

## Nội dung cần ôn tập

1. Tìm kiếm.
  2. Sắp xếp.
  3. Cây nhị phân / cây nhị phân tìm kiếm
  4. Bảng băm

## 1. TÌM KIẾM

#### **Tuyển Tính, Tuyển tính cải tiến, Nhị Phân, Nội Suy**

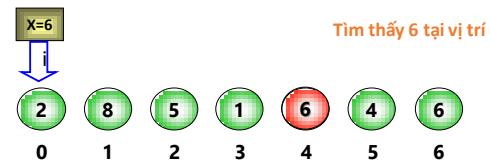
- Viết code và cho biết độ phức tạp
  - Chạy tay

## Tìm kiếm tuyển tính

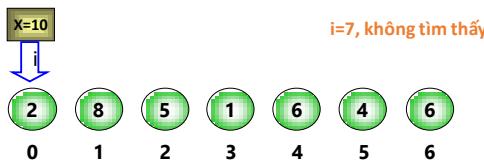
## ✓ Ý tưởng:

Thuật toán tiến hành so sánh x lần lượt với phần tử thứ nhất, thứ hai, ... của mảng a cho đến khi gặp được phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy x

## Tìm kiếm tuyển tính



### Tìm kiếm tuyến tính



### Tìm kiếm tuyến tính

**Cài đặt:** (trên mảng)

```
int linearSearch(int A[], int n, int x) {
    int i = 0;
    while (i < n)
    {
        if (A[i] == x)
            return i;
        i++;
    }
    return -1;
}
```

### Tìm kiếm tuyến tính

#### Đánh giá:

- Trường hợp tốt nhất (best case):  $a_0$  chứa khóa  $x \rightarrow$  số lần lặp là 1  $\rightarrow$  độ phức tạp hằng số  $O(1)$
- Trường hợp xấu nhất (worst case):  $A$  không có phần tử có khóa  $x \rightarrow$  số lần lặp là  $n \rightarrow$  độ phức tạp tuyến tính  $O(n)$ .
- Trường hợp trung bình (average case): độ phức tạp tuyến tính  $O(n)$ .

### Tìm kiếm tuyến tính CÀI TIẾN

Điều kiện dừng là gì ?

```
int linearSearch(int A[], int n, int x) {
    int i = 0;
    while (i < n)
    {
        if (A[i] == x)
            return i;
        i++;
    }
    return -1;
}
```

10

### Tìm kiếm tuyến tính CÀI TIẾN

**Phân tích:** Theo thuật toán tìm tuyến tính:

- Cần phải kiểm tra điều kiện dừng khi xét hết danh sách ( $i < n$ )
- Cần phải kiểm tra điều kiện dừng khi tìm thấy phần tử  $a_i$  trong vòng lặp

→ Rút gọn điều kiện dừng

### Tìm kiếm tuyến tính CÀI TIẾN

#### Ý tưởng:

- Thêm phần tử  $a_n$  có khóa  $x$  vào  $A$ , khi này  $A$  có  $n+1$  phần tử. Phần tử thêm vào được gọi là phần tử cầm canh.
- Chỉ cần điều kiện dừng là tìm thấy phần tử  $a_i$  có khóa  $x$

## Tìm kiếm tuyến tính CÀI TIẾN

Cài đặt: (trên mảng)

```
int linearSearchA(int A[], int n, int x) {
    int i = 0; A[n] = x; //A có hơn n phần tử
    while (A[i] != x)
        i++;
    if (i < n)
        return i;
    else
        return -1;
}
```

## Tìm kiếm nhị phân

Thuật toán:

**Đầu vào:** Danh sách **A** có **n** phần tử đã có thứ tự **R**, giá trị khóa **x** cần tìm.

**Đầu ra:** Chỉ số **i** của phần tử **a<sub>i</sub>** trong **A** có giá trị khóa là **x**. Trong trường hợp không tìm thấy **i=-1**

## Tìm kiếm nhị phân

Ý tưởng:

- Chọn **a<sub>m</sub>** ở giữa **A** để tận dụng kết quả so sánh với khóa **x**. **A** được chia thành hai phần: trước và sau **a<sub>m</sub>**. Chỉ số bắt đầu, kết thúc của **A** là **I, r**
- Nếu **x = a<sub>m</sub>**, tìm thấy và dừng.
- Xét thứ tự **x, a<sub>m</sub>**. Nếu thứ tự này
  - Là **R**, thì tìm **x** trong đoạn **[l, r]** với **r=m-1**;
  - Ngược lại, tìm **x** trong đoạn **[l, r]** với **l=m+1**.

## Tìm kiếm nhị phân – Thuật giải

**Input:** mảng a: a[0], a[1], ..., a[n-1] đã có thứ tự và giá trị x

**Output:** vị trí của khóa x tìm được, trả về -1 nếu không tìm thấy

**Các bước thực hiện:**

- ✓ **Bước 1:** left = 0; right = n - 1;
- ✓ **Bước 2:**
- mid = (left+right)/2; //lấy mốc so sánh
- So sánh a[mid] với x, có 3 khả năng :

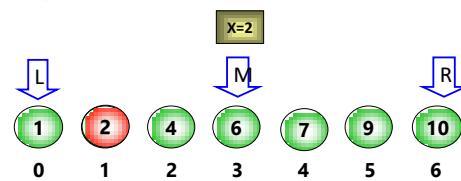
  - a[mid] = x: Tìm thấy. Dừng
  - a[mid] > x: //tim tiếp x trong dãy con a<sub>left</sub> .. a<sub>mid-1</sub>
    - right = mid - 1;
  - a[mid] < x: //tim tiếp x trong dãy con a<sub>mid+1</sub> .. a<sub>right</sub>
    - left = mid + 1;

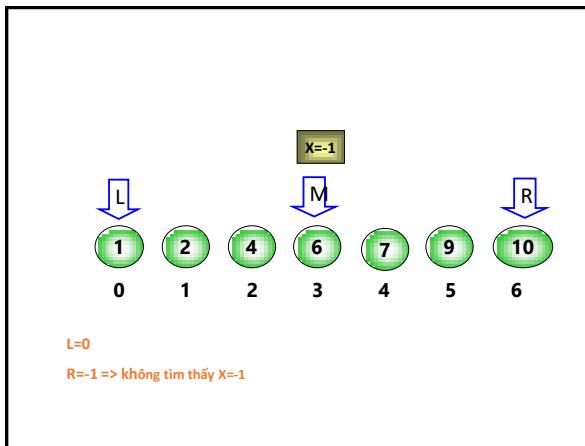
16

## Tìm kiếm nhị phân – Thuật giải

- ✓ **Bước 3:**
- Nếu left ≤ right //còn phần tử chưa xét, tìm tiếp.  
Lặp lại Bước 2.
  - Ngược lại: Dừng; //Đã xét hết tất cả các phần tử.

Tìm thấy 2 tại vị trí 1



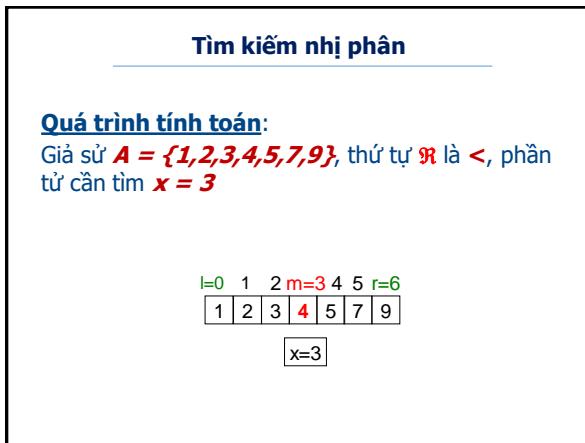


### Tìm kiếm nhị phân

#### Thuật toán:

```

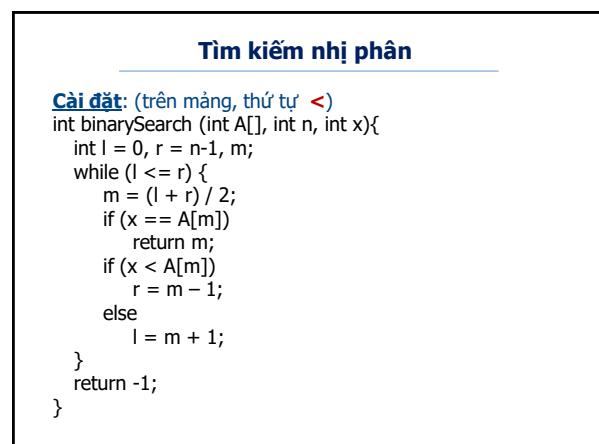
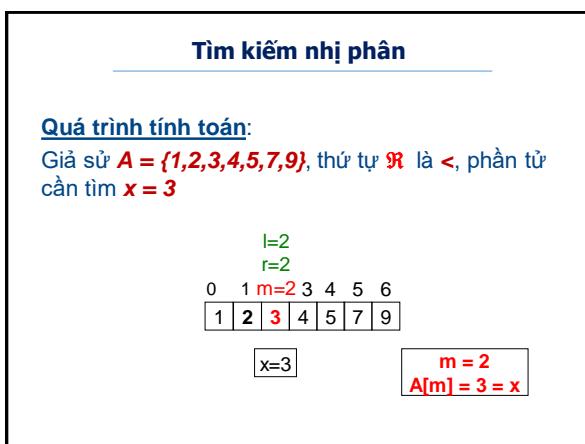
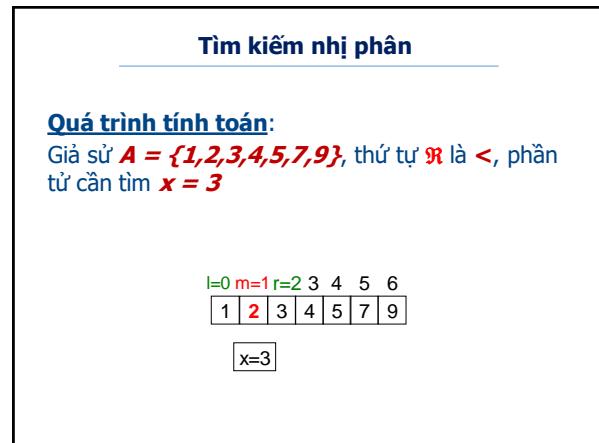
 $I \leftarrow 0, r \leftarrow n-1$ 
while  $I \leq r$ 
   $m \leftarrow (I + r) \text{ div } 2$ 
  if  $x = A[m]$  then return  $m$  end if
  if  $x < A[m]$  then  $r \leftarrow m - 1$ 
  else  $I \leftarrow m + 1$  end if
end while
return -1
  
```



### Tìm kiếm nhị phân

#### Quá trình tính toán:

Giả sử  $A = \{1, 2, 3, 4, 5, 7, 9\}$ , thứ tự  $\ll$  là  $<$ , phần tử cần tìm  $x = 3$



### Tìm kiếm nội suy

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

$$\text{mid} = (l+r)/2$$

$$\text{mid} = l + (r-l)/2$$

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

$$m = l + \frac{(r-l) \times (x - A[l])}{A[r] - A[l]}$$

$$1/2 \rightarrow \frac{x - A[l]}{A[r] - A[l]}$$

Nội Suy  
(Tí lệ % x nằm ở khoảng nào  
từ a[right] đến a[left])

### Tìm kiếm nội suy

**Từ khóa:** Interpolation Search

**Điều kiện:** Danh sách  $A = \{a_0, a_1, \dots, a_{n-1}\}$  đã có thứ tự  $\mathfrak{R}$  và giá trị khóa được rải đều trên danh sách.

