THUẬT TOÁN TÌM KIẾM

NỘI DUNG

/01 Tìm kiếm tuyến tính (Linear search)

/02 Tìm kiếm nhị phân (Binary search)

/03 Vị trí đầu tiên trong mảng tăng dần

/04 Vị trí cuối cùng trong mảng tăng dần

/05 Vị trí đầu tiên lớn hơn hoặc bằng X trong mảng tăng dần

/06 Vị trí cuối cùng lớn hơn hoặc bằng X trong mảng tăng dần

/07 LOWER_BOUND

/08 UPPER_BOUND

1.Tìm kiếm tuyến tính (Linear Search):

Ý tưởng: Duyệt tuần tự các phần tử trong mảng và so sánh giá trị cần tìm kiếm với từng phần tử trong mảng.

Các bài toán như tìm kiếm vị trí đầu tiên, cuối cùng, đếm số lần xuất hiện của phần tử trong mảng đều là biến đổi của thuật toán tìm kiếm tuyến tính.

Code

```
bool linearSearch(int a[], int n, int x){
    for(int i = 0; i < n; i++){
        if (x == a[i]){
            return true;
        }
    }
    return false;
}
```

Độ phức tạp: O(N)

2.Tìm kiếm nhị phân (Binary Search):

Ý tưởng: Tìm kiếm trong đoạn từ [left, right] của mảng, ở mỗi bước thuật toán tìm vị trí mid ở giữa đoạn left, right. Nếu phần tử cần tìm kiếm bằng phần tử ở vị trí mid thì kết luận là tìm thấy, nếu không ta có thể giảm 1 nửa đoạn tìm kiếm xuống và tiếp tục tìm kiếm ở bên trái hay bên phải của mid.

Đây là một thuật toán cực kì hiệu quả và quan trọng, khi học lập trình bạn cần nắm rõ thuật toán này.

Điều kiện áp dụng: Mảng đã được sắp xếp.

Code

```
bool 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 true;
        }
        else if(a[mid] < x){
            // Tìm kiếm ở bên phải
            left = mid + 1;
        }
        else{
            //Tìm kiếm ở bên trái
            right = mid - 1;
        }
    }
    return false;
}
```

Độ phức tạp: O(logN)

3. Vị trí đầu tiên trong mảng tăng dần:

Bài toán: Tìm vị trí đầu tiên của phần tử X trong mảng đã được sắp xếp

```
int firstPos(int a[], int n, int x){
    int res = -1, left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] == x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            right = mid - 1;
        }
        else if(a[mid] < x){
            left = mid + 1;
        }
        else{
            right = mid - 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

4. Vị trí cuối cùng trong mảng tăng dần:

Bài toán: Tìm vị trí cuối cùng của phần tử X trong mảng đã được sắp xếp

```
int lastPos(int a[], int n, int x){
    int res = -1, left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] == x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            left = mid + 1;
        }
        else if(a[mid] < x){
            left = mid + 1;
        }
        else{
            right = mid - 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

5. Vị trí đầu tiên lớn hơn hoặc bằng X trong mảng tăng dần:

Bài toán: Tìm vị trí đầu tiên của phần tử lớn hơn hoặc bằng X trong mảng đã được sắp xếp

```
int lower(int a[], int n, int x){
    int res = -1;
    int left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] >= x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            right = mid - 1;
        }
        else{
            left = mid + 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

6. Vị trí cuối cùng nhỏ hơn hoặc bằng X trong mảng tăng dần:

Bài toán: Tìm vị trí cuối cùng của phần tử nhỏ hơn hoặc bằng X trong mảng đã được sắp xếp

```
int upper(int a[], int n, int x){
    int res = -1;
    int left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;
        if(a[mid] <= x){
            res = mid; // cập nhật
            //Tìm thêm đáp án tốt hơn
            left = mid + 1;
        }
        else{
            right = mid - 1;
        }
    }
    return res;
}
```

Độ phức tạp: O(logN)

7. LOWER_BOUND:

CÚ PHÁP

