**LÝ THUYẾT TUẦN 1: SORT AND BINARY SEARCH**

Câu hỏi kì này.

* Chia để trị.
* Nguyên lí QS
* Phần tử pivot ? Ảnh hưởng như thế nào tới độ phức tạp ?
* Tại sao ĐPT của QS lại là O(nlog(n)) và tệ nhất là O(n^2)
* BS hoạt động ? tại sao phải so sánh với phần tử chính giữa tập mà không phải phần tử lệch trái hay phải ?
* Độ phức tạp của BS
* Giải thích code 2 bài: Số bé thứ K và Help Sudo

**Mục Lục**

[I. TÌM HIỂU VỀ CHIA ĐỂ TRỊ (DIVIDE AND CONQUER). 2](#_Toc103622605)

[1. Định nghĩa: 2](#_Toc103622606)

[2. Nguyên lý: 2](#_Toc103622607)

[3. Ví dụ: 2](#_Toc103622608)

[II. SẮP XẾP NHANH (QUICK SORT). 3](#_Toc103622609)

[1. Ý tưởng: 3](#_Toc103622610)

[2. Chọn phần tử chốt (pivot): 3](#_Toc103622611)

[3. Nguyên lý: 3](#_Toc103622612)

[4. Code mẫu: 4](#_Toc103622613)

[5. Độ phức tạp thuật toán: 4](#_Toc103622614)

[III. TÌM HIỂU VỀ TÌM KIẾM NHỊ PHÂN (BINARY SEARCH). 5](#_Toc103622615)

[1. Định nghĩa: 5](#_Toc103622616)

[2. Ý tưởng: 5](#_Toc103622617)

[IV. GIẢI THÍCH BÀI TOÁN. 7](#_Toc103622618)

[Problem A: Số bé thứ K. 7](#_Toc103622619)

[Problem H: Help Sudo. 8](#_Toc103622620)

# I. TÌM HIỂU VỀ CHIA ĐỂ TRỊ (DIVIDE AND CONQUER).

## 1. Định nghĩa:

- Chia để trị là một phương pháp quan trọng trong việc thiết kế các giải thuật. Ý tưởng của phương pháp này khá đơn giản và rất dễ hiểu:

Khi cần giải quyết một bài toán, ta sẽ tiến hành chia bài toán đó thành các bài toán con nhỏ hơn. Tiếp tục chia cho đến khi các bài toán nhỏ này không thể chia thêm nữa, khi đó ta sẽ giải quyết các bài toán nhỏ nhất này và cuối cùng kết hợp giải pháp của tất cả các bài toán nhỏ để tìm ra giải pháp của bài toán ban đầu.

Devide

Conquer

Combine

## 2. Nguyên lý:

Chia để trị được chia làm 3 bước rõ ràng:

- Bước 1: Chia (Divide/Break).

+ Ta chia bài toán ban đầu thành các bài toán con nhỏ hơn. Mỗi bài toán con nên là một phần của bài toán ban đầu. Ở bước này, ta sử dụng phương pháp đệ qui để chia nhỏ các bài toán cho đến khi không thể chia thêm nữa. Khi đó, các bài toán con được gọi là "atomic – nguyên tử" hoặc là Sub-Problem, nhưng chúng vẫn biểu diễn một phần nào đó của bài toán ban đầu. Hay nói cách khác những bài toán nhỏ này giống với bài toán ban đầu.

- Bước 2: Trị (Conquer/Solve).

+ Ta tiến hành giải quyết các bài toán con đã được chia.

- Bước 3: Kết hợp bài toán (Combine/Merge).

+ Từ kết quả của những bài toán con, ta kết hợp chúng một cách đệ quy từ bài toán đơn giản đến phức tạp, cuối cùng tìm ra được đáp án cho bài toán ban đầu.

## 3. Ví dụ:

- Bài toán Tháp Hà Nội.

- Sắp xếp trộn.

- Sắp xếp nhanh.

- Vd thực tế: Chia bánh chưng.

# II. SẮP XẾP NHANH (QUICK SORT).

So với thuật toán sắp xếp nổi bọt (bubble sort) thì thuật toán sắp xếp nhanh có tốc độ nhanh hơn. Thay vì đi theo sắp xếp từng cặp như bubble sort, chúng ta có thể chia dữ liệu ra thành 22 danh sách, rồi so sánh từng phần tử của danh sách với một phần tử được chọn (gọi là phần tử chốt) và mục đích của chúng ta là đưa phần tử chốt về đúng vị trí của nó.

