LECTURE 03

SORTING

Big-O Coding

Website: www.bigocoding.com

Sorting

Sorting (Sắp xếp): là quá trình bố trí lại các phần tử trong danh sách (mảng, ma trận,...) theo một thứ tự tăng dần hoặc giảm dần.

Có nhiều thuật toán sắp xếp. Mỗi thuật toán sẽ có thời gian thực hiện nhanh chậm khác nhau, không gian vùng nhớ dùng cho việc sắp xếp cũng khác nhau.

```
C++: sort
Java: Arrays.sort hoặc Collections.sort
Python: list.sort hoặc sorted
```

Độ phức tạp các thuật toán sắp xếp

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

Khai báo & sử dụng

Thư viện:

```
#include <algorithm>
using namespace std;
```

```
sort(iterator1, iterator2, option);
```

Trong đó:

- iterator1: Con trỏ vị trí bắt đầu sắp xếp.
- iterator2: Con trỏ vị trí kết thúc sắp xếp.
- option: Hàm định nghĩa cách thức so sánh giữa hai phần tử, dùng để xác định tiêu chuẩn sắp xếp.

Sử dụng list.sort()

```
list.sort([cmp[, key[, reverse]]])
```

Sử dụng sorted()

```
sorted(iterable[, cmp[, key[, reverse]]])
```

Trong đó:

- list.sort(): Sắp xếp trên vùng nhớ của list, không trả về giá trị, chỉ có thể sắp xếp toàn bộ list, nên sử dụng khi muốn sắp xếp toàn bộ list.
- sorted(): Sắp xếp dữ liệu trong vùng iterable và trả về một list mới, có thể sắp xếp một sublist.

Khai báo & sử dụng

Thư viện:

```
import java.util.Collections;
import java.util.Arrays;
import java.util.Comparator;
```

```
Arrays.sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c)
Collections.sort(List<T> a, Comparator<? super T> c)
```

Trong đó:

- a: Mảng dữ liệu cần sắp xếp (bắt buộc).
- fromIndex, toIndex: Sắp xếp dữ liệu thuộc đoạn [fromIndex, toIndex).
- c: Hàm định nghĩa cách thức so sánh giữa hai phần tử, dùng để xác định tiêu chuẩn sắp xếp.

Sắp xếp tăng dần (mặc định)

0	1	2	3	4
5	7	8	3	6


```
sort(v.begin(), v.end());
```

```
v.sort()
```

Hoặc

```
v = sorted(v)
```


```
Arrays.sort(v);
```

0	1	2	3	4
3	5	6	7	8

Sắp xếp tăng dần (viết hàm/tham số)

0	1	2	3	4
5	7	8	3	6

Sắp xếp tăng dần, viết hàm so sánh option.

```
sort(v.begin(), v.end(), option);
```

```
bool option(int a, int b)
{
    return a < b;
}
```

Sắp xếp tăng dần sử dụng tham số reverse.

```
v.sort(reverse=False)
```

Hoặc:

```
v = sorted(v, reverse=False)
```

0	1	2	3	4
3	5	6	7	8

Sắp xếp tăng dần (viết hàm/tham số)

0	1	2	3	4
5	7	8	3	6

```java
Arrays.sort(a, new Comparator<Integer>() {
    @Override
    public int compare(Integer o1, Integer o2) {
        return o1.compareTo(o2);
    }
});
// chỉ dùng được với object type
```

0	1	2	3	4
3	5	6	7	8

Sắp xếp giảm dần (greater/key)

0	1	2	3	4
5	7	8	3	6

Sắp xếp giảm dần, sử dụng greater trong thư viện <functional>.

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

Sắp xếp giảm dần sử dụng tham số key.

```
v.sort(key=lambda x: -x)
```

Hoặc:

```
v = sorted(v, key=lambda x: -x)
```

0	1	2	3	4
8	7	6	5	3

Sắp xếp giảm dần (viết hàm/tham số)

0	1	2	3	4
5	7	8	3	6

C++

Sắp xếp giảm dần, sử dụng hàm so sánh option.

```
sort(v.begin(), v.end(), option);
```

```
bool option(int a, int b)
{
    return a > b;
}
```

python

Sắp xếp giảm dần sử dụng tham số reverse.

```
v.sort(reverse=True)
```

Hoặc:

```
v = sorted(v, reverse=True)
```

0	1	2	3	4
8	7	6	5	3

Sắp xếp giảm dần (viết hàm/tham số)

Có 3 cách để viết một hàm sắp xếp giảm dần trong Java:

0	1	2	3	4
5	7	8	3	6

Cách 1: Sử dụng Collections.reverseOrder().

```
Arrays.sort(v, Collections.reverseOrder());
```

Cách 2: Tạo một class Comparator.

```
Arrays.sort(v, new Comparator<T>() {
    @Override
    public int compare(T o1, T o2) {
        return o2.compareTo(o1);
    }
} );
// thay T bằng kiểu dữ liệu tương ứng
```

Sắp xếp giảm dần (viết hàm/tham số)

0	1	2	3	4
5	7	8	3	6

Cách 3: Sử dụng lambda function (chỉ dùng trong Java 8).

```
Arrays.sort(v, (o1, o2) -> o2.compareTo(o1));
```

0	1	2	3	4
8	7	6	5	3

**Lưu ý:* *chỉ sử dụng được với object data không sử dụng được với primitive data.*

SỬ DỤNG HÀM SORT ĐỂ SẮP XẾP MẢNG CON

Sắp xếp mảng con

Trong C++ nếu muốn sắp xếp một đoạn các phần tử trong vector, bạn có thể dùng iterator để chỉ ra 2 vị trí đầu & cuối cần sắp xếp trong mảng.

0	1	2	3	4
5	7	8	3	6

```
sort(v.begin()+1, v.begin()+4);
```

0	1	2	3	4
5	3	7	8	6

Sắp xếp mảng con

Trong python nếu muốn sắp xếp các phần tử có chỉ số thuộc đoạn [first, last) trong mảng, thì sử dụng hàm sorted theo cú pháp:

<variable>[first,last] = sorted(<variable>[first:last][,cmp=...])

0	1	2	3	4
5	7	8	3	6

```
v[1:4] = sorted(v[1:4])
```

0	1	2	3	4
5	3	7	8	6

Sắp xếp mảng con

Trong Java nếu muốn sắp xếp các phần tử có chỉ số thuộc đoạn [fromIndex, toIndex) của **mảng**, thì ta chỉ việc truyền 2 tham số fromIndex, toIndex cho hàm và tham số Comparator nếu cần. Lưu ý chỉ sử dụng được trên mảng, List trong Java không cho phép sắp xếp một đoạn con.

0	1	2	3	4
5	7	8	3	6

```
Arrays.sort(v, 1, 4);
```

0	1	2	3	4
5	3	7	8	6

SỬ DỤNG HÀM SORT TRONG MẢNG CẤU TRÚC

Khai báo cấu trúc phân số

```cpp
struct Fraction
{
    int num;
    int denom;
};
```

```cpp
vector <Fraction> v;
v.push_back(Fraction{ 5, 4 });
v.push_back(Fraction{ 7, 9 });
v.push_back(Fraction{ 1, 8 });
v.push_back(Fraction{ 9, 2 });
v.push_back(Fraction{ 12, 8 });
```

```python
class Fraction:
    def __init__(self, num, denom):
        self.num = num
        self.denom = denom
```

```python
v = []
v.append(Fraction(5, 4))
v.append(Fraction(7, 9))
v.append(Fraction(1, 8))
v.append(Fraction(9, 2))
v.append(Fraction(12, 8))
```

0	1	2	3	4
5/4	7/9	1/8	9/2	12/8

Khai báo cấu trúc phân số

Trong Java cấu trúc bỏ vào Class. Để sắp xếp đối với class trong java cũng tương tự như sắp xếp giảm dần, ta có một số cách sau:

- Tạo class đối tượng là lớp kế thừa interface Comparable.
- Tạo một class ObjectComparator kế thừa interface Comparator phụ để thực hiện so sánh.
- So sánh bằng hàm lambda (Java 8)

```
class Fraction {
    public Integer num;
    public Integer denom;
    Fraction(int numerator, int denominator) {
        num = numerator;
        denom = denominator;
    }
}
```

Khai báo cấu trúc phân số


```
ArrayList<Fraction> v = new ArrayList<Fraction>();
v.add(new Fraction(5, 4));
v.add(new Fraction(7, 9));
v.add(new Fraction(1, 8));
v.add(new Fraction(9, 2));
v.add(new Fraction(12, 8));
```

