1. **Nêu bài toán:**

Cần tìm phần tử nhỏ thứ k của mảng, cách thông thường là sắp xếp mảng theo thứ tự tăng dần sau đó lấy phần tử thứ k của mảng, ta sẽ có phần tử nhỏ thứ k. Như vậy, độ phức tạp của thuật toán sẽ phụ thuộc vào độ phức tạp của thuật toán sắp xếp. Thuật toán Select-k là thuật toán tìm phần tử nhỏ thứ k mà độ phức tạp được tối ưu đến O(n).

1. **Mô tả chi tiết thuật toán:**
   * Ý tưởng chính của thuật toán dựa trên thuật toán sắp xếp Quick Sort
   * Ta có mảng A gồm n phần tử, cần tìm phần tử nhỏ thứ k của mảng
   * Chia mảng thành 2 phần theo khóa p (\*)
   * Phần bên trái là những phần tử nhỏ hơn A[p], phần bên phải là những phần tử lớn hơn A[p].
   * Sau đo so sánh độ dài của mảng bên trái với k:
     + Nếu độ dài đó bằng k, ta có phần tử nhỏ thứ k của mảng là A[p]
     + Nếu độ dài đó nhỏ hơn k, ta sẽ tiếp tục áp dụng thuật toán cho nửa bên phải của mảng với vị trí cần tìm mới là độ dài mảng – độ dài mảng bên trái – 1.
     + Nếu độ dài đó lớn hơn k, ta sẽ áp dụng thuật toán cho nửa bên trái của mảng.

(\*) Có 2 cách chọn ra khóa p:

* Cách chọn khóa p ngẫu nhiên.
* Và cách chọn khóa p tối ưu, là phần tử trung vị của từng phần.

1. **Độ phức tạp:**

Xét mảng n phần tử, ta có ít nhất 3 \* (⌈m/2⌉-1-1) <= trung vị của các trung vị.

Khi đó ta có nhiều nhất n – 1 – 3 \* (⌈m/2⌉ - 2) = 7n/10 + 5 phần tử lớn hơn trung vị của trung vị.

Ta có độ phức tạp của thuật toán là:

T(n) ≤ c⋅n + T(⌈n/5⌉) + T(⌈7n/10 + 5⌉)

Ta cần chứng minh với mọi n ≥ 1 thì T(n) ≤ kn với k là một hằng số, khi đó độ phức tạp của thuật toán sẽ là T(n) = O(n).

* Trường hợp 1: 1 ≤ n ≤ 100

Ta có T(n) ≤ c, mà ta luôn chọn được k sao cho c ≤ kn

* T(n) ≤ kn
* Độ phức tạp của thuật toán là O(n).
* Trường hợp 2: n ≥ 100

T(n) ≤ T(⌈n/5⌉) + T( ⌈7n/10 +5⌉) + cn

≤ k⌈n/5⌉ + k ⌈7n/10 +5⌉ + cn

≤ k(n/5 + 1) + k(7n/10 + 5 + 1) + cn

≤ 9kn/10 + 7k + cn

≤ kn + (7k + cn - kn/10)

Chọn k = 50c, khi đó 7k + cn - kn/10 ≤ 0 => T(n) ≤ kn

* Độ phức tạp của thuật toán là O(n).

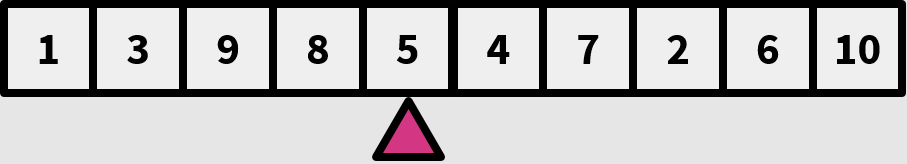
1. **Ví dụ**

Ví dụ 1: Tìm phần tử nhỏ thứ 2 của mảng, nhưng trong bài toán này, các phần tử được đếm từ 0 nên phần tử nhỏ thứ 2 sẽ là phần tử nhỏ thứ 1 trong lập trình.