**Phân tích:** Giá trị khóa rải đều trên danh sách → vị trí  $a_m$  chia danh sách tìm kiếm tương ứng với tỉ lệ giá trị  $x$  trong miền giá trị khóa của danh sách tìm kiếm.

### Tìm kiếm nội suy

#### ❖ TÌM KIẾM NỘI SUY

##### Ý tưởng:

- Thay vì xác định điểm  $m = (l + r) / 2$  như trong tìm kiếm nhị phân, xác định nội suy  $m$  như sau:

$$m = l + \frac{(r-l) \times (x - A[l])}{A[r] - A[l]}$$

- Các bước còn lại tương tự tìm kiếm nhị phân

### Tìm kiếm nội suy

#### Thuật toán:

**Đầu vào:** Danh sách  $A$  có  $n$  phần tử đã có thứ tự  $\mathfrak{R}$ , giá trị khóa  $x$  cần tìm.

**Đầu ra:** Chỉ số  $i$  của phần tử  $a_i$  trong  $A$  có giá trị khóa là  $x$ . Trong trường hợp không tìm thấy  $i=-1$

### Tìm kiếm nội suy

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

$$\text{mid} = (l+r)/2$$

$$\text{mid} = l + (r-l)/2$$

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

$$m = l + \frac{(r-l) \times (x - A[l])}{A[r] - A[l]}$$

$$1/2 \rightarrow \frac{x - A[l]}{A[r] - A[l]}$$

Nội Suy  
(Tí lệ % x nằm ở khoảng nào  
từ a[right] đến a[left])

### Tìm kiếm nội suy

**Từ khóa:** Interpolation Search

**Điều kiện:** Danh sách  $A = \{a_0, a_1, \dots, a_{n-1}\}$  đã có thứ tự  $\mathfrak{R}$  và giá trị khóa được rải đều trên danh sách.

**Phân tích:** Giá trị khóa rải đều trên danh sách → vị trí  $a_m$  chia danh sách tìm kiếm tương ứng với tỉ lệ giá trị  $x$  trong miền giá trị khóa của danh sách tìm kiếm.

## Tìm kiếm nội suy

### ❖ TÌM KIẾM NỘI SUY

#### Ý tưởng:

- Thay vì xác định điểm  $m = (l + r) / 2$  như trong tìm kiếm nhị phân, xác định nội suy  $m$  như sau:  

$$m = l + \frac{(r - l) \times (x - A[l])}{A[r] - A[l]}$$
- Các bước còn lại tương tự tìm kiếm nhị phân

## Tìm kiếm nội suy

#### Thuật toán:

**Đầu vào:** Danh sách  $A$  có  $n$  phần tử đã có thứ tự  $\mathfrak{R}$ , giá trị khóa  $x$  cần tìm.

**Đầu ra:** Chỉ số  $i$  của phần tử  $a_i$  trong  $A$  có giá trị khóa là  $x$ . Trong trường hợp không tìm thấy  $i=-1$

## Tìm kiếm nội suy

#### Thuật toán:

```
l ← 0, r ← n-1
while l ≤ r
    m ← l+((r-l)*(x-A[l]) / (A[r]-A[l]))
    if x = A[m] then return m end if
    if x < A[m] then r ← m - 1
    else l ← m + 1 end if
end while
return -1
```

## Tìm kiếm nội suy

#### Thuật toán:

```
l ← 0, r ← n-1
while l ≤ r
    m ← l+((r-l)*(x-A[l]) / (A[r]-A[l]))
    if x = A[m] then return m end if
    if x < A[m] then r ← m - 1
    else l ← m + 1 end if
end while
return -1
```

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

#### Câu 1 (4 điểm):

a. Anh/ chị hãy viết hàm cài đặt thuật toán tìm kiếm giá trị  $x$  bằng phương pháp **tìm nhị phân** trên mảng số nguyên có thứ tự tăng dần bằng ngôn ngữ C/C++ (2 điểm).

b. Trình bày các bước (vẽ từng bước) theo hàm đã cài đặt ở câu a để thực hiện việc tìm kiếm giá trị  $x=10$  trên mảng số nguyên có giá trị như sau: 6 11 27 34 41 53 (2 điểm).

#### Câu 2 (6 điểm):

a. Anh/ chị hãy trình bày ý tưởng thuật toán nổi bọt (Bubble Sort) để sắp xếp một mảng một chiều các số nguyên theo thứ tự tăng dần (2 điểm).

b. Viết hàm cài đặt thuật toán trong câu 2a bằng ngôn ngữ C/C++ (2 điểm).

c. Trình bày các bước (vẽ từng bước) thực hiện việc sắp xếp theo hàm đã cài đặt ở câu 2b với mảng các số nguyên có giá trị như sau: 14 23 35 11 41 8 (2 điểm).

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

#### Câu 2 (5 điểm)

Người ta muốn sử dụng danh sách liên kết đơn để quản lý danh sách các **phân số**, biết rằng mỗi phân số  $X$  đều có hai thành phần tử số (TS) và mẫu số (MS). Anh/Chị hãy thực hiện các yêu cầu sau đây:

a. Định nghĩa các cấu trúc cần thiết để lưu danh sách theo mô tả trên (2 điểm).

b. Viết các hàm cần thiết để nhập một danh sách gồm  $N$  phân số từ bàn phím, biết rằng khi nhập lần lượt từng phân số sẽ thêm vào đầu danh sách (2 điểm).

c. Viết hàm tính tổng các phân số có trong danh sách (1 điểm).

#### Câu 3 (2 điểm)

Dựa vào các kiến thức đã học Anh/Chị hãy so sánh hai loại cấu trúc dữ liệu sau đây: mảng và danh sách liên kết.

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 (2 điểm):

- Anh/Chị hãy trình bày ý tưởng của thuật toán tìm kiếm nhị phân (Binary Search) để tìm kiếm một số trên một mảng đã có thứ tự.
- Trình bày các bước (về từng bước) theo thuật toán tìm kiếm nhị phân thực hiện tìm kiếm giá trị  $x = 8$  trên mảng số nguyên có giá trị: 1 4 6 13 17 19.

Câu 2 (2.5 điểm):

- Anh/Chị hãy viết hàm cài đặt thuật toán **chọn trực tiếp** (Selection Sort) để sắp xếp một mảng các số nguyên có N phần tử theo chiều giảm dần bằng ngôn ngữ C/C++, cho biết độ phức tạp của giải thuật.
- Trình bày các bước (về từng bước) áp dụng thuật toán ở câu 2.a để sắp xếp mảng số nguyên {2, 7, 10, 9, 5, 3, 8, 40} giảm dần.

37

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 (3 điểm):

- Trình bày giải thuật sắp xếp tăng dần theo phương pháp **Chèn trực tiếp** (Insertion Sort).
- Sử dụng phương pháp **Chèn trực tiếp** (Insertion Sort) để sắp xếp tăng dần dãy số nguyên sau:

14	9	7	2	8	10
----	---	---	---	---	----

Câu 1 : (3.5 điểm)

Anh/chị hãy thực hiện :

- Trình bày giải thuật **Chọn trực tiếp** để sắp xếp một mảng các số nguyên giảm dần. (1.5 điểm)
- Trình bày các bước (về từng bước) áp dụng giải thuật trong câu 1.a để sắp xếp mảng số nguyên {10, 5, 30, 70, 40, 80, 90} giảm dần.

38

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1: (2.5 điểm)

- Trình bày các bước giải thuật sắp xếp Quick sort (không viết chương trình) để sắp xếp mảng số nguyên N phần tử **giảm dần**, cho biết độ phức tạp giải thuật.
- Trình bày các bước (về từng bước) áp dụng giải thuật trong câu 1.a để sắp xếp mảng số nguyên {10, 5, 30, 70, 40, 80, 90} giảm dần.

Câu 1 : (3.5 điểm)

Anh/chị hãy thực hiện :

- Trình bày giải thuật **Chọn trực tiếp** để sắp xếp một mảng các số nguyên **giảm dần**. (1.5 điểm)
- Trình bày các bước (về từng bước) thực hiện theo giải thuật câu a đối với mảng X gồm các số nguyên theo thứ tự từ trái qua phải, như sau: 91, 12, 30, 1, 83, 20 (2 điểm)

Câu 1 (3 điểm) : Cho dãy số ban đầu như sau : 17 72 99 32 58 70 44 12 23

Hãy thực hiện các yêu cầu sau:

- Hãy trình bày các bước thực hiện **thuật toán chọn trực tiếp** (1.5 d)
- Vẽ hình từng bước thực hiện của thuật toán trên để sắp xếp dãy số theo thứ tự **giảm dần** (không cần lập trình) (1.5 d)

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 (6 điểm) :

- Anh/chị hãy trình bày giải thuật toán **chọn trực tiếp** (Selection Sort) để sắp xếp một mảng các số nguyên tăng dần. (2 điểm)
- Viết hàm cài đặt thuật toán trên bằng ngôn ngữ C/C++. (2 điểm)
- Trình bày các bước (về từng bước) thực hiện sắp xếp theo hàm đã cài đặt ở câu b đối với mảng các số nguyên có giá trị như sau: 19 11 31 15 37 17 (2 điểm)

Câu 2 (4 điểm):

- Anh/chị hãy trình bày giải thuật toán **Tìm kiếm nhị phân** trên mảng số nguyên có thứ tự giảm dần bằng ngôn ngữ C/C++. (2 điểm)
- Trình bày các bước (về từng bước) theo hàm đã cài đặt ở câu a thực hiện tìm giá trị  $X=50$  trong mảng các số nguyên có giá trị như sau: 90 80 60 26 24 12 (2 điểm)

40

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 (2.5 điểm) :

- Hãy trình bày ý tưởng của giải thuật tìm kiếm tuyến tính và cho biết độ phức tạp của giải thuật. (1 điểm)
- Trình bày các bước (về từng bước) giải thuật tìm kiếm tuyến tính thực hiện tìm giá trị  $X=5$  trong mảng 6 số nguyên có giá trị: 81; 90; 62; 65; 12; 42 (1.5 điểm)

41

## Đề Minh Họa – Sắp xếp Tìm kiếm

Câu 1 (3 điểm) : Cho dãy số ban đầu như sau : 17 72 99 32 58 70 44 12 23

Hãy thực hiện các yêu cầu sau:

- Hãy trình bày các bước thực hiện **thuật toán chọn trực tiếp** (1.5 d)
- Vẽ hình từng bước thực hiện của thuật toán trên để sắp xếp dãy số theo thứ tự **giảm dần** (không cần lập trình) (1.5 d)

42

## 2. Sắp xếp

Selection Sort , InsertionSort,  
QuickSort, Merge Sort, HeapSort

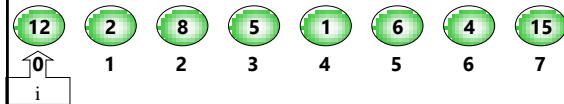
- Viết code – nêu ý tưởng
- Tính chất và độ phức tạp
- Chạy tay

43

### Selection Sort – Minh Họa

Vị trí nhỏ nhất(0,7)

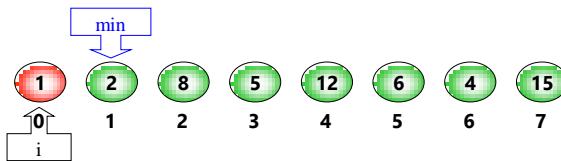
Swap(a[0], a[4])



### Selection Sort – Minh Họa

Vị trí nhỏ nhất(1,7)

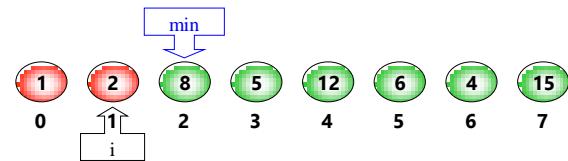
Swap(a[1], a[1])



### Selection Sort – Minh Họa

Vị trí nhỏ nhất(2,7)

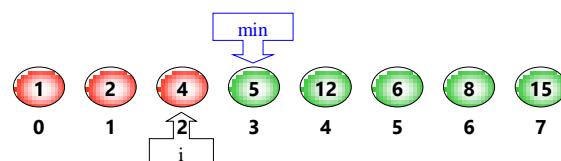
Swap(a[2], a[6])



### Selection Sort – Minh Họa

Vị trí nhỏ nhất(3,7)

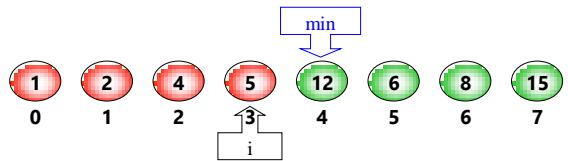
Swap(a[3], a[3])



### Selection Sort – Minh Họa

Vị trí nhỏ nhất(4,7)

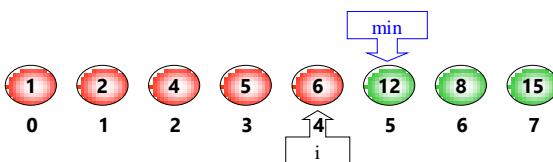
Swap(a[4], a[5])



### Selection Sort – Minh Họa

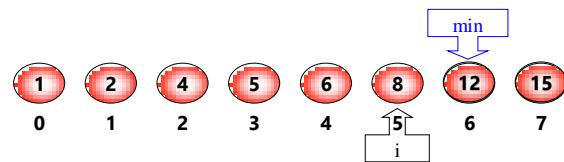
Vị trí nhỏ nhất(5,7)

**Swap(a[5], a[6])**



### Selection Sort – Minh Họa

Vị trí nhỏ nhất(6,7)



### Selection Sort – Ý Tưởng

Chọn phần tử nhỏ nhất cho các vị trí 0, 1, ..., n-1, cụ thể:

- **Lần chọn 0:**

- Chọn phần tử nhỏ nhất ( $a[min]$ ) trong khoảng vị trí từ 0 đến  $n-1$
- Đổi chỗ hai nút tại vị trí min và 0

- **Lần chọn 1:**

- Chọn phần tử nhỏ nhất ( $a[min]$ ) trong khoảng vị trí từ 1 đến  $n-1$
- Đổi chỗ hai nút tại vị trí min và 1

- **Lần chọn i:**

- Chọn phần tử nhỏ nhất ( $a[min]$ ) trong khoảng vị trí từ  $i$  đến  $n-1$
- Đổi chỗ hai nút tại vị trí min và  $i$

→ lần chọn cuối  $n-2$

### Selection Sort

**Input:** mảng  $a: a[0], a[1], \dots, a[n-1]$  chưa sắp xếp

**Output:** mảng  $a$  đã sắp xếp

Các bước thực hiện:

for ( $i=0; i < n-1; i++$ )

{

b1: Tìm phần tử  $a[min]$  nhỏ nhất trong dãy hiện hành từ  $a[i]$  đến  $a[n-1]$

b2: Hoán vị  $a[min]$  và  $a[i]$

}

### Selection Sort

```
void SelectionSort (int a[],int N )
{
    int min, temp; // chỉ số phần tử nhỏ nhất trong dãy hiện hành
    for (int i=0; i<n-1 ; i++)
    {
        // tìm phần tử nhỏ nhất
        min = i;
        for (int j = i + 1; j < n ; j++)
            if (a[j] < a[min])
                min = j;

        // đổi chỗ a[i] và a[min]
        temp = a[i]; a[min] = temp; a[i] = a[min];
    }
}
```

### Selection Sort – Độ Phức Tạp

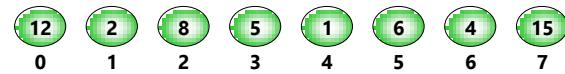
Trường hợp	Số lần so sánh $C(n)$	Số lần hoán vị
Tốt nhất	- O lượt thứ $i$ , bao giờ cũng cần $(n-i-1)$ lần so sánh để xác định phần tử nhỏ nhất hiện hành $C(n) = \sum_{i=0}^{n-1} n - i - 1$ $= (n-1) + (n-2) + \dots + (n-i-1) + \dots + 1$ $= n(n-1)/2$	0
Xấu nhất	$n(n-1)/2$	$M(n) = n-1$

Độ phức tạp tính toán :  $T(n) = O(n^2)$

### Insertion Sort – Ý Tưởng

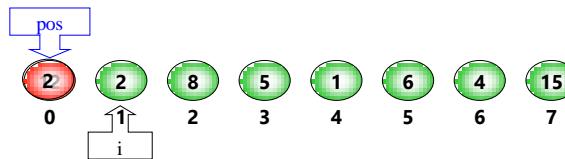
Thuật toán Insertion sort sắp xếp dựa trên tư tưởng là không gian cần sắp xếp đã được sắp xếp một phần và ta chỉ cần thêm một giá trị mới vào không gian này sao cho không gian mới được sắp xếp

### Insertion Sort – Minh Họa



### Insertion Sort – Minh Họa

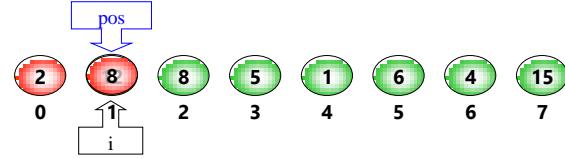
Insert  $a[1]$  into (0,0)



X

### Insertion Sort – Minh Họa

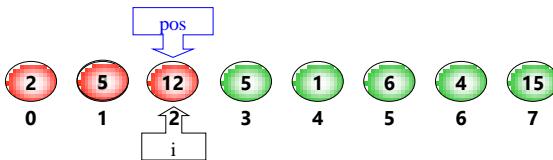
Insert  $a[2]$  into (0, 1)



X

### Insertion Sort – Minh Họa

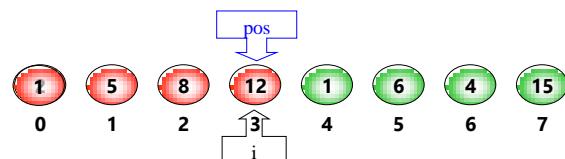
Insert  $a[3]$  into (0, 2)



X

### Insertion Sort – Minh Họa

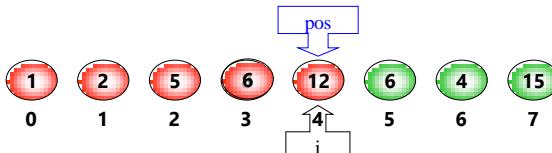
Insert  $a[4]$  into (0, 3)



X

### Insertion Sort – Minh Họa

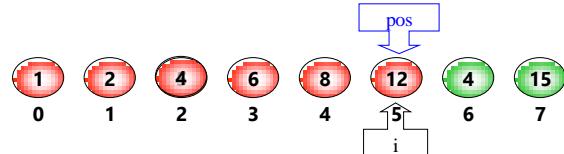
Insert  $a[5]$  into (0, 4)



X

### Insertion Sort – Minh Họa

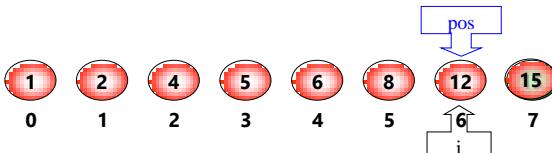
Insert  $a[6]$  into (0, 5)



X

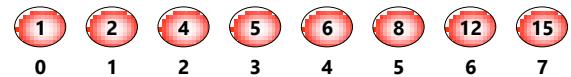
### Insertion Sort – Minh Họa

Insert  $a[8]$  into (0, 6)



X

### Insertion Sort – Minh Họa



### Insertion Sort – Ý Tưởng

- Giả sử i phần tử đầu tiên  $a_0, a_1, \dots, a_{i-1}$  đã có thứ tự.
  - Tìm cách chèn phần tử  $a_i$  vào vị trí thích hợp của đoạn đã được sắp để có đoạn mới  $a_0, a_1, \dots, a_i$  trở nên có thứ tự.
- Ban đầu, xem đoạn gồm 1 phần tử  $a_0$  đã được sắp
- Lần chèn 1: chèn  $a_1$  vào đoạn  $a_0$  để có đoạn  $a_0, a_1$  được sắp
  - Lần chèn 2: chèn  $a_2$  vào đoạn  $a_0, a_1$  để có đoạn  $a_0, a_1, a_2$  được sắp
  - Tiếp tục cho đến khi thêm xong  $a_{n-1}$  vào đoạn  $a_0, a_1, \dots, a_{n-2}$  sẽ có  $a_0, a_1, \dots, a_{n-1}$  được sắp.

### Insertion Sort – Ý Tưởng

**Input:** mảng a:  $a[0], a[1], \dots, a[n-1]$  chưa sắp xếp

**Output:** mảng a đã sắp xếp

**Các bước thực hiện:**

```
for (i=1; i<n; i++)           // giả sử có đoạn a[0] đã được sắp
{
    b1: x = a[i]               // x là phần tử cần chèn
    vào                                b2: Tìm vị trí  $\textcolor{red}{j}$  thích hợp trong đoạn a[0] đến a[i-1] để chèn a[i]
    b3: Dời chỗ các phần tử từ a[j] đến a[i-1] sang phải 1 vị trí để
        dành chỗ cho a[i]
    b4: a[i] = x;                // có đoạn a[0..a[i]] đã được sắp
```

### Insertion Sort – Cài đặt

```

void InsertionSort (int a[], int n )
{
    int i, j, x;
    for (i=1 ; i<n ; i++)           //đoạn a[0] đã sắp
    {
        x = a[i];                  //lưu giá trị a[i] tránh bị ghi đè khi dời chỗ
        // tim vị trí chèn x kết hợp với chỗ các phần tử sẽдвиг sau x trong dãy mới
        for (j = i-1, j >= 0 && x < a[j]; j--) //chèn được
            a[j+1] = a[j];             //dời chỗ

        a[j+1] = x;                 // chèn x vào vị trí hợp lệ
    }
}

```

### Quick Sort

Thuật toán quick sort 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 một phần tử thì sắp xếp không gian con này bằng thuật toán Quick Sort.

