Big-O Blue

LECTURE 03

SORTING

Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Sorting là gì?

Sorting là việc sắp xếp các phần tử trong danh sách (mảng, ma trận, cấu trúc...) theo một thứ tự tăng dần hoặc giảm dần.

Có nhiều thuật toán để sắp xếp mỗi thuật toán sẽ sắp xếp 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	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

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: Sắp tăng dần hoặc giảm dần.

Khai báo: sử dụng list.sort()

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

Khai báo: 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/list dữ liệu cần sắp xếp (bắt buộc).
- fromIndex: Vị trí bắt đầu của dãy con cần sắp xếp.
- toIndex: Vị trí sau vị trí kết thúc của dãy con cần sắp xếp.
- c: Hàm so sánh độ ưu tiên sắp xếp.

Sắp xếp tăng dần (mặc định)

0	1	2	3	4
5	7	8	3	6

Sắp xếp tăng dần (mặc định).

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

Sắp xếp tăng dần (mặc định).

```
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

C++

python

Sắp xếp tăng dần, sử dụng 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

```
Integer[] a = new Integer[] {5, 7, 8, 3, 6};
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

C++

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

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

python

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 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)

Cách 3: sử dụng lambda function (chỉ dùng trong java8).

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

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

SỬ DỤNG HÀM SORT ĐỂ SẮP XẾP MẢNG CON NHƯ THẾ NÀO?

Sắp xếp mảng con

Trong trường hợp 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

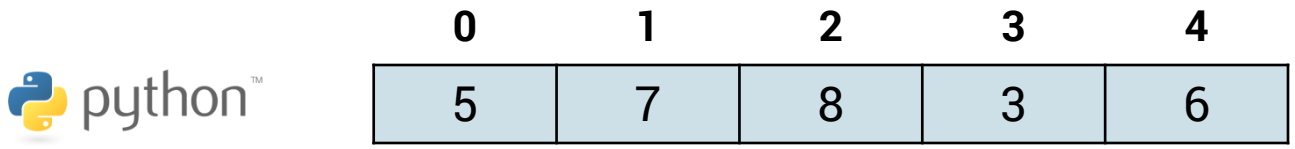
```
sort(v.begin()+1, v.begin()+4);
```

0	1	2	3	4
5	3	7	8	6

Sắp xếp mảng con

Trong trường hợp 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=...])




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

Sắp xếp mảng con

Như cú pháp đã mô tả ở slide 5. Trong trường hợp 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 NHƯ THẾ NÀO?

Khai báo cấu trúc phân số

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

C++

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

```
Fraction t0 = {5, 4};
Fraction t1 = {7, 9};
Fraction t2 = {1, 8};
Fraction t3 = {9, 2};
Fraction t4 = {12, 8};
vector <Fraction> f;
f.push_back(t0);
f.push_back(t1);
f.push_back(t2);
f.push_back(t3);
f.push_back(t4);
```

python

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

```
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))
```

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(f.begin(), f.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 giảm dần

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


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

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

```
# 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

Khai báo cấu trúc phân số

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

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 (java8)

```
class Fraction {
    public Integer numerator;
    public Integer denominator;

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

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

Sử dụng kế thừa Comparator

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

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

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 2 THÀNH PHẦN KHÁC NHAU TRONG STRUCT

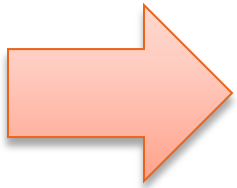
Cho danh sách **"Sinh viên",** mỗi sinh viên có Điểm (score) và Mã số sinh viên (ID), sắp xếp danh sách này theo 2 tiêu chí sau:

- 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


```
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;
    Student sv;
    sv.id = 100, sv.score = 8.5;
    list_student.push_back(sv);
    sv.id = 101, sv.score = 7.5;
    list_student.push_back(sv);
    sv.id = 102, sv.score = 8.5;
    list_student.push_back(sv);
    sv.id = 103, sv.score = 10.0;
    list_student.push_back(sv);
    sv.id = 104, sv.score = 10.0;
    list_student.push_back(sv);
    sv.id = 105, sv.score = 4.5;
    list_student.push_back(sv);
    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


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

    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


```
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


```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 != other.score)
            return other.score.compareTo(this.score);
        return this.id.compareTo(other.id);
    }
}
```

Sắp xếp cấu trúc sinh viên (main)


```java
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:

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>());
```

Hỏi đáp