

## Contents

ĐỀ CƯƠNG CHƯƠNG 02 .....	3
Thuật toán Interchange Sort .....	3
1. Bài toán dẫn nhập.....	3
2. Nghịch thế .....	3
3. Tư tưởng thuật toán.....	4
4. Áp dụng thuật toán.....	4
5. Hàm cài đặt .....	5
6. Danh sách liên kết đơn.....	6
7. Một cách cài đặt khác .....	6
Thuật toán Selection Sort .....	8
1. Bài toán dẫn nhập:.....	8
2. Tư tưởng thuật toán.....	8
3. Áp dụng thuật toán.....	8
4. Hàm cài đặt .....	10
5. Danh sách liên kết đơn.....	10
Thuật toán Bubble Sort.....	11
1. Bài toán dẫn nhập.....	11
2. Tư tưởng thuật toán.....	11
3. Áp dụng thuật toán.....	11
4. Hàm cài đặt .....	12
Thuật toán Insertion Sort .....	13
1. Bài toán dẫn nhập.....	13
2. Tư tưởng thuật toán.....	14
3. Thuật toán Insertion Sort .....	14
4. Hàm cài đặt .....	14
5. Ví dụ.....	14
Thuật toán QuickSort.....	16
1. Bài toán dẫn nhập.....	16
2. Tư tưởng thuật toán Quick Sort .....	16

- 3. Hàm cài đặt ..... 16
- 4. Một cách trình bày khác của thuật toán Quick Sort (Slide bài giảng cách này).. 17

## ĐỀ CƯƠNG CHƯƠNG 02

### Thuật toán Interchange Sort

#### 1. Bài toán dẫn nhập

- Bài toán: Viết hàm liệt kê tất cả các cặp giá trị trong mảng 1 chiều các số nguyên. Lưu ý: cặp (1,2) và cặp (2,1) là giống nhau.
- Ví dụ:

12	43	1	34	22
----	----	---	----	----

- Các cặp giá trị trong mảng là:
  - + (12, 43) , (12,1), (12,34), (12,22)
  - + (43,1), (43, 34), (43, 22)
  - + (1, 34), (1, 22)
  - + (34,22)
- Định nghĩa hàm:

```
void LietKe(int a[], int n)
{
    for (int i = 0; i <= n - 2; i++)
    {
        for (int j = i + 1; j <= n - 1; j++)
            printf("(%d,%d) ", a[i], a[j]);
    }
}
```

#### 2. Nghịch thế

- Khái niệm: 1 cặp giá trị (ai, aj) được gọi là nghịch thế khi ai, aj không thỏa điều kiện sắp thứ tự.
- Ví dụ 1: Cho mảng 1 chiều các số thực a có n phần tử: a0, a1, a2,..., an-2, an-1. Hãy sắp mảng theo thứ tự tăng dần, Khi đó cặp giá trị (ai,aj) (i<=j) được gọi là nghịch thế khi ai>=aj.
- Ví dụ 2: Cho mảng 1 chiều các số thực a có n phần tử: a0, a1, a2,..., an-2, an-1. Hãy sắp mảng theo thứ tự giảm dần, Khi đó cặp giá trị (ai,aj) (i<=j) được gọi là nghịch thế khi ai<=aj.
- Ví dụ 3: Hãy liệt kê các cặp giá trị nghịch thế trong mảng sau, biết rằng yêu cầu là sắp mảng tăng dần.

14	29	-1	10	5	23
----	----	----	----	---	----

- + Kết quả:
  - (14,-1), (14,10), (14,5)
  - (29, -1), ( 29, 10), ( 29, 5), (29, 23)
  - (10, 5)

### 3. Tư tưởng thuật toán

Thuật toán Interchange Sort sẽ duyệt qua tất cả các cặp giá trị trong mảng và hoán vị hai giá trị trong một cặp nếu cặp giá trị trong một cặp giá trị đó là nghịch thế.