0	1	2	3	4
5/4	7/9	1/8	9/2	12/8

Sắp xếp mảng cấu trúc tăng dần

0	1	2	3	4
5/4	7/9	1/8	9/2	12/8

C++

```
bool option(const Fraction &x, const
Fraction &y)
{
   return (double)x.num/x.denom <
          (double)y.num/y.denom;
}
```

```
sort(v.begin(), v.end(), option);
```

python

```
v.sort(key=lambda fraction:
fraction.num/fraction.denom)
```

0	1	2	3	4
1/8	7/9	5/4	12/8	9/2

Sắp xếp mảng cấu trúc tăng dần

0	1	2	3	4
5/4	7/9	1/8	9/2	12/8

Sử dụng kế thừa Comparator

```
Collections.sort(v, new Comparator<Fraction>() {
    @Override
    public int compare(Fraction o1, Fraction o2) {
        return ((Double)(1.0*o1.num/o1.denom))
        .compareTo((Double)(1.0*o2.num/o2.denom));
    }
});
```

0	1	2	3	4
1/8	7/9	5/4	12/8	9/2

Sắp xếp mảng cấu trúc giảm dần

0	1	2	3	4
5/4	7/9	1/8	9/2	12/8


```cpp
bool option(const Fraction &x, const
Fraction &y)
{
    return (double)x.num/x.denom >
           (double)y.num/y.denom;
}
```

```cpp
sort(v.begin(), v.end(), option);
```

```python
# Sử dụng key làm hàm so sánh
sắp xếp tăng dần.
# Dùng reverse đảo ngược mảng
thành giảm dần.
v.sort(key=lambda fraction:
fraction.num/fraction.denom,
       reverse=True)
```

0	1	2	3	4
9/2	12/8	5/4	7/9	1/8

Sắp xếp mảng cấu trúc giảm dần

0	1	2	3	4
5/4	7/9	1/8	9/2	12/8

Sử dụng hàm Lambda:

```
Collections.sort(v, (o1, o2) ->
        ((Double)(1.0*o2.numerator/o2.denominator))
        .compareTo((Double)(1.0*o1.numerator/o1.denominator)));
```

0	1	2	3	4
9/2	12/8	5/4	7/9	1/8

SẮP XẾP DỰA VÀO NHIỀU THÀNH PHẦN KHÁC NHAU TRONG STRUCT

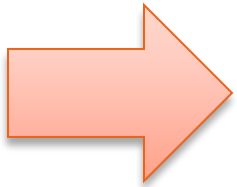
Cho danh sách **"Sinh viên",** mỗi sinh viên có Điểm (score) và Mã số sinh viên (ID). Sắp xếp danh sách này theo 2 tiêu chí sau:

- Sinh viên nào có điểm cao hơn thì sẽ xếp trên.
- Nếu điểm bằng nhau thì sinh viên nào có mã số nhỏ hơn sẽ xếp trên.

Sắp xếp cấu trúc sinh viên

Cho bảng điểm như sau, sắp xếp sinh viên nào điểm (Score) cao hơn thì sẽ xếp trên, nếu điểm bằng nhau thì mã số (ID) nhỏ hơn sẽ xếp trên.

ID	Score
100	8.5
101	7.5
102	8.5
103	10.0
104	10.0
105	4.5



ID	Score
103	10.0
104	10.0
100	8.5
102	8.5
101	7.5
105	4.5

Sắp xếp cấu trúc sinh viên

Cách 1: Viết thêm 1 function phụ để sắp xếp theo tiêu chí đưa ra.

```cpp
struct Student
{
    int id;
    double score;
};
```


```cpp
bool option(const& Student A, const& Student B)
{
    if (A.score > B.score || (A.score == B.score && A.id < B.id))
        return true;
    return false;
}
```

Sắp xếp cấu trúc sinh viên

```cpp
int main()
{
    vector<Student> list_student;
    list_student.push_back({ 100, 8.5 });
    list_student.push_back({ 101, 7.5 });
    list_student.push_back({ 102, 8.5 });
    list_student.push_back({ 103, 10 });
    list_student.push_back({ 104, 10 });
    list_student.push_back({ 105, 4.5 });
    sort(list_student.begin(), list_student.end(), option);
    for (int i = 0; i < list_student.size(); i++)
        cout << list_student[i].id << " " << list_student[i].score << endl;
    return 0;
}
```

Sắp xếp cấu trúc sinh viên

Cách 1: Viết thêm 1 function phụ để sắp xếp theo tiêu chí đưa ra.

```
class Student:
    def __init__(self, id = 0, score = 0):
        self.id = id
        self.score = score
```