```
lower_bound(first_iter, last_iter, X);
```

SỬ DỤNG LOWER_BOUND

Lower_bound là một hàm tương tự như mục 5 ở trên, nó có thể áp dụng cho mảng, vector, set, map.

Điều kiện áp dụng: Mảng, vector đã được sắp xếp tăng dần

Tương tự như hàm sort, lower_bound cũng trả về iterator chứ không trả về giá trị.

Nó trả về vị trí đầu tiên của phần tử lớn hơn hoặc bằng X, nếu trong mảng, vector bạn tìm kiếm không có phần tử lớn hơn hoặc bằng X (tất cả đều nhỏ hơn X) thì lower_bound trả về last_iter

Ví dụ lower_bound với mảng:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a, a + 7, 3);
    cout << *it << endl;
    cout << (it - a) << endl;
}
```

OUTPUT: 3
2

Giải thích: Phần tử đầu tiên lớn hơn hoặc bằng 3 trong mảng là 3 ở chỉ số 2

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a, a + 7, 8);
    if(it == a + 7){
        cout << "NOT FOUND";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a) << endl;
}
```

OUTPUT: NOT FOUND
7

Ví dụ lower_bound với vector:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 3);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: 3
2

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 8);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: NOT FOUND
7

Dự đoán Output của các chương trình sau

Quiz 1

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 2);
    --it;
    cout << *it << endl;
}
```

Quiz 2

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 7);
    --it;
    cout << *it << endl;
}
```

Dự đoán Output của các chương trình sau

Quiz 3

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 7);
    it -= 2;
    cout << *it << endl;
}
```

Quiz 4

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = lower_bound(a.begin(), a.end(), 0);
    it--;
    cout << *it << endl;
}
```

8. UPPER_BOUND:

CÚ PHÁP

```
upper_bound(first_iter, last_iter, X);
```

SỬ DỤNG UPPER_BOUND

Điều kiện áp dụng: Mảng, vector đã được sắp xếp tăng dần

Upper_bound trả về vị trí đầu tiên của phần tử lớn X, nếu trong mảng, vector bạn tìm kiếm không có phần tử lớn hơn hoặc bằng X (tất cả đều nhỏ hơn hoặc bằng X) thì upper_bound trả về last_iter.

Ví dụ upper_bound với mảng:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a, a + 7, 3);
    cout << *it << endl;
    cout << (it - a) << endl;
}
```

OUTPUT: 4
5

Giải thích: Phần tử đầu tiên lớn hơn 3 trong mảng là 4 ở chỉ số 5

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[7] = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a, a + 7, 8);
    if(it == a + 7){
        cout << "NOT FOUND";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a) << endl;
}
```

OUTPUT: NOT FOUND
7

Ví dụ upper_bound với vector:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 3);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: 4
5

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 2, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 6);
    if(it == a.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << *it << endl;
    }
    cout << (it - a.begin()) << endl;
}
```

OUTPUT: NOT FOUND
7

Dự đoán Output của các chương trình sau

Quiz 1

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 2);
    --it;
    cout << *it << endl;
}
```

Quiz 2

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int> a = {1, 1, 3, 3, 3, 4, 6};
    auto it = upper_bound(a.begin(), a.end(), 6);
    --it;
    cout << *it << endl;
}
```