### 4. Áp dụng thuật toán

Hãy sắp xếp mảng sau tăng dần:

24	45	23	13	43	-1
----	----	----	----	----	----

❖ **Thứ tự các bước khi sắp tăng dần mảng bằng thuật toán Interchange Sort.**

– Bước 1:

24	45	23	13	43	-1
----	----	----	----	----	----

24	45	23	13	43	-1
----	----	----	----	----	----

23	45	24	13	43	-1
----	----	----	----	----	----

13	45	24	23	43	-1
----	----	----	----	----	----

13	45	24	23	43	-1
----	----	----	----	----	----

-1	45	24	23	43	13
----	----	----	----	----	----

– Bước 2:

-1	45	24	23	43	13
----	----	----	----	----	----

-1	24	45	23	43	13
----	----	----	----	----	----

-1	23	45	24	43	13
----	----	----	----	----	----

-1	23	45	24	43	13
----	----	----	----	----	----

-1	13	45	24	43	23
----	----	----	----	----	----

– Bước 3

-1	13	45	24	43	23
----	----	----	----	----	----

-1	13	24	45	43	23
----	----	----	----	----	----

-1	13	24	45	43	23
----	----	----	----	----	----

-1	13	23	45	43	24
----	----	----	----	----	----

– Bước 4:

-1	13	23	45	43	24
----	----	----	----	----	----

-1	13	23	43	45	24
----	----	----	----	----	----

-1	13	23	24	45	43
----	----	----	----	----	----

– Bước 5:

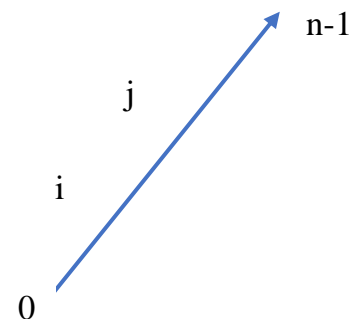
-1	13	23	24	45	43
-1	13	23	24	43	45

## 5. Hàm cài đặt

– Bài toán: Định nghĩa hàm sắp mảng một chiều các số nguyên tăng dần bằng thuật toán Interchange Sort.

– Hàm cài đặt:

```
void InterchangeSort(int a[], int n)
{
    for (int i = 0; i <= n - 2; i++)
    {
        for (int j=i+1; j<= n - 1; j++)
            if (a[i] > a[j])
            {
                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
    }
}
```



– Bài toán: Định nghĩa hàm sắp mảng một chiều các phân số tăng dần bằng thuật toán Interchange Sort

– Khai báo kiểu dữ liệu biểu diễn phân số

```
struct phanso
{
    int tu;
```

```

    int mau;
};
typedef struct phanso PHANSO;
int sosanh(PHANSO x, PHANSO y)
{
    float a = (float)x.tu / x.mau;
    float b = (float)y.tu / y.mau;
    if (a > b)
        return 1;
    if (a < b)
        return -1;
    return 0;
}
void InterchangeSortPS(PHANSO a[], int n)
{
    for (int i = 0; i <= n - 2; i++)
    {
        for (int j = i + 1; j <= n - 1; j++)
            if (sosanh(a[i], a[j]) == 1)
            {
                PHANSO temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
    }
}

```

## 6. Danh sách liên kết đơn

- Bài toán: Định nghĩa hàm sắp danh sách liên kết đơn các số nguyên tăng dần bằng thuật toán Interchange Sort.
- Cấu trúc dữ liệu

## 7. Một cách cài đặt khác

- Bài toán: Định nghĩa hàm sắp mảng một chiều các số nguyên tăng dần bằng thuật toán Interchange Sort.

```

void InterchangeSort(int a[], int n)
{
    for (int i = 0; i <= n - 2; i++)
    {
        for (int j = n - 1; j >= i + 1; j--)
            if (a[i] > a[j])
            {
                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
    }
}

```

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

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

## Thuật toán Selection Sort

### 1. Bài toán dẫn nhập:

- Bài toán: Viết hàm tìm vị trí giá trị lớn nhất trong mảng một chiều các số thực.
- Ví dụ:

0	1	2	3	4
12	43	1	34	22

- Kết quả: 1
- Hàm cài đặt:

```
int ViTriLonNhat(float a[], int n)
{
    int lc = 0;
    for (int i = 0; i < n; i++)
    {
        if (a[i] > a[lc])
            lc = i;
    }
    return lc;
}
```