## 1. Ý tưởng:

- Dựa trên ý tưởng chia để trị. QS chia mảng thành hai anh sách bằng cách só sánh từng phần tử của danh sách với một phần tử được chọn gọi là phần tử chốt (pivot). Những phần tử nhỏ hơn hoặc bằng pivot được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử còn lại lớn hơn pivot thì đưa về sau và nằm trong danh sách con thứ hai. Tiếp tục chia như vậy với 2 danh sách con tạo được, đến khi nào ta thu được tất cả các danh sách con có độ dài bằng 1.

## 2. Chọn phần tử chốt (pivot):

Kỹ thuật chọn phần tử chốt ảnh hưởng khá nhiều đến khả năng rơi vào các vòng lặp vô hạn đối với các trường hợp đặc biệt. Tốt nhất là chọn phần tử chốt là trung vị của danh sách. Khi đó sau {\displaystyle log\_{2}(n)}log2(n) lần phân chia ta sẽ đạt tới kích thước danh sách bằng 1. Tuy nhiên điều đó rất khó. Có các cách chọn phần tử chốt như sau:

- Chọn phần tử trung vị của phần tử đầu, giữa và cuối làm pivot.

- Chọn phần tử đứng đầu hoặc cuối làm pivot.

- Chọn phần tử random trong mảng làm pivot.

## 3. Nguyên lý:



## 4. Code mẫu:

void QuickSort(int a[], int left, int right)

{

    int i = left, j = right;

    int pos = a[(left + right) / 2];

    while (i <= j)

    {

        while (a[i] < pos)

            i++;

        while (a[j] > pos)

            j--;

        if (i <= j)

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

    }

    if (left < j)

        QuickSort(a, left, j);

    if (i < right)

        QuickSort(a, i, right);

}

## 5. Độ phức tạp thuật toán:

- Trường hợp tốt nhất: Ta luôn chọn được pivot là phần tử trung vị của dãy số. Khi đó, 2 dãy con sẽ là 2 dãy bằng nhau, và ta cần log2(n) lần phân chia để đạt tới kích thước bằng 1. Mỗi lần phân chia, ta sẽ duyệt qua n phần tử của dãy. Cho nên, chung quy lại, độ phức tạp sẽ là O(nlog2(n)) hay O(nlog(n)). Đối với các trường hợp pivot là phần tử có giá trị nằm giữa dãy số, độ phức tạp sẽ là O(nloga(n)) hay vẫn là O(nlog(n)).

- Trường hợp xấu nhất: Ta luôn chọn được pivot là phần tử lớn nhất hoặc bé nhất của dãy đang xét. Do đó, khi phân chia làm 2 dãy, ta chia được 1 dãy có 1 phần tử và 1 dãy có n – 1 phần tử, cho nên sẽ cần tới n lần phân chia. Sau n lần phân chia, mỗi lần duyệt qua n phần tử, như thế độ phức tạp của thuật toán sẽ là O(n2).

# III. TÌM HIỂU VỀ TÌM KIẾM NHỊ PHÂN (BINARY SEARCH).

## 1. Định nghĩa:

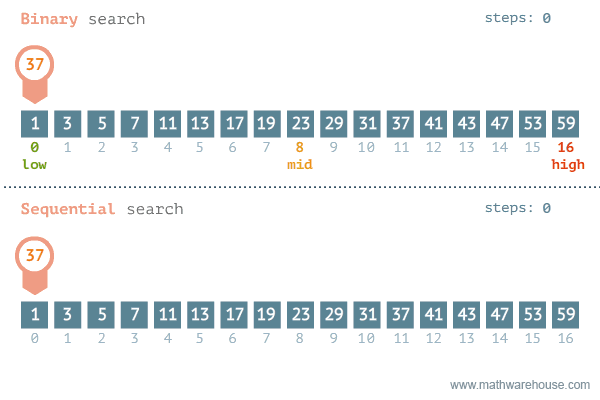
- Tìm kiếm: là một phần không thể thiếu của mọi ứng dụng, website hay phần mềm. Tính năng tìm kiếm cho phép người sử dụng nhanh chóng truy vấn và tìm kiếm các bản ghi theo mong muốn. Trong lập trình, với những bài toán tìm kiếm đơn giản, ta vẫn thường sử dụng cách tìm kiếm tuần tự hay tìm kiếm tuyến tính. Tuy nhiên, với những dãy số dài, phức tạp, cách tìm kiếm này lại rất hạn chế, bởi nó cần duyệt lần lượt tất cả các phần tử của dãy. Độ phức tạp là O(n).