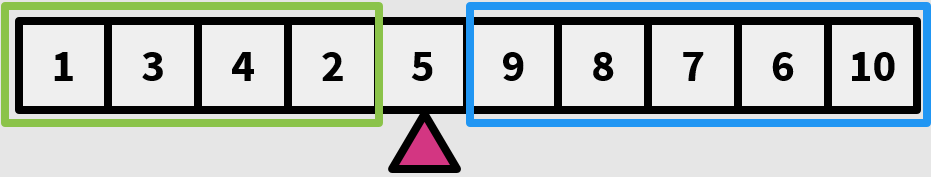
Mảng ban đầu:



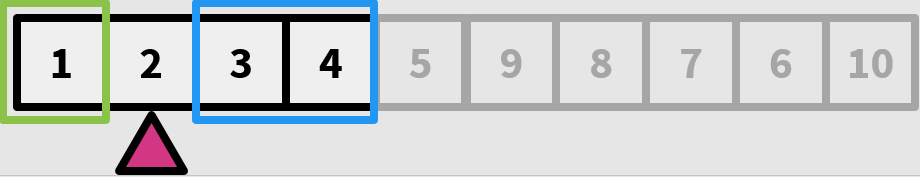
* Chọn phần tử trung vị của mảng



* Chia mảng thành 2 phần xung quanh phần tử được chọn làm khóa



* So sánh độ dài mảng bên trái với 1, ta thấy độ dài mảng bên trái lớn hơn 1 nên ta tiếp tục duyệt mảng bên trái, chọn phần tử trung vị là 2



* Lúc này độ dài chuỗi bên trái bằng 1 nên ta có phần tử nhỏ thứ 1 của mảng là phần tử 2.

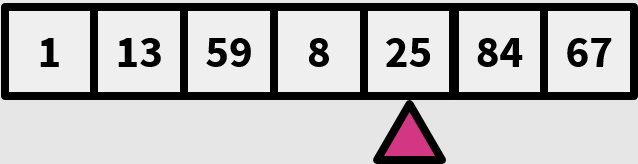
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Mảng ban đầu | Trung vị | Danh sách trái | Danh sách phải |
| Ban đầu | 1, 3, 9, 8, 5, 4, 7, 2, 6, 10 | 5 | 1, 3, 4, 2 | 9, 8, 7, 6, 10 |
| Vòng lặp 1 | 1, 3, 4, 2 | 2 | 1 | 3, 4 |

Ví dụ 2: Tìm phần tử nhỏ thứ 6 của mảng, tương đương với phần tử 5 trong lập trình.

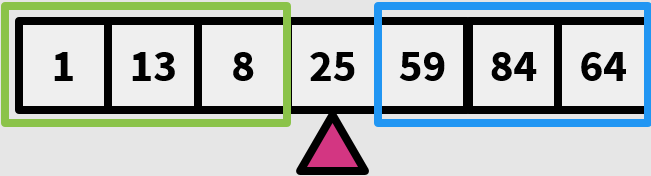
Mảng ban đầu:



Chọn phần tử trung vị là phần tử 25:



Chia mảng thành 2 phần xung quanh vị trí phần tử 25:



So sánh độ dài mảng bên trái với 5, ta thấy độ dài mảng bên trái nhỏ hơn 5, nên ta sẽ tiếp tục áp dụng thuật toán cho mảng bên phải với vị trí mới cần duyệt là 1.

Chọn phần tử trung vị là 64.

Chia mảng thành 2 phần xung quanh vị trí phần tử 64:



Lúc này, ta thấy độ dài chuỗi bên trái bằng 1, nên phần tử cần tìm sẽ là 64.

Phần tử nhỏ thứ 6 của mảng là: 64

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Mảng ban đầu | Trung vị | Danh sách trái | Danh sách phải |
| Ban đầu | 1, 13, 59, 8, 25, 84, 67 | 25 | 1, 13, 8 | 59, 84, 67 |
| Vòng lặp 1 | 59, 84, 67 | 64 | 59 | 84 |