### 2. Tư tưởng thuật toán

- Bài toán: Cho mảng 1 chiều a có n phần tử:  $a_0, a_1, a_2, \dots, a_{n-2}, a_{n-1}$ . Hãy sắp xếp các phần tử trong mảng tăng dần.
- Bước 0: Tìm vị trí giá trị nhỏ nhất trong phạm vi  $[0, n-1]$  và hoán vị giá trị tại vị trí này và phần tử  $a[0]$ .
- Bước 1: Tìm vị trí giá trị nhỏ nhất trong phạm vi  $[1, n-1]$  và hoán vị giá trị tại vị trí này và phần tử  $a[1]$ .
- ...
- Bước i: Tìm vị trí giá trị nhỏ nhất trong phạm vi  $[i, n-1]$  và hoán vị giá trị tại vị trí này và phần tử  $a[i]$ .
- Bước n-2: Tìm vị trí giá trị nhỏ nhất trong phạm vi  $[n-2, n-1]$  và hoán vị giá trị tại vị trí này và phần tử  $a[n-2]$ .

### 3. Áp dụng thuật toán

- Hãy sắp xếp mảng sau tăng dần:

24	45	23	13	43	-1
----	----	----	----	----	----

### ❖ Thứ tự các bước khi sắp tăng dần mảng trên bằng thuật toán Selection Sort.

- **Bước 0:**
  - + Tìm vị trí nhỏ nhất trong phạm vi  $[0, 5]$  của mảng



0	1	2	3	4	5
24	45	23	13	43	-1

+ Kết quả: Vị trí 5

+ Hoán vị giá trị tại vị trí nhỏ nhất và giá trị tại vị trí 0:

0	1	2	3	4	5
24	45	23	13	43	-1

+ Kết quả sau khi sắp xếp ở bước 0:

0	1	2	3	4	5
-1	45	23	13	43	24

#### – Bước 1:

+ Tìm vị trí nhỏ nhất trong phạm vi [1,5] của mảng

0	1	2	3	4	5
-1	45	23	13	43	24

+ Kết quả: vị trí 3

+ Hoán vị giá trị tại vị trí nhỏ nhất và giá trị tại vị trí 1:

0	1	2	3	4	5
-1	45	23	13	43	24

+ Kết quả sau khi sắp xếp ở bước 1:

0	1	2	3	4	5
-1	13	23	45	43	24

#### – Bước 2:

+ Tìm vị trí nhỏ nhất trong phạm vi [2,5] của mảng

0	1	2	3	4	5
-1	13	23	45	43	24

+ Kết quả: vị trí 2

+ Hoán vị giá trị tại vị trí nhỏ nhất và giá trị tại vị trí 2:

0	1	2	3	4	5
-1	13	23	45	43	24

+ Kết quả sau khi sắp xếp ở bước 2:

0	1	2	3	4	5
-1	13	23	45	43	24

#### – Bước 3:

+ Tìm vị trí nhỏ nhất trong phạm vi [3,5] của mảng

0	1	2	3	4	5
-1	13	23	45	43	24

+ Kết quả: vị trí 5

- + Hoán vị giá trị tại vị trí nhỏ nhất và giá trị tại vị trí 3:

0	1	2	3	4	5
-1	13	23	45	43	24

- + Kết quả sau khi sắp xếp ở bước 2:

0	1	2	3	4	5
-1	13	23	24	43	45

#### – Bước 4:

- + Tìm vị trí nhỏ nhất trong phạm vi [4,5] của mảng

0	1	2	3	4	5
-1	13	23	24	43	45

- + Kết quả: vị trí 5

- + Hoán vị giá trị tại vị trí nhỏ nhất và giá trị tại vị trí 3:

0	1	2	3	4	5
-1	13	23	24	43	45

- + Kết quả sau khi sắp xếp ở bước 2:

0	1	2	3	4	5
-1	13	23	24	43	45

## 4. Hàm cài đặt

- Bài toán: Định nghĩa hàm sắp mảng một chiều các số nguyên tăng dần bằng thuật toán Selection Sort.
- Hàm cài đặt

```
void SelectionSort_(int a[], int n)
{
    for (int i = 0; i <= n - 2; i++)
    {
        int lc = i;
        for (int j = i + 1; j <= n - 1; j++)
            if (a[j] < a[lc])
                lc = j;
        int temp = a[i];
        a[i] = a[lc];
        a[lc] = temp;
    }
}
```

## 5. Danh sách liên kết đơn

- Bài toán: Định nghĩa hàm sắp danh sách liên kết đơn các số nguyên tăng dần bằng thuật toán Selection Sort.