Dự đoán Output của các chương trình sau

Quiz 3

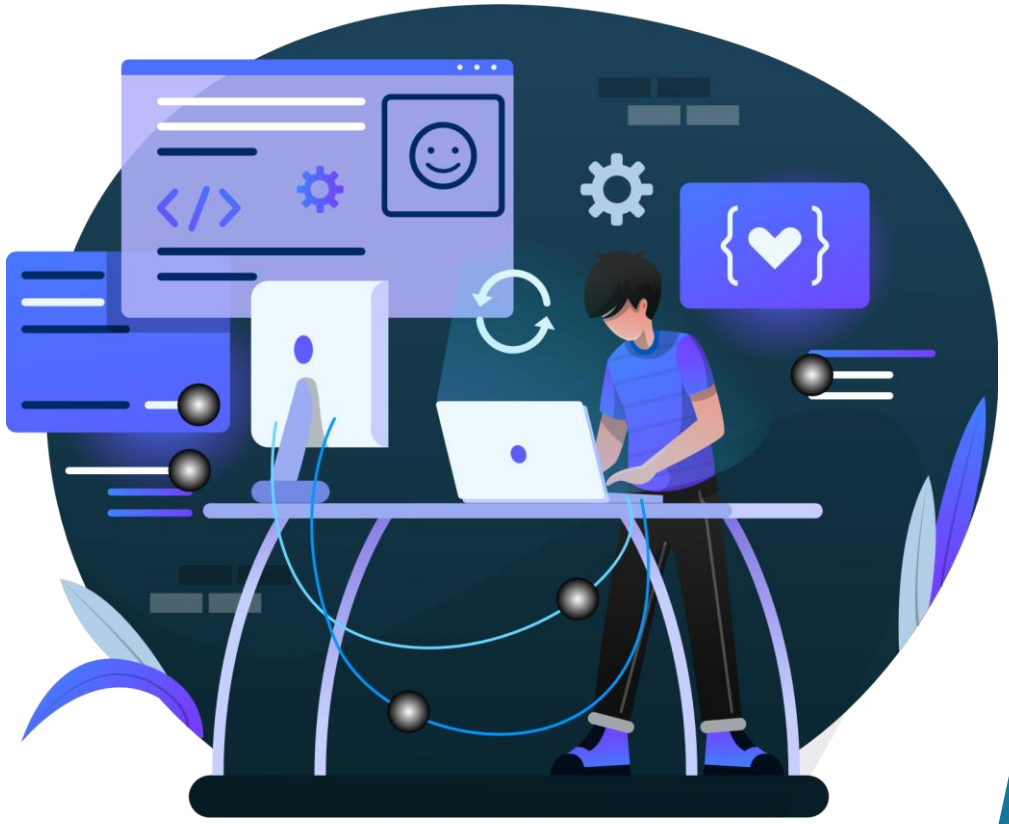
```
#include <bits/stdc++.h>
using namespace std;

int main(){
   vector<int> a = {1, 1, 3, 3, 3, 4, 6};
   auto it = upper_bound(a.begin(), a.end(), 6);
   it -= 2;
   cout << *it << endl;
}
```

Quiz 4

```
#include <bits/stdc++.h>
using namespace std;

int main(){
   vector<int> a = {1, 1, 3, 3, 3, 4, 6};
   auto it = upper_bound(a.begin(), a.end(), 0);
   it--;
   cout << *it << endl;
}
```



THUẬT TOÁN SẮP XẾP

Thuật toán sắp xếp:

Sắp xếp là một thuật toán quan trọng, được sử dụng cực kì nhiều trong các ứng dụng thực tế. Các bạn có thể không biết các thuật toán như quy hoạch động, chia và trị nhưng bắt buộc phải nắm được sắp xếp và tìm kiếm.

1.Thuật toán sắp xếp chọn (Selection sort):

```
void selectionSort(int a[], int n){
    for(int i = 0; i < n; i++){
        int min_pos = i;
        for (int j = i + 1; j < n; j++){
            if (a[j] < a[min_pos]){
                min_pos = j;
            }
        }
        swap(a[min_pos], a[i]);
    }
}
```

Độ phức tạp: O(N^2)

2.Thuật toán sắp xếp nổi bọt (Bubble sort):

```
void bubbleSort(int a[], int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n - i - 1; j++){
            if (a[j] > a[j + 1]){
                swap(a[j], a[j + 1]);
            }
        }
    }
}
```

Độ phức tạp: O(N^2)

3.Thuật toán sắp xếp chèn (Insertsion sort):

```
void insertionSort(int a[], int n){
    for(int i = 1; i < n; i++){
        int pos = i - 1, x = a[i];
        while(pos >= 0 && a[pos] > x){
            a[pos + 1] = a[pos];
            --pos;
        }
        a[pos + 1] = x;
    }
}
```

Độ phức tạp: O(N^2)

4. Thuật toán sắp xếp đếm phân phối (Counting sort):

Điều kiện áp dụng: Có thể khai báo được mảng đếm có số lượng phần tử lớn hơn giá trị lớn nhất của phần tử trong mảng

```
int dem[1000001]; // 0 <= a[i] <= 10^6
void countingSort(int a[], int n){
    int K = -1e9;
    for(int i = 0; i < n; i++){
        dem[a[i]]++;
        K = max(K, a[i]);
    }
    for(int i = 0; i <= K; i++){
        if(dem[i]){
            for(int j = 0; j < dem[i]; j++){
                cout << i << ' ';
            }
        }
    }
}
```

Độ phức tạp: O(N^2)

5. Thuật toán sắp xếp trộn (Merge sort):

Độ phức tạp: O(NLogN)