Sắp xếp cấu trúc sinh viên

```python
if __name__ == '__main__':
    list_student = []
    list_student.append(Student(100, 8.5))
    list_student.append(Student(101, 7.5))
    list_student.append(Student(102, 8.5))
    list_student.append(Student(103, 10.0))
    list_student.append(Student(104, 10.0))
    list_student.append(Student(105, 4.5))
    # mặc định Python sắp tăng dần
    # để sắp điểm giảm dần thì ta đảo ngược dấu
    list_student.sort(key=lambda s: (-s.score, s.id))
    for student in list_student:
        print(student.id, student.score)
```

Sắp xếp cấu trúc sinh viên

Cách 1: Viết thêm 1 function phụ để sắp xếp theo tiêu chí đưa ra.


```java
class Student {
    public Integer id;
    public Double score;
    public Student(int id, double score) {
        this.id = id;
        this.score = score;
    }
}
class StudentCompare implements Comparator<Student> {
    @Override
    public int compare(Student o1, Student o2) {
        if (!o1.score.equals(o2.score))
            return o2.score.compareTo(o1.score);
        return o1.id.compareTo(o2.id);
    }
}
```

Sắp xếp cấu trúc sinh viên

```
public class Main {
    public static void main (String[] args) {
        ArrayList<Student> list_student = new ArrayList<Student>();
        list_student.add(new Student(100, 8.5));
        list_student.add(new Student(101, 7.5));
        list_student.add(new Student(102, 8.5));
        list_student.add(new Student(103, 10.0));
        list_student.add(new Student(104, 10.0));
        list_student.add(new Student(105, 4.5));
        Collections.sort(list_student, new StudentCompare());
        for (Student e : list_student) {
            System.out.println(e.id + " " + e.score);
        }
    }
}
```

Sắp xếp cấu trúc sinh viên

Cách 2: Viết 1 operator bên trong struct để sắp xếp theo tiêu chí đề ra.

```cpp
struct Student {
    int id;
    double score;
    bool operator < (const Student& B)
    {
        if (score > B.score || (score == B.score && id < B.id) )
            return true;
        return false;
    }
};
```

Sắp xếp cấu trúc sinh viên


```cpp
int main()
{
    vector<Student> list_student;
    list_student.push_back({ 100, 8.5 });
    list_student.push_back({ 101, 7.5 });
    list_student.push_back({ 102, 8.5 });
    list_student.push_back({ 103, 10 });
    list_student.push_back({ 104, 10 });
    list_student.push_back({ 105, 4.5 });
    sort(list_student.begin(), list_student.end());
    for (int i = 0; i < list_student.size(); i++)
        cout << list_student[i].id << " " << list_student[i].score << endl;
    return 0;
}
```

Sắp xếp cấu trúc sinh viên

Cách 2: Viết 1 operator bên trong class để sắp xếp theo tiêu chí đề ra.

```python
class Student:
    def __init__(self, id = 0, score = 0):
        self.id = id
        self.score = score
    def __lt__(self, other):
        if (self.score > other.score)
           or (self.score == other.score and self.id < other.id):
            return True
        return False
```

Sắp xếp cấu trúc sinh viên

```python
if __name__ == '__main__':
    list_student = []
    list_student.append(Student(100, 8.5));
    list_student.append(Student(101, 7.5));
    list_student.append(Student(102, 8.5));
    list_student.append(Student(103, 10.0));
    list_student.append(Student(104, 10.0));
    list_student.append(Student(105, 4.5));
    list_student.sort()
    for student in list_student:
        print(student.id, student.score)
```

Sắp xếp cấu trúc sinh viên

Cách 2: Viết 1 operator bên trong class để sắp xếp theo tiêu chí đề ra.

```java
class Student implements Comparable<Student> {
    public Integer id;
    public Double score;
    public Student(int id, double score) {
        this.id = id;
        this.score = score;
    }
    @Override
    public int compareTo(Student other) {
        if (!this.score.equals(other.score))
            return other.score.compareTo(this.score);
        return this.id.compareTo(other.id);
    }
}
```

Sắp xếp cấu trúc sinh viên (main)


```
public class Main {
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Student> list_student = new ArrayList<Student>();
        list_student.add(new Student(100, 8.5));
        list_student.add(new Student(101, 7.5));
        list_student.add(new Student(102, 8.5));
        list_student.add(new Student(103, 10.0));
        list_student.add(new Student(104, 10.0));
        list_student.add(new Student(105, 4.5));
        Collections.sort(list_student);
        for (Student e : list_student) {
            System.out.println(e.id + " " + e.score);
        }
    }
}
```

Sắp xếp mảng tĩnh tăng dần C++

Cho mảng tĩnh một chiều như sau:

	0	1	2	3	4
a	5	7	8	3	6

Tăng dần:

```
int n = 5;
int a[] = {5, 7, 8, 3, 6};
sort(a, a + n);
```

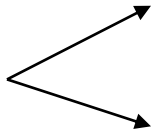
Giảm dần:

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

Các ngôn ngữ dùng thuật toán sắp xếp nào

Introsort

Timsort



Primitive type: Dual pivot Quicksort.