## Thuật toán Bubble Sort

### 1. Bài toán dẫn nhập

- Bài toán: Hãy liệt kê các cặp giá trị nằm kế tiếp nhau trong mảng một chiều các số nguyên.
- Ví dụ:

0	1	2	3	4
12	43	1	34	22

- Kết quả: (12,43), (43,1), (1,34), (34,22)
- Hàm cài đặt:

```
void LietKe(int a[], int n)
{
    for (int i = 0; i <= n-2; i++)
        printf("(%d , %d)", a[i], a[i+1]);
}
```

### 2. Tư tưởng thuật toán

- Bài toán: Cho mảng 1 chiều a có n phần tử: a<sub>0</sub>, a<sub>1</sub>, a<sub>2</sub>, a<sub>3</sub>, ..., a<sub>n-2</sub>, a<sub>n-1</sub>. Hãy sắp xếp các phần tử trong mảng tăng dần.
- Tư tưởng của thuật toán Bubble Sort là nhẹ nổi lên và nặng chìm xuống.
- Khái niệm nặng nhẹ: là khái niệm trừu tượng.

### 3. Áp dụng thuật toán

- Hãy sắp xếp mảng sau khi tăng dần

24	45	23	13	43	-1
----	----	----	----	----	----

❖ Thứ tự các bước khi sắp tăng dần mảng trên bằng thuật toán Bubble Sort.

- Bước 1: Nhẹ nổi lên, nặng chìm xuống lần thứ 1

24	24	24	24	<b>24</b>	<b>-1</b>
45	45	45	<b>45</b>	<b>-1</b>	24
23	23	<b>23</b>	<b>-1</b>	45	45
13	<b>13</b>	<b>-1</b>	23	23	23
<b>43</b>	<b>-1</b>	13	13	13	13
<b>-1</b>	43	43	43	43	43

- Bước 2: Nhẹ nổi lên, nặng chìm xuống lần thứ 2

-1	-1	-1	-1	<b>-1</b>
24	24	24	<b>24</b>	<b>13</b>
45	45	<b>45</b>	<b>13</b>	24
23	<b>23</b>	<b>13</b>	45	45
<b>13</b>	<b>13</b>	23	23	23

<b>43</b>	43	43	43	43
-----------	----	----	----	----

- Bước 3: Nhẹ nổi lên, nặng chìm xuống lần thứ 3

-1	-1	-1	<b>-1</b>
13	13	13	<b>13</b>
24	24	<b>24</b>	<b>23</b>
45	<b>45</b>	<b>23</b>	24
<b>23</b>	<b>23</b>	45	45
<b>43</b>	43	43	43

- Bước 4: Nhẹ nổi lên, nặng chìm xuống lần thứ 4

-1	-1	<b>-1</b>
13	13	<b>13</b>
23	23	<b>23</b>
24	<b>24</b>	<b>24</b>
<b>45</b>	<b>43</b>	43
<b>43</b>	45	45

- Bước 5: Nhẹ nổi lên, nặng chìm xuống lần thứ 5

-1	<b>-1</b>
13	<b>13</b>
23	<b>23</b>
24	<b>24</b>
<b>43</b>	<b>43</b>
<b>45</b>	45

#### 4. Hàm cài đặt

- Bài toán: Định nghĩa hàm sắp mảng một chiều các số nguyên tăng dần bằng thuật toán Bubble Sort.

```
//xét cặp từ cuối mảng
void BubbleSort(int a[], int n)
{
    for (int i = 0; i <= n - 2; i++)
        for (int j = n - 1; j >= i + 1; j--)
            if (a[j] < a[j - 1])
            {
                int temp = a[j];
                a[j] = a[j - 1];
                a[j - 1] = temp;
            }
}
```

## Thuật toán Insertion Sort


### 1. Bài toán dẫn nhập

- Bài toán 1: Định nghĩa hàm thêm giá trị x vào mảng một chiều các số thực có n phần tử đã được sắp tăng sao cho mảng vẫn được sắp tăng.
- Ví dụ: Thêm x=42