68

### Quick Sort-Ý tưởng

Giải thuật QuickSort sắp xếp dãy  $a_1, a_2, \dots, a_N$  dựa trên việc phân hoạch dãy ban đầu thành 3 phần :

- Phần 1: Gồm các phần tử có giá trị bé hơn  $x$
- Phần 2: Gồm các phần tử có giá trị bằng  $x$
- Phần 3: Gồm các phần tử có giá trị lớn hơn  $x$   
với  $x$  là giá trị của một phần tử tùy ý trong dãy ban đầu.

### Quick Sort – Ý tưởng

▪ Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 đoạn:

- 1.  $a_k \leq x$ , với  $k = 1 \dots j$
- 2.  $a_k = x$ , với  $k = j+1 \dots i-1$
- 3.  $a_k \geq x$ , với  $k = i \dots N$

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

### Quick Sort – Ý tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 chỉ có 1 phần tử : đã có thứ tự  
→ khi đó dãy con ban đầu đã được sắp.

### Quick Sort – Ý tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- Đoạn thứ 2 đã có thứ tự.

- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn 1, 3 được sắp.

- Để sắp xếp các đoạn 1 và 3, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày ...

## Quick Sort – Ý tưởng

- Bước 1:** Nếu  $left \geq right$  //dãy có ít hơn 2 phần tử  
Kết thúc; //dãy đã được sắp xếp
- Bước 2:** Phân hoạch dãy  $a_{left} \dots a_{right}$  thành các đoạn:  $a_{left}.. a_j, a_{j+1}.. a_{l-1}, a_l.. a_{right}$   
*Đoạn 1  $\leq x$*   
*Đoạn 2:  $a_{j+1}.. a_{l-1} = x$*   
*Đoạn 3:  $a_l.. a_{right} \geq x$*
- Bước 3: Sắp xếp đoạn 1:**  $a_{left}.. a_j$
- Bước 4: Sắp xếp đoạn 3:**  $a_l.. a_{right}$

## Quick Sort – Ý tưởng

- Bước 1 :** Chọn tùy ý một phần tử  $a[k]$  trong dãy là giá trị mốc ( $l \leq k \leq r$ ):  
 $x = a[k]; i = l; j = r;$
- Bước 2 :** Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ :
  - Bước 2a :** Trong khi ( $a[i] < x$ )  $i++;$
  - Bước 2b :** Trong khi ( $a[j] > x$ )  $j--;$
  - Bước 2c :** Nếu  $i < j$  Swap( $a[i], a[j]$ );
- Bước 3 :** Nếu  $i < j$ : Lặp lại Bước 2.  
Ngược lại: Dừng

## Quick Sort – Ý tưởng

- Chọn 1 phần tử bất kỳ trong mảng làm **nút trục**, xác định vị trí hợp lệ của nút này trong mảng (vị trí **pivot**).
- Phân hoạch các phần tử còn lại sao cho từ vị trí 0 đến pivot-1 đều có giá trị nhỏ hơn hoặc bằng nút trục, từ vị trí pivot+1 đến n-1 lớn hơn nút trục.
- Quá trình tiếp tục như thế với 2 mảng con này.

## Quick Sort – Giải thuật

**Input:** - mảng a:  $a[0], a[1], \dots, a[n-1]$  chưa sắp xếp giữa 2 vị trí left và right  
- 2 giá trị left, right

**Output:** mảng a đã sắp xếp giữa 2 vị trí left và right

**Các bước thực hiện:**

1. if ( $left >= right$ )	Kết thúc giải thuật <i>// điều kiện dừng, dãy có ít hơn 2 phần tử</i>
2. if ( $left < right$ )	// bước đệ quy
2.1. Phân hoạch dãy $a_{left} \dots a_{right}$ thành các dãy con: /* phân hoạch dãy làm 3 phần: */	<i>dãy con 1: <math>a[i] \leq a[pivot], i &lt; pivot</math> nút làm trục: <math>a[pivot]</math> dãy con 2: <math>a[i] &gt; a[pivot], i &gt; pivot */</math></i>
2.2. Gọi đệ quy: <b>QuickSort(a[], left, pivot - 1)</b>	
2.3. Gọi đệ quy: <b>QuickSort(a[], pivot + 1, right)</b>	

## Partition – Phân hoạch

- Input: dãy các nút từ vị trí left đến right
- Output: đổi chỗ các nút sao cho các nút nhỏ hơn hoặc bằng nút trục đưa về trước nút trục, các nút lớn hơn đưa về sau nút trục
- Các bước thực hiện:
  - Chọn tùy ý một phần tử  $a[k]$  trong dãy là giá trị mốc ( $left \leq k \leq right$ ):  
 $x = a[k]; i = left; j = right;$
  - Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ :  
*// xử lý 2 hướng quét để đổi chỗ các cặp sai vị trí*

## Partition – Phân hoạch

- Quét hướng từ left: xuất phát từ left và tăng dần, dừng lại khi gặp nút lớn hơn hay bằng nút trục, ghi nhận vị trí i lúc này
- Quét hướng từ right: xuất phát từ right và giảm dần, dừng lại khi gặp nút nhỏ hơn hay bằng nút trục, ghi nhận vị trí j lúc này.
- if ( $i \leq j$ ) Đổi chỗ 2 nút tại 2 vị trí i và j, tăng i và giảm j.  
*// cứ tiếp tục quét theo 2 hướng và đổi chỗ các cặp nút sai vị trí.*
- Nếu  $i \leq j$ : Lặp lại Bước 2.  
Ngược lại: Dừng

## Quick Sort – Cài đặt

```
void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2]; // chọn phần tử chính giữa làm trục
    i = left; j = right;
    while(i < j) // phân hoạch
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Doicho(a[i],a[j]);
            i++; j--;
        }
    }
    if(left < j) QuickSort(a, left, j); // gọi đệ quy
    if(i < right) QuickSort(a, i, right);
}
```

## Quick Sort – Chạy tay

Chạy thuật toán Quick Sort sắp xếp mảng sau theo chiều tăng dần

M={3,5,33,1,8,12,4,23,8}

80

## Quick Sort – Cài đặt

M={3,5,33,1,8,12,4,23,8}

a = M, left = 0, right = n-1 = 8  
x = a[4] = 8  
i = 0; j = 8

```
void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2]; // chọn phần tử chính giữa làm trục
    i = left; j = right;
    while(i < j) // phân hoạch
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Doicho(a[i],a[j]);
            i++; j--;
        }
    }
    if(left < j) QuickSort(a, left, j); // gọi đệ quy
    if(i < right) QuickSort(a, i, right);
}
```

i=0 < j=8 đúng

a[i=0] = 3 < 8 đúng → i=1

a[i=1] = 5 < 8 đúng → i=2

a[i=2] = 3 > 8 sai → j=2

a[j=8] = 8 > 8 sai → j=8

i=2 <= j=9 đúng  
Đổi chỗ a[i=2] cho a[j=9]

a={3,5,8,1,8,12,4,23,33}

i=3 < j=8 đúng

i=3 < j=7 đúng

## Quick Sort – Chạy tay

a={3,5,8,1,8,12,4,23,33}

a = M, left = 0, right = n-1 = 8  
x = a[4] = 8  
i = 0; j = 8

```
void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2]; // chọn phần tử chính giữa làm trục
    i = left; j = right;
    while(i < j) // phân hoạch
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Doicho(a[i],a[j]);
            i++; j--;
        }
    }
    if(left < j) QuickSort(a, left, j); // gọi đệ quy
    if(i < right) QuickSort(a, i, right);
}
```

i=3 < j=7 đúng

a[i=3] = 1 < 8 đúng → i=4

a[i=4] = 8 > 8 sai → i=4

a[j=7] = 23 > 8 đúng → j=6

a[j=6] = 4 > 8 sai → j=7

i=4 <= j=7 đúng

Đổi chỗ a[i=4] cho a[j=6]

a={3,5,8,1,4,12,8,23,33}

i=5, j=5

i=5 < j=5 Sai

left = 0 < j=5 đúng Quicksort(a,0,5)

i=5 < right=8 đúng Quicksort(a,5,8)

## Quick Sort – Chạy tay

a={3,5,8,1,4,12,8,23,33}

Quicksort(a,0,5)

a = a, left = 0, right = 5  
x = a[2] = 8  
i = 0; j = 5

```
void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2]; // chọn phần tử chính giữa làm trục
    i = left; j = right;
    while(i < j) // phân hoạch
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Doicho(a[i],a[j]);
            i++; j--;
        }
    }
    if(left < j) QuickSort(a, left, j); // gọi đệ quy
    if(i < right) QuickSort(a, i, right);
}
```

i=0 < j=5 đúng

a[i=0] = 3 < 8 đúng → i=1

a[i=1] = 5 < 8 đúng → i=2

a[i=2] = 8 > 8 sai → i=2

a[j=5] = 12 > 8 đúng → j=4

a[j=4] = 4 > 8 sai → j=4

i=2 <= j=4 đúng

Đổi chỗ a[i=2] cho a[j=4]

a={3,5,4,1,8,12,8,23,33}

i=3 < j=3 Sai

left = 0 < j=3 đúng Quicksort(a,0,3)

i=3 < right=5 đúng Quicksort(a,0,3)

## Quick Sort – Chạy tay

a={3,5,4,1,8,12,8,23,33}

Quicksort(a,0,5)

a = a, left = 0, right = 5  
x = a[2] = 8  
i = 0; j = 5

```
void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2]; // chọn phần tử chính giữa làm trục
    i = left; j = right;
    while(i < j) // phân hoạch
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Doicho(a[i],a[j]);
            i++; j--;
        }
    }
    if(left < j) QuickSort(a, left, j); // gọi đệ quy
    if(i < right) QuickSort(a, i, right);
}
```

i=0 < j=5 đúng

a[i=0] = 3 < 8 đúng → i=1

a[i=1] = 5 < 8 đúng → i=2

a[i=2] = 8 > 8 sai → i=2

a[j=5] = 12 > 8 đúng → j=4

a[j=4] = 4 > 8 sai → j=4

i=2 <= j=4 đúng

Đổi chỗ a[i=2] cho a[j=4]

a={3,5,4,1,8,12,8,23,33}

i=3, j=3

left = 0 < j=3 đúng Quicksort(a,0,3)

i=3 < right=5 đúng Quicksort(a,0,3)

