;

        this->salary = salary;

    }

    ~Employee()

    {

    }

    friend istream &operator>>(istream &is, Employee &e)

    {

        cout << "Nhap ma: " << endl;

        is >> e.id;

        cout << "Nhap ho ten: " << endl;

        cin.ignore();

        getline(is, e.name);

        cout << "Nhap nam sinh: " << endl;

        is >> e.yearOfBirth;

        cout << "Nhap luong " << endl;

        is >> e.salary;

        return is;

    }

    friend ostream &operator<<(ostream &os, Employee &e)

    {

        os << "Ma nhan vien: " << e.id << endl;

        os << "\tHo ten: " << e.name << endl;

        os << "\tNam sinh: " << e.yearOfBirth << endl;

        os << "\tLuong: " << e.salary << endl;

        return os;

    }

    int getId()

    {

        return id;

    }

    int setId(int id)

    {

        this->id = id;

    }

};

class Node

{

public:

    Employee employee;

    Node \*next;

    Node(Employee employee = Employee())

    {

        this->employee = employee;

        this->next = nullptr;

    }

    Node(Employee employee, Node \*next)

    {

        this->employee = employee;

        this->next = next;

    }

};

class Manage

{

public:

    int size;

    Node \*head;

    Node \*tail;

    Manage()

    {

        size = 0;

        this->head = nullptr;

        this->tail = nullptr;

    }

    void Input()

    {

        cin >> size;

        for (int i = 0; i < size; i++)

        {

            Employee employee = Employee();

            cin >> employee;

            this->InsertLast(employee);

        }

    }

    void Display()

    {

        Node \*temp = this->head;

        while (temp != nullptr)

        {

            cout << temp->employee;

            temp = temp->next;

        }

    }

    void InsertFirst(Employee employee)

    {

        Node \*newNode = new Node(employee);

        if (this->head == nullptr)

        {

            this->head = newNode;

        }

        else

        {

            newNode->next = this->head;

            this->head = newNode;

        }

    }

    void InsertLast(Employee employee)

    {

        Node \*newNode = new Node(employee);

        if (this->head == nullptr)

        {

            this->head = newNode;

            this->tail = newNode;

        }

        else

        {

            this->tail->next = newNode;

            this->tail = newNode;

        }

    }

void GetListBySalaryBigger(int salary){

        Node \*temp = head;

        while (temp != nullptr)

        {

            if(temp->employee.salary > salary){

                cout << temp->employee;

            }

            temp = temp->next;

        }

    }

    void Remove(int id)

    {

        Node \*temp = this->head;

        if (temp != nullptr && temp->employee.getId() == id)

        {

            this->head = temp->next;

            delete temp;

            return;

        }

        else

        {

            Node \*curr = this->head;

            while (temp != nullptr && temp->employee.getId() != id)

            {

                curr = temp;

                temp = temp->next;

            }

            if (!temp)

            {

                cout << "Khong tim thay nhan vien can xoa!" << endl;

                return;

            }

            curr->next = temp->next;

            delete temp;

        }

    }

};

int main()

{

    Manage m;

    m.Input();

    m.Display();

    m.GetListBySalaryBigger(1);

    m.Remove(123);

    m.Remove(1);

    m.Display();

    return 0;

}

**Bài toán với dãy điểm**

#include <iostream>

using namespace std;

class Point

{

public:

    int x, y;

    Point()

    {

        this->x = 0;

        this->y = 0;

    }

    Point(int x, int y)

    {

        this->x = x;

        this->y = y;

    }

    ~Point()

    {

        this->x = 0;

        this->y = 0;

    }

    friend istream &operator>>(istream &is, Point &p)

    {

        is >> p.x;

        is >> p.y;

        return is;

    }

    friend ostream &operator<<(ostream &os, Point &p)

    {

        os << p.x << " " << p.y << endl;

        return os;

    }

};

class ListPoint

{

public:

    Point \*p;

    ListPoint()

    {

        this->p = nullptr;

    }

    ListPoint(int n)

    {

        this->p = new Point[n];

    }

    ~ListPoint()

    {

        if (p != nullptr)

        {

            delete p;

        }

    }

    void Input(int n)

    {

        for (int i = 0; i < n; i++)

        {

            cin >> this->p[i];

        }

    }

    void Output(int n)

    {

        for (int i = 0; i < n; i++)

        {

            cout << this->p[i];

        }

    }

    void InsertionSort(int n)

    {

        int i, currX, currY, j;

        for (i = 1; i < n; i++)

        {

            currX = this->p[i].x;

            currY = this->p[i].y;

            j = i - 1;

            while (j >= 0 && this->p[j].x > currX)

            {

                this->p[j + 1] = this->p[j];

                j = j - 1;

            }

            this->p[j + 1].x = currX;

            this->p[j + 1].y = currY;

        }

    }

};

int main(int argc, char const \*argv[])

{

    int size;

    cin >> size;

    ListPoint lp(size);

    lp.Input(size);

    lp.InsertionSort(size);

    lp.Output(size);

    return 0;

}