0	1	2	3	4	5	6
-32	17	29	47	53	68	97

0	1	2	3	4	5	6	7
-32	17	29	42	47	53	68	97



- Bài toán 2: Định nghĩa hàm thêm một giá trị x vào mảng một chiều các số thực có i phần tử đã được sắp tăng sao cho mảng vẫn được sắp tăng.

```
void ThemBaoToan(float a[], int &n, float x)
{
    int i = n - 1;
    while (i >= 0 && a[i]>x)
    {
        a[i + 1] = a[i];
        i--;
    }
    a[i + 1] = x;
    n++;
}
```

- Một cách cài đặt khác:

```
void ThemBaoToan1(float a[],int &n, float x)
{
    int i;
    for (i = n - 1; i >= 0 && a[i] > x; i--)
        a[i + 1] = a[i];
    a[i + 1] = x;
    n++;
}
```

- Bài toán 2: Định nghĩa hàm thêm một giá trị x vào mảng một chiều các số thực có i phần tử đã được sắp tăng sao cho mảng vẫn được sắp tăng.

```
void ThemBaoToan2(float a[], int &i, float x)
{
    int j;
```

```

        for (j = i - 1; j >= 0 && a[j] > x; j--)
            a[j + 1] = a[j];
        a[j + 1] = x;
        i++;
    }

```

## 2. Tư tưởng thuật toán

- Thuật toán Insertion sort sắp xếp dựa trên tư tưởng là không gian cần sắp xếp đã được sắp xếp một phần và ta chỉ cần thêm một giá trị mới vào không gian này sao cho không gian mới được sắp xếp mà thôi.

## 3. Thuật toán Insertion Sort

- Bước 1: Chèn  $a[1]$  vào mảng  $a$  có 1 phần tử đã được sắp tăng để được mảng  $a$  có 2 phần tử sắp tăng.
- Bước 2: Chèn  $a[2]$  vào mảng  $a$  có 2 phần tử đã được sắp tăng để được mảng  $a$  có 3 phần tử sắp tăng.
- ...
- Bước  $i$ : Chèn  $a[i]$  vào mảng  $a$  có  $i$  phần tử đã được sắp tăng để được mảng  $a$  có  $(i+1)$  phần tử sắp tăng.
- ...
- Bước  $n-1$ : Chèn  $a[n-1]$  vào mảng  $a$  có  $(n-1)$  phần tử đã được sắp tăng để được mảng  $a$  có  $n$  phần tử sắp tăng.

## 4. Hàm cài đặt

- Bài toán: Định nghĩa hàm sắp mảng 1 chiều các số nguyên tăng dần bằng thuật toán insertion sort.
- Hàm cài đặt:

```

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

```

## 5. Ví dụ

Giả sử một mảng được cho ngẫu nhiên 12, 11, 13, 5, 6 thì việc áp dụng thuật toán Insertion Sort được mô tả như sau:

- Ta sẽ bắt đầu từ phần tử thứ 2 của mảng, nên vòng lặp sẽ lặp qua 11, 13, 5, 6.

<b>12</b>	11	13	5	6
-----------	----	----	---	---

- $i = 1$  (Phần tử thứ 2 của mảng). Vì 11 nhỏ hơn 12 nên di chuyển 12 lên vị trí thứ  $i$  và chèn 11 vào trước 12 (vị trí thứ  $i - 1$ )

<b>11</b>	<b>12</b>	13	5	6
-----------	-----------	----	---	---

- Với  $i = 2$ : 13 sẽ vẫn ở vị trí của nó vì tất cả các phần tử trong  $A[0 \dots i - 1]$  đều nhỏ hơn 13

<b>11</b>	<b>12</b>	<b>13</b>	5	6
-----------	-----------	-----------	---	---

- Với  $i = 3$ : 5 sẽ di chuyển về đầu và tất cả các phần tử khác từ 11 đến 13 sẽ di chuyển trước một vị trí so với vị trí hiện tại của chúng.

<b>5</b>	<b>11</b>	<b>12</b>	<b>13</b>	6
----------	-----------	-----------	-----------	---

- Với  $i = 4$ : 6 sẽ chuyển đến vị trí sau 5, và các phần tử từ 11 đến 13 sẽ di chuyển trước một vị trí so với vị trí hiện tại của chúng.