## Quick Sort – Chạy tay

a={3,5,4,1,8,12,8,23,33}

```
Quicksort(a,0,3)
{
    a = a, left =0, right = 3
    x = a[1] =5
    i = 0; j=3

    void Quicksort(int a[], int left, int right)
    {
        int i, j, x;
        x = a[(left+right)/2]; // chọn phần tử chính giữa làm trục
        i = left; j = right;
        while(i < j) // phán hoạch
        {
            while(a[i] < x) i++;
            while(a[j] > x) j--;
            if(i <= j)
            {
                DoiChoi(a[i],a[j]);
                i++; j--;
            }
        }
        if(left < j) Quicksort(a, left, j); // gọi đệ quy
        if(r < right) Quicksort(a, i, right);
    }
}
```

i=0< j=3 đúng  
a[i=0] =3<5 đúng → i=1  
a[i=1] =5>5 sai → i=1  
a[j=3] =1>5 sai → j=3  
i=1< j=3 đúng  
Đổi chỗ a[i=1] cho a[j=3]  
a={3,1,4,5,8,12,8,23,33}  
i=2, j=2  
i=2< j=2 Sai  
left = 0< j=2 đúng Quicksort(a,0,2)  
i=2 < right=3 đúng Quicksort(a,2,3)

## Quick Sort – Cài đặt

a={3,1,4,5,8,12,8,23,33}

```
Quicksort(a,0,2)
```

```
a = a, left =0, right = 2
x = a[1] =1
i = 0; j =2
```

```
void Quicksort(int a[], int left, int right)
```

```
{
    int i, j, x;
```

```
x = a[(left+right)/2]; // chọn phần tử chính giữa làm trục
```

```
i = left; j = right;
```

```
while(i < j) // phán hoạch
```

```
{
    while(a[i] < x) i++;
    while(a[j] > x) j--;
    if(i <= j)
    {
        DoiChoi(a[i],a[j]);
        i++; j--;
    }
}
if(left < j) Quicksort(a, left, j); // gọi đệ qui
if(r < right) Quicksort(a, i, right);
```

i=0< j=2 đúng  
a[i=0] =3<1 sai → i=0

a[j=2] =4>1 đúng → j=1

a[j=1]=1>1 sai → j=1

i=0< j=1 đúng

Đổi chỗ a[i=0] cho a[j=1]

a={1,3,4,5,8,12,8,23,33}

i=1, j=0

i=1 < j=0 Sai

left = 0< j=0 Sai Dừng

i=1 < right=2 đúng Quicksort(a,1,2)

## Quick Sort – Đánh giá

Hiệu quả thực hiện của giải thuật QuickSort phụ thuộc vào việc chọn giá trị mốc

Trường hợp	Số lần so sánh	Số lần đổi chỗ
Tốt nhất	Nút trục lục nào cũng ở giữa dãy, 2 dãy con lúc nào cũng cân đối nhau	- Cần log <sub>2</sub> n lần phân hoạch C(n) = (1*n) + (2*n/2) + (4*n/4) +...+(n*n/n) = nlg n
Xấu nhất	Nút trục lục nào cũng ở đầu hay cuối dãy, 2 dãy con lệch nhau và suy biến thành 1 dãy con có số phần tử bắt đì 1	- Cần n lần phân hoạch C(n) = n + (n-1) + (n-2) +...+ 1 = n(n+1)/2

## Quick Sort – Độ phức tạp

Trường hợp	Độ phức tạp
Tốt nhất	$n * \log(n)$
Trung bình	$n * \log(n)$
Xấu nhất	$n^2$

## Quick Sort – Nhận xét

- Phức tạp hơn Bubble Sort nhưng hiệu quả hơn, trong trường hợp tốt nhất độ phức tạp O(nlogn).
- Quick Sort thích hợp cho dãy ban đầu chưa có thứ tự.
- Quick Sort **kém hiệu quả** khi dãy ban đầu gần có thứ tự, đặc biệt trường hợp xấu nhất là dãy đã có thứ tự (tăng hoặc giảm)

## Quick Sort

Câu 1: (2.5 điểm)

a. Trình bày các bước giải thuật sắp xếp Quick sort (không viết chương trình) để sắp xếp mảng số nguyên N phần tử **giảm dần**, cho biết độ phức tạp giải thuật. (1.5 điểm)

b. Trình bày các bước (vẽ từng bước) áp dụng giải thuật trong câu 1.a để sắp xếp mảng số nguyên {10, 5, 30, 70, 40, 80, 90} giảm dần. (1 điểm)

Mảng ban đầu :

10	5	30	70	40	80	90
----	---	----	----	----	----	----

### Quick Sort

Mảng ban đầu :

10	5	30	70	40	80	90
----	---	----	----	----	----	----

Lần 1 : Chọn phần tử 70 làm phần tử phân hoạch:

90	80	70	30	40	5	10
----	----	----	----	----	---	----

Lần 2 : Chọn phần tử 80 làm phần tử phân hoạch [đoạn 90,80]

90	80	70	30	40	5	10
----	----	----	----	----	---	----

Lần 3 : Chọn phần tử 5 làm phần tử phân hoạch [đoạn 30,40,5,10]

90	80	70	30	40	10	5
----	----	----	----	----	----	---

Lần 4 : Chọn phần tử 40 làm phần tử phân hoạch [đoạn 30,40,10]

90	80	70	40	30	10	5
----	----	----	----	----	----	---

Kết thúc : Mảng sắp giảm : 90, 80, 70, 40, 30, 10, 5

91

### Merge Sort – Ý tưởng

- Thuật toán Merge Sort chia không gian cần sắp xếp thành 2 không gian con.
  - + Nếu không gian con thứ nhất có nhiều hơn một phần tử thì sắp xếp không gian con này bằng thuật toán Merge Sort.
  - + Nếu không gian con thứ hai có nhiều hơn một phần tử thì sắp xếp không gian con này bằng thuật toán Merge Sort.
- Trộn 2 không gian con đã được sắp xếp lại với nhau.

92

### Merge Sort – Trộn

```
void MergeSort(int []M,int left,int right)
{
    if (left >= right) return;
    int mid = (left + right) / 2;
    MergeSort(M, left, mid);
    MergeSort(M, mid + 1, right);
    Merge(M, left, mid, right);
}
```

93

### Merge Sort – Trộn

```
void Merge(int []M,int left,int mid,int right)
{
    int[] Temp = new int[right - left + 1];
    int pos = 0;
    int i = left;
    int j = mid + 1;
    while(!(i==mid && j>right))
    {
        if((i<=mid && j <=right && M[i]<M[j]) || j>right)
            Temp[pos++] = M[i++];
        else
            Temp[pos++] = M[j++];
    }
    for(i=0;i<Temp.Length;i++)
        M[left + i] = Temp[i];
}
```

94

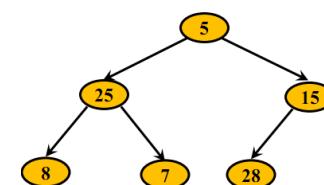
### Merge Sort – Nhận xét

- Không tối ưu bộ nhớ vì có dùng mảng tạm trong quá trình trộn.
- Nhanh hơn Quick Sort vì thời gian thực hiện Merge Sort có bậc nhỏ hơn  $O(n\log n)$ , còn trong trường hợp tốt nhất Quick Sort mới có bậc  $O(n^2)$ .
- Thường dùng Merge Sort để sắp xếp khối dữ liệu lớn ở bộ nhớ ngoài.

95

### Heap Sort

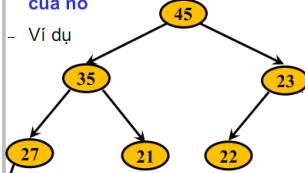
- Định nghĩa Heap sơ khởi: Heap là một cây nhị phân đầy đủ
- Ví dụ:



96

## Heap Sort

- Định nghĩa Heap sơ khởi: Heap là một cây nhị phân đầy đủ
- Mỗi nút trong Heap chứa một giá trị có thể so sánh với giá trị của nút khác.
- Đặc điểm của Heap là giá trị của mỗi nút  $\geq$  giá trị của các nút con của nó
- Ví dụ



97

## Định nghĩa Heap

**Heap là một cây nhị phân thỏa các tính chất sau:**

- + Heap là một cây đầy đủ;
- + Giá trị của mỗi nút không bao giờ bé hơn giá trị của các nút con

Hệ quả:

- + Nút lớn nhất là ... ?

98

## Heap Sort – Ý tưởng

- Chuyển dãy cần sắp xếp thành cấu trúc Heap
  - Đầu tiên xem heap chỉ có 1 nút ( $a[0]$ ).
  - Gọi tác vụ Insert để thêm nút  $a[1]$  vào heap, ta được heap có 2 nút ( $a[0], a[1]$ )
  - Tiếp tục gọi tác vụ Insert để lần lượt thêm các nút  $a[i]$  vào heap, đến khi chuyển dãy ban đầu thành heap  $n$  nút.
- Lần lượt chuyển nút gốc của heap đưa về vị trí thích hợp (về phía cuối)

99

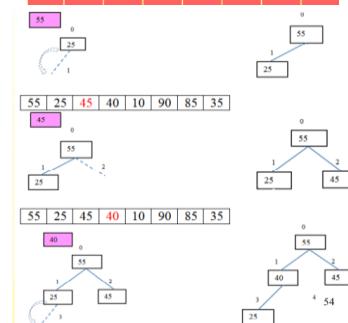
## Heap Sort – Ý tưởng

- Lần lượt chuyển nút gốc của heap đưa về vị trí thích hợp (về phía cuối)
  - Đầu tiên heap có  $n$  nút ( $a[0], a[1], \dots, a[n-1]$ ).
  - Gọi tác vụ removeroot để xóa nút gốc (nút lớn nhất), được heap  $n-1$  nút, chuyển nút gốc bị xóa thành  $a[n-1]$ .
  - Với heap  $n-1$  nút ( $a[0], a[1], \dots, a[n-2]$ ), gọi tác vụ removeroot để xóa nút gốc, được heap  $n-2$  nút, chuyển nút gốc bị xóa thành  $a[n-2]$ .
  - Tiếp tục chuyển nút gốc về vị trí thích hợp, cuối cùng ta được dãy đã sắp thứ tự.