Thao tác trộn:

```
void merge(int a[], int l, int m, int r){
    int n1 = m - l + 1, n2 = r - m;
    int x[n1], y[n2];
    for(int j = l; j <= m; j++)
        x[j - l] = a[j];
    for(int j = m + 1; j <= r; j++)
        y[j - m - 1] = a[j];
    int i = 0, j = 0, cnt = l;
    while(i < n1 && j < n2){
        if(x[i] <= y[j])
            a[cnt++] = x[i++];
        else
            a[cnt++] = y[j++];
    }
    while(i < n1) a[cnt++] = x[i++];
    while(j < n2) a[cnt++] = y[j++];
}
```

Hàm merge soft và main:

```
void mergeSort(int a[], int l, int r){
    if(l < r){
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m , r);
    }
}
int main(){
    int n; cin >> n;
    int a[n];
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }
    mergeSort(a, 0, n - 1);
    for(int x : a) cout << x << ' ';
}
```

6. Thuật toán quick sort:

Độ phức tạp: O(NLogN)

Thao tác phân hoạch bằng Lomuto partition:

```
int partition(int a[], int l, int r){
    int pivot = a[r];
    int i = l - 1;
    for(int j = l; j < r; j++){
        if(a[j] <= pivot){
            ++i;
            swap(a[i], a[j]);
        }
    }
    ++i;
    swap(a[i], a[r]);
    return i;
}
```

Hàm quick soft và main:

```
void quicksort(int a[], int l, int r){
    if(l < r){
        int m = (l + r) / 2;
        int p = partition(a, l, r);
        quicksort(a, l, p - 1);
        quicksort(a, p + 1, r);
    }
}

int main(){
    int n; cin >> n;
    int a[n];
    for(int i = 0; i < n; i++) cin >> a[i];
    quicksort(a, 0, n - 1);
    for(int x : a) cout << x << ' ';
}
```

7. Hàm soft trong thư viện STL:

Thư viện STL cung cấp 2 hàm sort

sort

Hàm sort được cài đặt bằng Intro sort (kết hợp của quick sort và heap sort)

stable_sort

Hàm stable_sort được cài đặt bằng thuật toán merge sort nên có thêm tính chất stable

STL

7. Hàm soft trong thư viện STL:

CÚ PHÁP

Cú pháp áp dụng hàm sort với toàn bộ mảng, mặc định sẽ được sắp xếp theo thứ tự tăng dần về giá trị số và tăng dần về thứ tự từ điển về giá trị kí tự

```
//Sort mảng a có n phần tử
sort (a, a + n);
```

Sort theo thứ tự giảm dần, ta thêm tham số greater vào hàm sort, chú ý tới kiểu dữ liệu của mảng, trong trường hợp này mảng a là mảng int

```
sort(a, a + n, greater<int>());
```

7. Hàm soft trong thư viện STL:

Sort mảng a trong đoạn từ chỉ số x tới chỉ số y

```
sort(a + x, a + y + 1);
sort(a + x, a + y + 1, greater<int>());
```

Sort toàn bộ vector v

```
sort(v.begin(), v.end());
sort(v.begin(), v.end(), greater<int>());
```

Sort vector từ chỉ số x tới chỉ số y

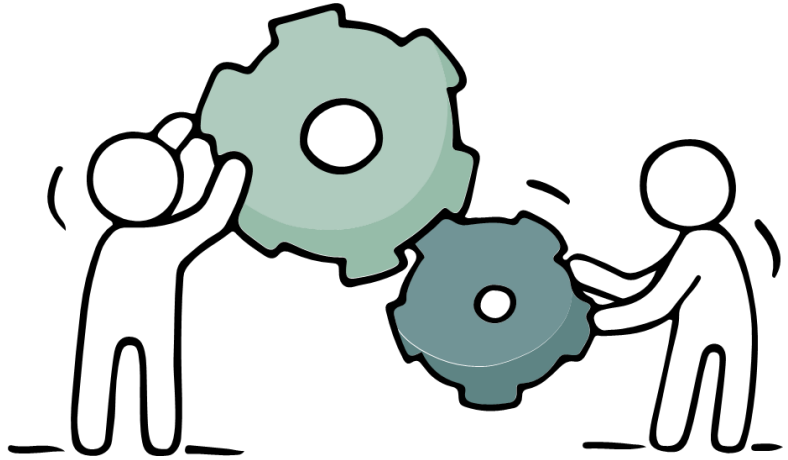
```
sort(v.begin() + x, v.begin() + y);
sort(v.begin() + x, v.begin() + y, greater<int>());
```

Đối với stable_sort các bạn áp dụng tương tự

8. Xây dựng hàm comparator cho hàm sort:

Nếu các bạn chỉ có thể sử dụng hàm sort để sắp xếp tăng dần và giảm dần thì sẽ không áp dụng nó vào các bài toán thực tế được.