<b>5</b>	<b>6</b>	<b>11</b>	<b>12</b>	<b>13</b>
----------	----------	-----------	-----------	-----------

➔ Vậy mảng sau khi được sắp xếp là 5, 6, 11, 12, 13

# Thuật toán QuickSort

## 1. Bài toán dẫn nhập

- Bài toán: Cho mảng một chiều các số nguyên và giá trị x. Hãy tách mảng a ban đầu thành 2 mảng b và c sao cho mảng b chỉ chứa các giá trị nhỏ hơn x, mảng c chứa các giá trị lớn hơn x.

```
void Split(int a[], int n, int x,
           int b[], int &k, int c[], int &l)
{
    k = l = 0;
    for (int i = 0; i < n; i++)
    {
        if (a[i] < x)
            b[k++] = a[i];
        else if (a[i] > x)
        {
            c[l++] = a[i];
        }
    }
}
```

## 2. Tư tưởng thuật toán Quick Sort

- Thuật toán QuickSort chia không gian cần sắp xếp thành 2 không gian con là không gian con 1 và không gian con 2. Không gian con 1 là không gian mà tất cả các phần tử thuộc không gian này đều nhỏ hơn tất cả các phần tử thuộc không gian con 2.
  - + Nếu không gian con thứ nhất có nhiều hơn một phần tử thì sắp xếp không gian con này bằng thuật toán quick sort.
  - + Nếu không gian con thứ hai có nhiều hơn 1 phần tử thì sắp xếp không gian con này bằng thuật toán quick sort.

## 3. Hàm cài đặt

```
void QuickSort(int a[], int n)
{
    if (n <= 1)
        return;
    int b[100]; int k;
    int c[100]; int l;
    int trongtai = a[0];
    int i;
    Split(a, n, trongtai, b, k, c, l);
    QuickSort(b, k);
    QuickSort(c, l);
    for (i = 0; i < k; i++)
        a[i] = b[i];
    for (i = 0; i < n-k-l; i++)
        a[k+i] = trongtai;
```



```

    for (i = 0; i < l; i++)
        a[k+(n-k-l)+i] = c[i];
}

void Split(int a[], int n, int x,
int b[], int &k, int c[], int &l)
{
    k = l = 0;
    for (int i = 0; i < n; i++)
    {
        if (a[i] < x)
            b[k++] = a[i];
        else if (a[i] > x)
        {
            c[l++] = a[i];
        }
    }
}

```

#### 4. Một cách trình bày khác của thuật toán Quick Sort (Slide bài giảng cách này)

##### ❖ Ý tưởng:

- Quick sort là một trong những thuật toán chia để trị, Quick sort chia một mảng lớn thành 2 mảng con nhỏ hơn:
  - Mảng có phần tử nhỏ.
  - Mảng có phần tử lớn.
- Sau đó Quick sort có thể sort các mảng con bằng phương pháp đệ quy, ý tưởng của Quick sort là:
  1. Chọn một phần tử để so sánh, gọi là phần tử Key (Pivot), từ trong mảng đầu tiên.
  2. Phân vùng và sort mảng con trong phân vùng làm sao cho các phần tử lớn hơn từ phần tử Key nằm sau (bên phải) và các phần tử bé hơn phần tử Key nằm trước (bên trái). Đây được gọi là quá trình phân vùng.
  3. Cuối cùng là đệ quy sử dụng các bước trên cho các mảng với phần tử bé hơn và phân tách với các phần tử lớn hơn sau khi phân vùng.

##### ❖ Các bước thực hiện (Quy trình thuật toán)

- Bước 1: Lấy phần tử chốt là phần tử ở cuối danh sách.
- Bước 2: Chia mảng theo phần tử chốt.
- Bước 3: Sử dụng sắp xếp nhanh một cách đệ quy với mảng con bên trái.
- Bước 4: Sử dụng sắp xếp nhanh một cách đệ quy với mảng con bên phải.