- Tìm kiếm nhị phân: hay còn gọi là tìm kiếm nửa khoảng, tìm kiếm logarit. Ngay từ tên gọi, ta đã có thể đoán được độ phức tạp của thuật toán này là O(log(n)).

## 2. Ý tưởng:

- Thuật toán tìm kiếm sẽ so sánh các phần tử cần tìm với phần tử chính giữa của một dãy số đã được sắp xếp. Nếu phần tử chính giữa chính là phần tử cần tìm, thì lập tức trả về kết quả, nếu không, biên tìm kiếm sẽ được thu gọn lại.

- Cụ thể, với một dãy đã được sắp xếp, nếu phần tử chính giữa nhỏ hơn phần tử cần tìm, phần tử chính giữa được cập nhật làm biên dưới, ngược lại nếu phần tử chính giữa lớn hơn giá trị cần tìm, phần tử chính giữa sẽ được cập nhật làm biên trên. Chúng ta cứ tiếp tục như vậy đến khi nào tìm được giá trị cần tìm, hoặc 2 biên trên, dưới không còn thỏa mãn tính tuyến tính ban đầu nữa.

- Ta cần lấy phần tử chính giữa để so sánh bởi vì để tối ưu hóa nhất có thể thuật toán, tránh bị rơi vào các trường hợp xấu và trở thành tìm kiếm tuần tự.



3. Code mẫu:

// BINARY SEARCH

int BN(int a[], int value, int left, int right)

{

    while (left < right)

    {

        int mid = (right - left) / 2;

        if (a[mid] == value)

        {

            return mid;

        }

        else if (a[mid] < value)

        {

            left = mid + 1;

        }

        else

            right = mid - 1;

    }

    return -1;

}

# IV. GIẢI THÍCH BÀI TOÁN.

## Problem A: Số bé thứ K.

Bài này, ta đơn giản là dùng QuickSort để sắp xếp lại mảng đã cho, rồi sau đó gọi ra giá trị ở vị trí thứ k – 1 của mảng.

Code mẫu:

#include <bits/stdc++.h>

using namespace std;

#define endl '\n'

const int big = 1e5 + 10;

int a[big];

// QUICK SORT

void QS(int a[], int left, int right){

    int i = left, j = right;

    int pos = a[(left + right) / 2];

    while (i <= j){

        while (a[i] < pos) i++;

        while (a[j] > pos) j--;

        if (i <= j) swap(a[i], a[j]), i++, j--;

    }

    if (left < j) QS(a, left, j);

    if (i < right) QS(a, i, right);

}

int main(){

    int n, k;

    cin >> n >> k;

    for (int i = 0; i < n; i++) cin >> a[i];

    QS(a, 0, n - 1);

    cout << a[k - 1];

}

## Problem H: Help Sudo.

Bài này sử dụng BinarySearch để tìm kiếm cây có chiều cao bằng k. Trước đó, ta cần sắp xếp lại mảng đã cho bằng cách sử dụng QuickSort. Đối với khoảng biên nhỏ như này, ta vẫn có thể sử dụng tìm kiếm tuần tự được.

Code mẫu:

#include<bits/stdc++.h>

using namespace std;

#define endl '\n'

const int big = 1e6 + 10;

int a[big];

int BS(int a[], int l, int r, int k){

    while(l <= r){

        int pos = (l + r) / 2;

        if(a[pos] < k) l = pos + 1;

        else if(a[pos] > k) r = pos - 1;

        else if(a[pos] == k) return 1;

    }

    return -1;

}

void QS(int a[], int l, int r){

    int i = l, j = r;

    int pos = a[(l + r) / 2];

    while(i <= j){

        while(a[i] < pos) i++;

        while(a[j] > pos) j--;

        if(i <= j) swap(a[i], a[j]), i++, j--;

    }

    if(l < j) QS(a, l, j);

    if(i < r) QS(a, i, r);

}

int main(){

    ios::sync\_with\_stdio(false);cin.tie(0);cout.tie(0);

    int n, k;

    cin >> n >> k;

    for(int i = 0; i < n; i++){

        cin >> a[i];

    }

    QS(a, 0, n - 1);

    if(BS(a, 0, n - 1, k) == 1) cout << "YES";

    else cout << "NO";

}