100

## Heap Sort -Ví dụ minh họa – Tạo Heap

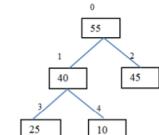
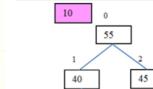
25 | 55 | 45 | 40 | 10 | 90 | 85 | 35



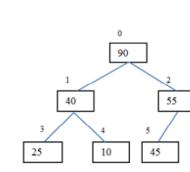
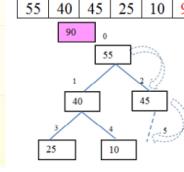
101

## Heap Sort -Ví dụ minh họa – Tạo Heap

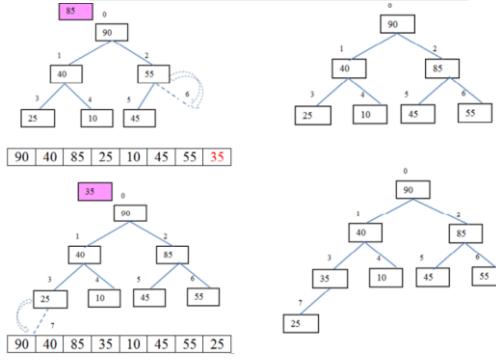
35 | 40 | 45 | 25 | 10 | 90 | 85 | 35



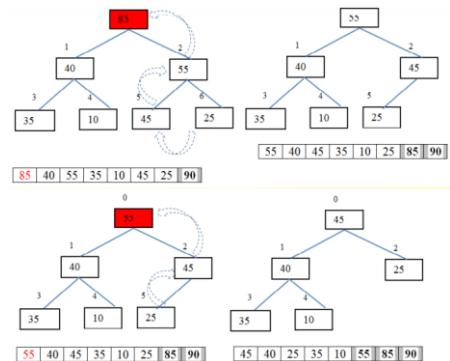
55 | 40 | 45 | 25 | 10 | 90 | 85 | 35



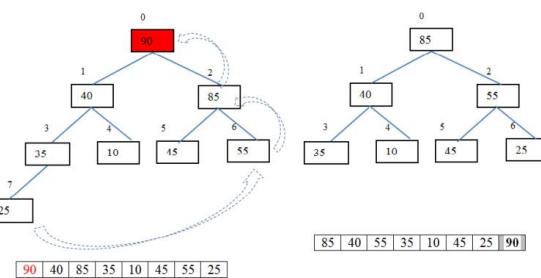
### Heap Sort - Ví dụ minh họa – Tạo Heap



### Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



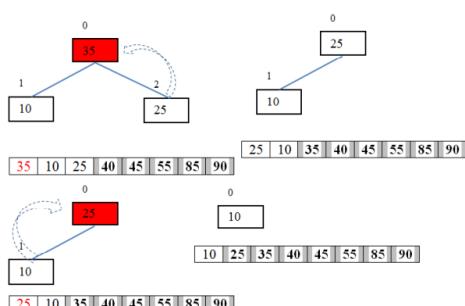
### Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



103

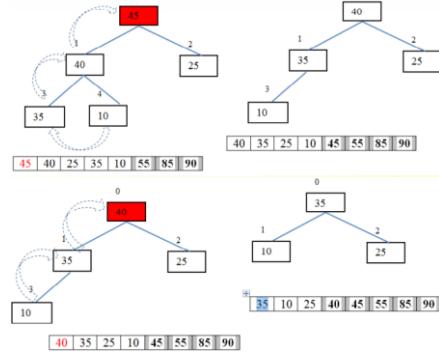
104

### Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



105

### Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



106

### Heap Sort – Cài đặt

```
void HeapSort(int a[], int n)
{
    int i, x, s, f, cuoiheap;
    /* I. chuyển đổi thành cây trúc heap bằng cách tuân tự thêm nút a[i] vào heap*/
    for (i = 1; i < n; i++)
    {
        x = a[i];
        /* insert (a[], i, x)*/
        s = i;
        f = (s-1)/2;
        while (s > 0 && a[f] < x) // lần theo đường đi từ vị trí i về vị trí 0 để lắn lượt
            // kéo các nút có giá trị nhỏ hơn x lên 1 mức
        {
            a[s] = a[f];
            s = f;
            f = (s-1)/2;
        }
        a[s] = x;
        // khi gặp nút đầu tiên có giá trị >= x thì đưa nút x
        // vào vị trí nút con của nút này
    }
}
```

108

## Heap Sort – Cài đặt

```

/*2. Lần lượt xóa nút gốc (a[0]), đưa về vị trí thích hợp*/
for (i = n-1; i > 0; i--)
{
    cuoicheap = a[i];           // lưu lại giá trị nút cuối của heap a[n-1]
    a[i] = a[0];                // chuyển nút gốc a[0] bị xóa về phía cuối
    /* removeroot(a[], i+1)*/
    f = 0;
    s = largeson(f, i-1);
    while (s >= 0)
    {
        a[f] = a[s];
        f = s;
        s = largeson (f, i-1);
    }
    a[f] = cuoicheap;
} //end for
} // kết thúc

```

109

## Heap Sort – Nhận xét

- Chỉ thao tác trên 1 mảng a[] : đầu tiên chuyển a[] thành cấu trúc heap, sau đó lần lượt chuyển các nút gốc của heap về vị trí phù hợp trên a[].
- Không phụ thuộc vào tính thứ tự của dãy cần sắp.
- Một trong những giải thuật sắp xếp nội hiệu quả nhất.

**Độ phức tạp tính toán : O(nlgn)**

110

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 (3 điểm):

- a) Trình bày giải thuật sắp xếp tăng dần theo phương pháp *Chèn trực tiếp* (*Insertion Sort*).

- b) Sử dụng phương pháp *Chèn trực tiếp* (*Insertion Sort*) để sắp xếp tăng dần dãy số nguyên sau:

14	9	7	2	8	10
----	---	---	---	---	----

Câu 1 : (3.5 điểm)

Anh/chị hãy thực hiện :

- a. Trình bày giải thuật **Chọn trực tiếp** để sắp xếp một mảng các số nguyên giảm dần. (1.5 điểm)  
b. Trình bày các bước (về từng bước) thực hiện theo giải thuật câu a đối với mảng X gồm các số nguyên theo thứ tự từ trái qua phải, như sau: 91, 12, 30, 1, 83, 20 (2 điểm)

111

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 : (2.5 điểm)

- a. Trình bày các bước giải thuật sắp xếp Quick sort (không viết chương trình) để sắp xếp mảng số nguyên N phần tử **giảm dần**, cho biết độ phức tạp giải thuật.  
b. Trình bày các bước (về từng bước) áp dụng giải thuật trong câu a để sắp xếp mảng số nguyên {10, 5, 30, 70, 40, 80, 90} giảm dần.

Câu 1 : (3.5 điểm)

Anh/chị hãy thực hiện :

- a. Trình bày giải thuật **Chọn trực tiếp** để sắp xếp một mảng các số nguyên giảm dần. (1.5 điểm)  
b. Trình bày các bước (về từng bước) thực hiện theo giải thuật câu a đối với mảng X gồm các số nguyên theo thứ tự từ trái qua phải, như sau: 91, 12, 30, 1, 83, 20 (2 điểm)

Câu 1 (3 điểm) : Cho dãy số ban đầu như sau : 17 72 99 32 58 70 44 12 23

Hãy thực hiện các yêu cầu sau:

- a. Hãy trình bày các bước thực hiện **thuật toán chọn trực tiếp** (1.5 đ)  
b. Vẽ hình từng bước thực hiện của thuật toán trên để sắp xếp dãy số theo thứ tự giảm dần (không cần lập trình) (1.5 đ)

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 (6 điểm) :

- a) Anh/chị hãy trình bày giải thuật toán **chọn trực tiếp** (*Selection Sort*) để sắp xếp một mảng các số nguyên tăng dần. (2 điểm)  
b) Viết hàm cài đặt thuật toán trên bằng ngôn ngữ C/C++. (2 điểm)  
c) Trình bày các bước (về từng bước) thực hiện sắp xếp theo hàm đã cài đặt ở câu b đối với mảng các số nguyên có giá trị như sau: 19 11 31 15 37 17 (2 điểm)

Câu 2 (4 điểm):

- a) Anh/chị hãy viết hàm cài đặt thuật toán **Tìm kiếm nhị phân** trên mảng số nguyên có thứ tự giảm dần bằng ngôn ngữ C/C++. (2 điểm)  
b) Trình bày các bước (về từng bước) theo hàm đã cài đặt ở câu a thực hiện tìm giá trị X=50 trong mảng các số nguyên có giá trị như sau: 90 80 60 26 24 12 (2 điểm)

113

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

Câu 1 (2.5 điểm) :

- a. Hãy trình bày ý tưởng của giải thuật tìm kiếm tuyến tính và cho biết độ phức tạp của giải thuật. (1 điểm)  
b. Trình bày các bước (về từng bước) giải thuật tìm kiếm tuyến tính thực hiện tìm giá trị X=5 trong mảng 6 số nguyên có giá trị: 81; 90; 62; 65; 12; 42 (1.5 điểm)

114

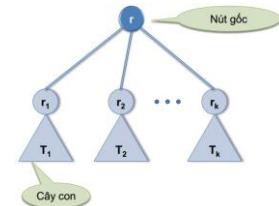
### 3. Cây và cây nhị phân

- Các khái niệm cơ bản của cây, bậc, mức, chiều cao, độ sâu
- Duyệt cây, kiểm tra tính chất của cây
- In ra các node có giá trị đặc biệt : 1 cây con, 2 cây con, giá trị max, min, node lá, node nhánh, ...

115

### CÂY – Khái Niệm

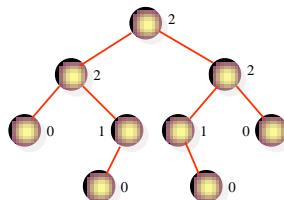
Cây là một tập hợp  $T$  **các phần tử** (gọi là **nút -node** của **cây**), trong đó có một nút đặc biệt gọi là **nút gốc**, các nút còn lại được chia thành những tập rời nhau  $T_1, T_2, \dots, T_n$  theo quan hệ phân cấp, **trong đó  $T$ , cũng là 1 cây**. Mỗi nút ở cấp  $i$  sẽ quản lý một số nút ở cấp  $i+1$ . Quan hệ này người ta gọi là quan hệ cha – con.



### CÂY – Một số khái Niệm

#### Bậc – Degree/Oder

➢ **Bậc của một nút:** là **số cây con** của nút đó



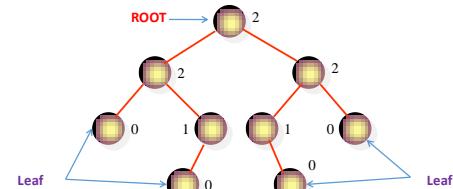
➢ **Bậc của một cây:** là **bậc lớn nhất** của các nút trong cây

➔ Cây có bậc  $n$  gọi là cây  $n$ -phân

117

### CÂY – Một số khái Niệm

➢ **Nút gốc (root):** là nút **không có nút cha**.



➢ **Nút lá (leaf):** là nút có **bậc bằng 0** (**node không có cây con**)

➢ **Nút nhánh (branch/internal):** là nút có **bậc khác 0** và **không phải là gốc**

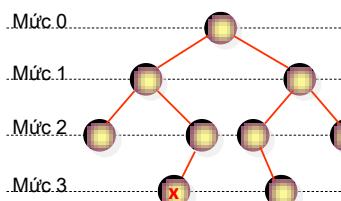
118

### CÂY – Một số khái Niệm

➢ **Mức của một nút (level):**

➢ Mức (gốc ( $T$ ) ) = 0.