### ❖ Thuật toán phân đoạn

- Đặt pivot là phần tử cuối cùng của dãy số arr.
- Chúng ta bắt đầu từ phần tử trái nhất của dãy số có chỉ số là left, và phần tử phải nhất của dãy số có chỉ số là right - 1 (bỏ qua phần tử pivot).
- Chừng nào  $left < right$  mà  $arr[left] > pivot$  và  $arr[right] < pivot$  thì đổi chỗ hai phần tử left và right.
- Sau cùng, ta đổi chỗ hai phần tử left và pivot cho nhau.
- Khi đó, phần tử left đã đứng đúng vị trí và chia dãy số làm đôi (bên trái và bên phải)

### ❖ Thuật toán phân đoạn – Cài đặt

```
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];    // pivot
    int left = low;
    int right = high - 1;
    while (true){
        while (left <= right && arr[left] < pivot) left++;
        while (right >= left && arr[right] > pivot) right--;
        if (left >= right) break;
        swap(arr[left], arr[right]);
        left++;
        right--;
    }
    swap(arr[left], arr[high]);
    return left;
}
```

- Cách khác:

```
int partition(int a[], int low, int high)
{
    int pivot = a[high];
    int vt = (low - 1);
    for (int i = low; i <= high - 1; i++)
    {
        if (a[i] < pivot)
        {
            vt++;
            swap(a[vt], a[i]);
        }
    }
    vt = vt + 1;
    swap(a[vt], a[high]);
}
```

```

        return vt;
    }

```

#### ❖ Hàm cài đặt:

```

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi là chỉ số nơi phần tử này đã đứng đúng vị trí
        và là phần tử chia mảng làm 2 mảng con trái & phải */
        int pi = partition(arr, low, high);

        // Gọi đệ quy sắp xếp 2 mảng con trái và phải
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

```

#### ❖ Ví dụ

Lưu ý: Thuật toán Quick Sort chỉ cần minh họa 1 bước của thuật toán phân hoạch.

arr[]	10	80	30	90	40	50	70
Indexes	0	1	2	3	4	5	6

Ví dụ chọn phần tử cuối là phần tử chốt

- pivot = 6, left = 0, right = 5
- arr[left] = 10 < arr[pivot] = 70 và left <= right, left = 1
- arr[left] = 80 > arr[pivot] = 70, tạm dừng
- arr[right] = 50 < arr[pivot] = 70, tạm dừng
- Do left < right, đổi chỗ arr[left], arr[right]

arr[]	10	80	30	90	40	50	70
Indexes	0	1	2	3	4	5	6

<b>arr[]</b>	<b>10</b>	<b>50</b>	<b>30</b>	<b>90</b>	<b>40</b>	<b>80</b>	<b>70</b>
Indexes	0	<b>1</b>	2	3	4	<b>5</b>	6

Lần 2

<b>arr[]</b>	<b>10</b>	<b>50</b>	<b>30</b>	<b>90</b>	<b>40</b>	<b>80</b>	<b>70</b>
Indexes	0	1	2	3	4	5	6

- $arr[] = \{10, 50, 30, 90, 40, 80, 70\}$
- $left = 2, right = 4$
- $arr[left] = 30 < arr[pivot] = 70$  và  $left \leq right$ ,  $left = 3$
- $arr[left] = 90 > arr[pivot] = 70$ , tạm dừng
- $arr[right] = 40 < arr[pivot] = 70$ , tạm dừng
- Do  $left < right$ , đổi chỗ  $arr[left], arr[right]$

<b>arr[]</b>	<b>10</b>	<b>50</b>	<b>30</b>	<b>90</b>	<b>40</b>	<b>80</b>	<b>70</b>
Indexes	0	1	2	<b>3</b>	<b>4</b>	5	6

<b>arr[]</b>	<b>10</b>	<b>50</b>	<b>30</b>	<b>40</b>	<b>90</b>	<b>80</b>	<b>70</b>
Indexes	0	1	2	<b>3</b>	<b>4</b>	5	6

$left=4, right=3$

Do  $\text{left} \geq \text{right}$ , đổi chỗ  $\text{arr}[\text{left}]$ ,  $\text{arr}[\text{pivot}]$

$\text{left}=4$ ,  $\text{right}=3$

$\text{arr}[\text{left}]$

$\text{arr}[\text{pivot}]$

arr[]	10	50	30	40	90	80	70
Indexes	0	1	2	3	4	5	6

arr[]	10	50	30	40	70	80	90
Indexes	0	1	2	3	4	5	6

**Kết quả sau khi phân đoạn:**

arr[]	10	50	30	40	70	80	90
Indexes	0	1	2	3	4	5	6