Object type: Tim sort.

Bài toán minh họa

CODEFORCES β
Sponsored by Telegram

Devu, the Dumb Guy

Có n môn học, môn thứ i sẽ có c_i chương. Devu mong muốn học hết tất cả các môn học.

Ở môn đầu tiên Devu học trong x giờ. Nhưng sau khi học xong môn đầu tiên môn tiếp theo cậu sẽ học nhanh hơn bình thường 1 giờ. Tuy vậy, mỗi chương Devu vẫn cần ít nhất 1 giờ để học qua các chương.

Nhiệm vụ của bạn là tính thời gian tối thiểu mà Devu cần để học xong n môn học đã cho.

Bài toán minh họa

CODEFORCES β
Sponsored by Telegram

Devu, the Dumb Guy

Input:

Dòng đầu tiên chứa hai số nguyên *n*, *x* ($1 \le n, x \le 10^5$). Trong đó, n là số lượng môn học, x là tốc độ để đọc một chương trong môn học đầu tiên.

Dòng tiếp theo chứa n số nguyên c_i ($1 \le c_i \le 10^5$) là số chương cần đọc của môn học thứ i.

Output:

In ra một số nguyên duy nhất là thời gian tối thiểu để Devu học xong n môn học.

Ví dụ:

4 2 5 1 2 1	10

Hướng dẫn giải

- **Bước 1:** Đọc toàn bộ dữ liệu đề bài vào cấu trúc dữ liệu chương trình.

```
4 2
5 1 2 1
```

O(n)

Khai báo biến n, x và mảng v để chứa dữ liệu đầu vào.

- **Bước 2:** Sắp xếp mảng lại tăng dần

0	1	2	3
1	1	2	5

O(nlogn)

- **Bước 3:** Xử lý phần nội dung chính đề bài yêu cầu.
  - Duyệt vòng lặp từ môn đầu tiên đến môn cuối cùng. O(n)
  - Môn đầu tiên sẽ học trong 1 (chương) x 2 (giờ) = 2 giờ, môn thứ hai 1 giờ, môn thứ ba 2 giờ, môn thứ tư 5 giờ.
  - Tổng thời gian học: 1 + 2 + 2 + 5 = 10.

Source Code Devu, the Dumb Guy


```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
long long selfLearning(vector<int>& subjects, int x)
{
    sort(subjects.begin(), subjects.end());
    long long min_time = 0;
    for (int subject : subjects)
    {
        min_time += 1LL * subject * x;
        if (x > 1)
            x--;
    }
    return min_time;
}
```

Source Code Devu, the Dumb Guy

```
17. int main()
18. {
19.     int n, x;
20.     cin >> n >> x;
21.     vector<int> subjects(n);
22.     for (int i = 0; i < n; i++)
23.         cin >> subjects[i];
24.     long long result = selfLearning(subjects, x);
25.     cout << result;
26.     return 0;
27. }
```

Source Code Devu, the Dumb Guy

```python
def selfLearning(subjects, x):
    subjects.sort()
    min_time = 0
    for subject in subjects:
        min_time += subject * x
        if x > 1:
            x -= 1
    return min_time
n, x = map(int, input().split())
subjects = list(map(int, input().split()))
result = selfLearning(subjects, x)
print(result)
```

Source Code Devu, the Dumb Guy

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    private static long selfLearning(Integer[] subjects, int x) {
        Arrays.sort(subjects);
        long min_time = 0;
        for (int subject : subjects) {
            min_time += 1L * subject * x;
            if (x > 1)
                x--;
        }
        return min_time;
    }
```

Source Code Devu, the Dumb Guy

```
15.     public static void main(String[] args) {
16.         Scanner sc = new Scanner(System.in);
17.         int n = sc.nextInt();
18.         int x = sc.nextInt();
19.         Integer[] subjects = new Integer[n];
20.
21.         for (int i = 0; i < n; i++) {
22.             subjects[i] = sc.nextInt();
23.         }
24.         long result = selfLearning(subjects, x);
25.         System.out.println(result);
26.     }
27. }
```

Hỏi đáp