➢ Gọi  $T_1, T_2, T_3, \dots, T_n$  là các cây con của  $T_0$  :  
Mức ( $T_1$ ) = Mức ( $T_2$ ) =  $\dots$  = Mức ( $T_n$ ) = Mức ( $T_0$ ) + 1.



➢ **Độ sâu của một nút (dept):** Độ dài đường đi giữa node gốc và node đó. Node ở mức  $i$  thì có độ dài  $i$

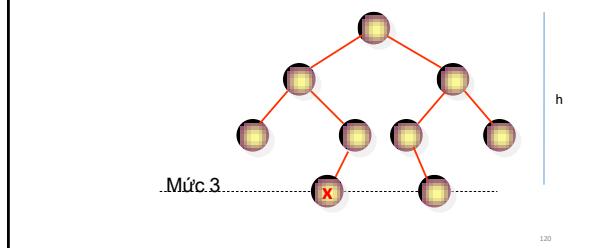
119

### CÂY – Một số khái Niệm

➢ **Chiều cao của cây (height):**

➢ **Cây rỗng** = 0.

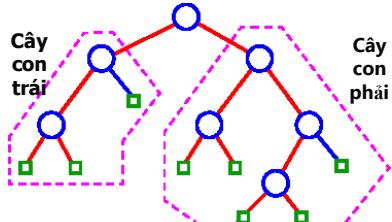
➢ **Cây khác rỗng:** Mức lớn nhất giữa các node trên cây



120

### Cây Nhị Phân (Binary Tree)

- Mỗi nút có tối đa 2 cây con



121

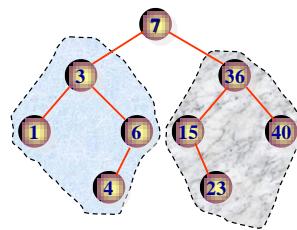
### 3. Cây nhị phân tìm kiếm

- Các thông tin cây, khai báo cây
- Duyệt cây theo các thứ tự
- Xây/ vẽ cây cây từ một dãy, từ một kết quả duyệt.
- Kiểm tra tính chất trên cây: số nguyên tố, ...
- Viết các hàm với bài toán con

122

### Cây Nhị Phân Tìm Kiếm – Binary Search Tree

- Là cây **nhị phân**
- Giá trị của một node luôn **lớn hơn giá trị của các node nhánh trái và nhỏ hơn giá trị các node nhánh phải**
- Nút có giá trị nhỏ nhất **nằm ở nút trái nhất của cây**
- Nút có giá trị lớn nhất **nằm ở nút phải nhất của cây**



123

### Tạo cây – Ví dụ

#### Bài toán 1: Vẽ cây nhị phân tìm kiếm từ dãy số

**Vẽ cây nhị phân tìm kiếm** (chỉ vẽ cây kết quả) từ dãy số nguyên khi **xây dựng cây theo thứ tự từ trái qua phải của dãy số**: 72; 67; 73; 58; 5; 4; 27; 53; 61; 32.

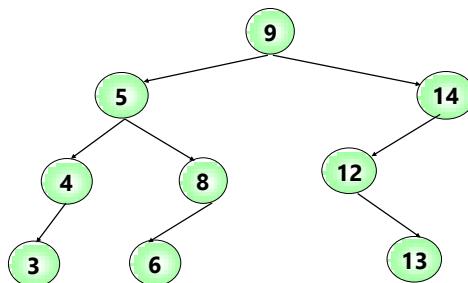
Làm theo nguyên tắc **thêm một node vào cây**:

- \* Luôn **bắt đầu so sánh từ node gốc**.
- \* Đảm bảo đặc điểm **lớn bên phải, nhỏ bên trái**.

124

### Tạo cây – Ví dụ

9, 5, 4, 8, 6, 3, 14, 12, 13

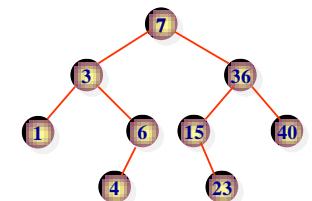


125

### Các thao tác trên Binary Search Tree

- Cho cây nhị phân tìm kiếm duyệt cây theo thứ tự yêu cầu

NLR  
NRL  
LNR  
RNL  
LRN  
RLN



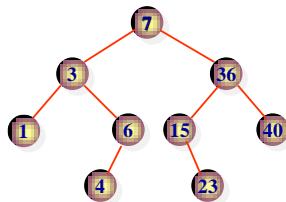
126

### Các thao tác trên Binary Search Tree

➤ Cho cây nhị phân tìm kiếm duyệt cây theo thứ tự yêu cầu

**NLR**

```
void NLR(Tree t)
{
    if(t!=NULL)
    {
        cout<<t->key;
        NLR(t->pLeft);
        NLR(t->pRight);
    }
}
```



**NLR:7,3,1,6,4,36,15,23,40**

### Tạo cây – Ví dụ

Bài toán 2: **Vẽ cây** khi biết kết quả duyệt **cây**.

Hãy **vẽ cây nhị phân tìm kiếm** T biết rằng khi duyệt cây theo thứ tự **Left –Right – Node** thì được dãy như sau: 5, 3, 7, 9, 8, 11, 6, 20, 19, 37, 25, 21, 15, 12.

Làm theo **nguyên tắc**:

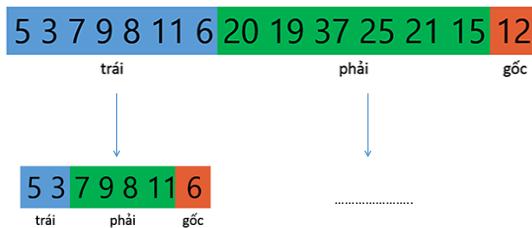
- (1) Tìm **node gốc**
- (2) Tìm đoạn lớn hơn **node gốc** sẽ là nhánh phải, **đoạn nhỏ hơn node gốc** sẽ là nhánh trái.
- (3) Với mỗi đoạn vừa tìm được, **tìm node gốc của từng đoạn** và tiếp tục tìm đoạn lớn hơn và nhỏ hơn **node gốc**.

128

### Tạo cây – Ví dụ

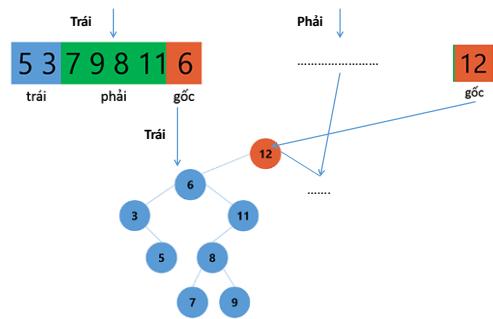
Cho kết quả duyệt LRN: 5 3 7 9 8 11 6 20 19 37 25 21 15 12

\* **Node gốc:** 12



129

### Tạo cây – Ví dụ



130

### Tạo cây – Ví dụ



131

### Các thao tác trên Binary Search Tree

➤ Tìm node có **key bằng x** – Không dùng đệ quy

```
TNode * searchNode(TREE Root, Data x)
{
    TNode *p = Root;
    while(p != NULL)
    {
        if(x == p->Key)
            return p;
        else
            if(x < p->Key)
                p = p->pLeft;
            else
                p = p->pRight;
    }
    return NULL;
}
```

132

### Các thao tác trên Binary Search Tree

➤ Tìm node có **key bằng x** – dùng đệ quy

```
TNode *SearchTNode(TREE T, int x)
{
    if(T!=NULL)
    {
        if(T->key==x)
            return T;
        else
            if(x>T->key)
                return SearchTNode(T->pRight,x);
            else
                return SearchTNode(T->pLeft,x);
    }
    return NULL;
}
```

133

### Binary Search Tree – Bài tập

Cho cây nhị phân tìm kiếm, mỗi node có giá trị nguyên, hãy định nghĩa các hàm sau:

1. In ra các node có giá trị chẵn
2. In ra các node có giá trị lớn hơn x
3. Đếm số node của cây
4. Tính độ cao của cây
5. Tìm node có giá trị x
6. Tìm node có giá trị lớn nhất
7. Tìm node có giá trị nhỏ nhất của cây con phải

134

### Binary Search Tree – Bài tập

8. Đếm số node lá (node bậc 0)
9. Đếm số node có 1 cây con (node bậc 1)
10. Đếm số node chỉ có 1 cây con phải
11. Đếm số node có 1 cây con trái
12. Đếm số node 2 cây con (node bậc 2)
13. In các node trên từng mức của cây
14. Cho biết độ dài đường đi từ gốc đến node x

135

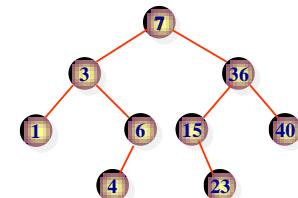
### Các thao tác trên Binary Search Tree

➤ Xóa một node trên cây

➤ Hủy 1 phần tử trên cây **phải đảm bảo điều kiện ràng buộc của Cây nhị phân tìm kiếm**

➤ Có 3 trường hợp khi hủy 1 nút trên cây

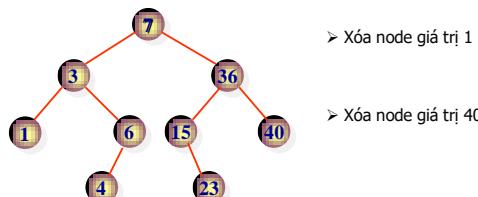
- TH1: X là nút **lá**
- TH2: X chỉ **có 1 cây con** (cây con trái hoặc cây con phải)
- TH3: X có **đầy đủ 2 cây con**



136

### Các thao tác trên Binary Search Tree

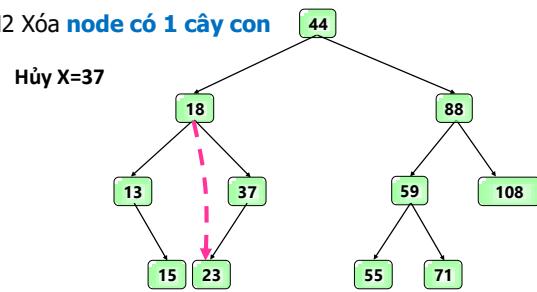
➤ TH1: Xóa **node lá** trên cây không ảnh hưởng đến các nút khác trên cây



137

### Các thao tác trên Binary Search Tree

➤ TH2 Xóa **node có 1 cây con**



138

Trước khi xoá x ta **móc nối cha của x với con duy nhất của x**

### Các thao tác trên Binary Search Tree

#### ➤ TH3 Xóa node có 2 cây con

- ❖ Tìm **phản tử thế mạng** cho phần tử cần xóa
- ❖ Có 2 cách tìm nút thế mạng
  - C1: Nút có khoá **nhỏ nhất (trái nhất)** bên **cây con phải** node cần xóa
  - C2: Nút có khoá **lớn nhất (phải nhất)** bên **cây con trái** của node cần xóa