Hàm comparator do người dùng tự xây dựng để thực hiện yêu cầu sắp xếp riêng theo từng bài toán thực tế.



Sắp xếp mảng tăng dần

Hàm comparator

```
#include <bits/stdc++.h>
using namespace std;

bool cmp(int a, int b){
    if (a < b)
        return true;
    else
        return false;
    //return a < b;
}
```

Hàm main:

```
int main(){
    int n; cin >> n;
    int a[1000];
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }
    sort(a, a + n, cmp);
    for(int i = 0; i < n; i++){
        cout << a[i] << ' ';
    }
}
```

Sắp xếp mảng giảm dần

Hàm comparator

```
#include <bits/stdc++.h>
using namespace std;

bool cmp(int a, int b){
    if (a > b)
        return true;
    else
        return false;
    //return a > b;
}
```

Hàm main:

```
int main(){
    int n; cin >> n;
    int a[1000];
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }
    sort(a, a + n, cmp);
    for(int i = 0; i < n; i++){
        cout << a[i] << ' ';
    }
}
```

Sắp xếp theo trị tuyệt đối tăng dần

Hàm comparator

```
#include <bits/stdc++.h>
using namespace std;

bool cmp(int a, int b){
    if (abs(a) < abs(b))
        return true;
    else
        return false;
    //return abs(a) < abs(b);
}
```

Hàm main:

```
int main(){
    int n = 5;
    int a[5] = {-9, 2, 4, -3, 1};
    sort(a, a + n, cmp);
    for(int x : a)
        cout << x << ' ';
}
```

OUTPUT: 1 2 -3 4 -9

Sắp xếp theo tổng chữ số tăng dần

Hàm comparator

```
#include <bits/stdc++.h>
using namespace std;

int sum(int n){
    int res = 0;
    while(n){
        res += n % 10; n /= 10;
    }
    return res;
}
bool cmp(int a, int b){
    if(sum(a) < sum(b))
        return true;
    else return false;
}
```

Hàm main:

```
int main(){
    int n = 5;
    int a[5] = {22, 1, 4, 9, 33};
    sort(a, a + n, cmp);
    for(int x : a)
        cout << x << ' ';
}
```

OUTPUT: 1 22 4 33 9

Sắp xếp theo tổng chữ số tăng dần, nếu 2 số có cùng tổng chữ số thì số nhỏ hơn sẽ xếp trước

Hàm comparator

```
#include <bits/stdc++.h>
using namespace std;

int sum(int n){
    int res = 0;
    while(n){
        res += n % 10; n /= 10;
    }
    return res;
}
bool cmp(int a, int b){
    if(sum(a) != sum(b))
        return sum(a) < sum(b);
    else return a < b;
}
```

Hàm main:

```
int main(){
    int n = 5;
    int a[5] = {22, 6, 4, 9, 33};
    sort(a, a + n, cmp);
    for(int x : a)
        cout << x << ' ';
}
```

OUTPUT: 4 22 6 33 9

Sắp xếp theo tổng chữ số tăng dần, nếu 2 số có cùng tổng chữ số thì số nào được nhập trước sẽ xếp trước (Dùng stable_sort)

Hàm comparator

```
#include <bits/stdc++.h>
using namespace std;

int sum(int n){
    int res = 0;
    while(n){
        res += n % 10; n /= 10;
    }
    return res;
}
bool cmp(int a, int b){
    return sum(a) < sum(b);
}
```

Hàm main:

```
int main(){
    int n = 5;
    int a[5] = {22, 6, 4, 9, 33};
    stable_sort(a, a + n, cmp);
    for(int x : a)
        cout << x << ' ';
}
```

OUTPUT: 22 4 6 33 9