139

### Các thao tác trên Binary Search Tree

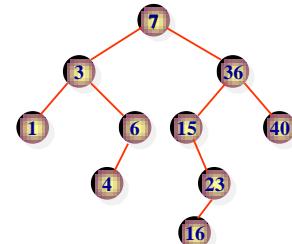
#### ➤ TH3 Xóa node có 2 cây con

**Bước 1:** Tìm node thế mạng

- **Cách 1:** Tìm node trái nhất của cây con phải
- **Cách 2:** Tìm node phải nhất của cây con trái

**Bước 2:** Thay giá trị của node thế mạng vào node cần xóa

**Bước 3:** Xóa node thế mạng

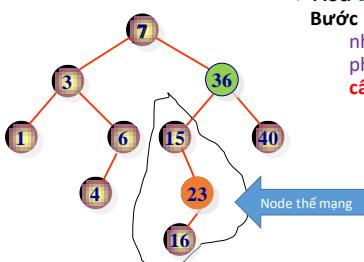


140

### Các thao tác trên Binary Search Tree

#### ➤ Xóa node có giá trị 36

**Bước 1:** Tìm node thế mạng  
nhỏ nhất cây con bên  
phải hoặc **lớn nhất** bên  
**cây con trái** node 36

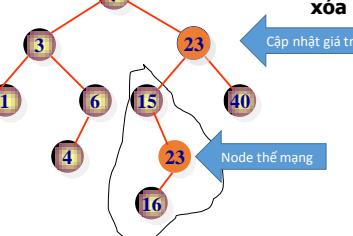


141

### Các thao tác trên Binary Search Tree

#### ➤ Xóa node có giá trị 36

**Bước 2:** Thay **giá trị** node  
thế mạng cho **node cần**  
**xóa**

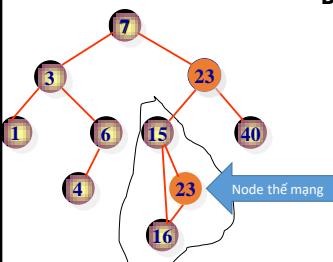


142

### Các thao tác trên Binary Search Tree

#### ➤ Xóa node có giá trị 36

**Bước 3:** Xóa node thế mạng



143

### Đề Minh Họa – **Cây nhị phân tìm kiếm**

#### Câu 2: (4 điểm)

Cho dãy số sau: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31

Hãy thực hiện các yêu cầu sau:

- a. Xây dựng **cây nhị phân tìm kiếm** từ dãy số đã cho vào cây theo thứ tự thêm các số từ trái sang phải của dãy số (1 điểm)
- b. Duyệt cây trong câu 2.a theo Node-Left-Right, Right-Left-Node (1 điểm)
- c. Xóa khỏi cây **lần lượt** các nút 8, 11, 43, 6 (vẽ hình từng trường hợp) sao cho cây vẫn là cây nhị phân tìm kiếm sau khi xóa nút. (1 điểm)
- d. Viết hàm in ra màn hình các nút trên cây có duy nhất một nút con (1 điểm)  
void Print(Tree T)

144

### Đề Minh Họa – Cây nhị phân tìm kiếm

Câu 3 : (4 điểm)

Cho dãy số như sau : {93, 11, 97, 65, 27, 70, 13, 83, 54, 101}

Anh / chị hãy thực hiện :

- Xây dựng cây nhị phân tìm kiếm từ dãy số trên **lần lượt từ trái sang phải** (1.5 điểm)
- Xóa lần lượt theo thứ tự các nút {54, 11, 93} (1.5 điểm)  
Lưu ý: Khi xóa 1 nút, cân bằng cây khi xảy ra mất cân bằng, cho biết nút bị mất cân bằng và loại mất cân bằng
- Viết hàm đếm số nút trên cây chỉ có duy nhất 1 nút con phải (1 điểm)

145

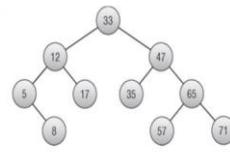
### Đề Minh Họa – Cây nhị phân tìm kiếm

Câu 2 (2 điểm) : Hãy phát biểu định nghĩa cây nhị phân tìm kiếm. Vẽ cây nhị phân tìm kiếm (chi vẽ cây kết quả) từ dãy số nguyên khi xây dựng cây theo thứ tự từ trái qua phải của dãy số: 72; 67; 73; 58; 5; 4; 27; 53; 61; 32.

146

### Đề Minh Họa – Cây nhị phân tìm kiếm

Câu 3 (3 điểm) : Cho cây nhị phân tìm kiếm T như hình bên. Anh / chị hãy thực hiện :



147

### Đề Minh Họa – Cây nhị phân tìm kiếm

Câu 2: (3 điểm)

Cho một cây nhị phân tìm kiếm T, mỗi nút là một số nguyên.

- Hãy vẽ cây nhị phân tìm kiếm T biết rằng khi duyệt cây theo thứ tự Left – Right – Node thì được dãy như sau:  
5, 3, 7, 9, 8, 11, 6, 20, 19, 37, 25, 21, 15, 12.
- Viết hàm đếm xem trong cây có bao nhiêu số chẵn, bao nhiêu số lẻ

148

### 5. Bảng băm

- Khái niệm, khai báo, so sánh với các CTDL đã học
- Xử lý các trường hợp khi bị đụng độ với các cách khác nhau.

149

### Hash Table - Hash Map

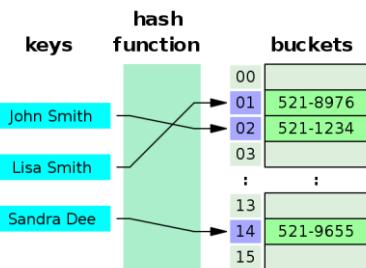
- Bảng băm là một CTDL trong đó **mỗi phần tử là một cặp (khóa, giá trị)** (key - value)

key                  value

hello	hola
red	roja
blue	azul

150

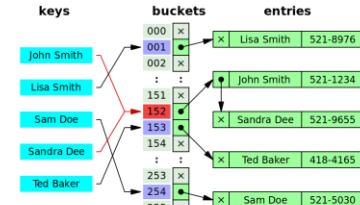
## Hash Table - Hash Map



151

## Đụng độ - Collision

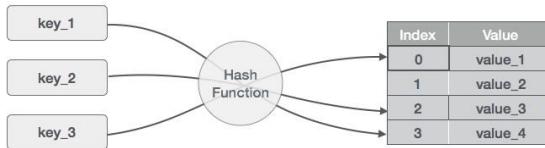
**Sự đụng độ** là hiện tượng **các khóa khác nhau nhưng băm cùng địa chỉ** như nhau.



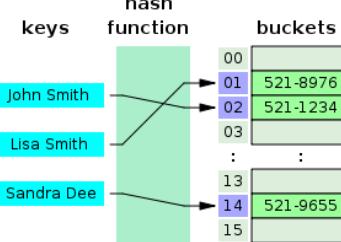
152

## Hàm băm - HASH Function

Hàm được dùng để **ánh xạ một khóa – Key vào một dãy các số nguyên** và dùng các giá trị nguyên này để truy xuất dữ liệu



153



154

## Yêu cầu của hash function

Hàm băm tốt phải thỏa mãn các điều kiện sau:

- Tính toán **nhanh**.
- Các **khoá được phân bố đều** trong bảng.
- Ít xảy **ra đụng độ** (**Colission**).
- Giải quyết trường hợp băm với các **khoá không phải là số nguyên**.

155

## Ví dụ dùng bảng Hash

Các khóa: M, O, T, V, I, D, U

$$\text{hash}(x) = \text{char\_index}(x) \% 10$$

Tìm V

Tìm F

T	0
U	1
V	2
M	3
D	4
O	5
I	6
	7
	8
	9

Không có

$$\text{char\_index: Space=0, A=1, B=2, ..., Z=27}$$

M	O	T	V	I	D	U	F
3	5	0	2	9	4	1	6

## **Phương pháp giải quyết đụng độ (Collision)**

(1) Phương pháp nối kết (**chaining method**): các phần tử bị **băm** **cùng địa chỉ** (các phần tử bị xung đột) được gom **thành một DSLK**.

(2) **Phương pháp băm lại (rehash function):** Nếu băm lần đầu bị xung đột thì băm lại lần 1, nếu bị xung đột nữa thì băm lại lần 2,... Quá trình **băm lại diễn ra cho đến khi không còn xung đột nữa**

157

### (1) Phương pháp nối kết (chaining method)

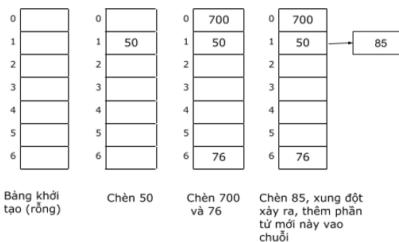
**Mỗi phần tử** của bảng băm **là một danh sách liên kết (bucket)**.

- Chèn một phần tử vào bảng băm ta phải chèn nó vào trong một danh sách liên kết.
  - Trường hợp hai phần tử có chung giá trị (hash code) thì ta sẽ chèn chúng vào chung một danh sách liên kết

158

## (1) Kỹ thuật Separate Chaining

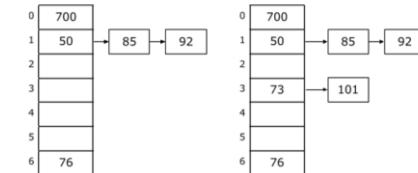
Giả sử ta có hàm băm chuyển đổi **các khóa 50, 700, 76, 85, 92, 73, 101** bằng cách **chia cho 7 rồi lấy số dư**.



159

## (1) Kỹ thuật Separate Chaining

Giả sử ta có hàm băm chuyển đổi **các khóa 50, 700, 76, 85, 92, 73, 101** bằng cách **chia cho 7 rồi lấy số dư**.



Chèn 92, xung  
đột xảy ra, chè  
phần tử mới và

Chèn 73 và 10)

160

## (1) Kỹ thuật Separate Chaining

#### **Ưu điểm:**

- **Cài đặt đơn giản**
  - Không phải lo tới kích thước bảng băm, và ta luôn có thể **thêm dữ liệu vào bảng bằng cách thêm vào các danh sách liên kết**.

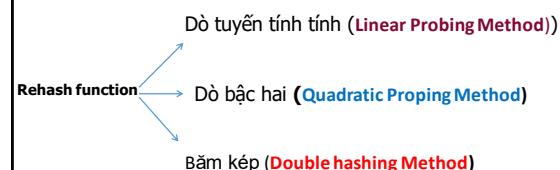
#### Nhược điểm:

- Phải sử dụng vùng nhớ ngoài.
  - Khi mà chuỗi (danh sách liên kết) trở nên quá dài, lúc đó thời gian cho các thao tác tìm kiếm, xóa phần tử có thể rất tốn thời gian.

161

## (2) Phương pháp (rehash function)

Nếu bấm lần đầu bị xung đột thì bấm lại lần 1, nếu bị xung đột nữa thì bấm lại lần 2,... Quá trình **bấm lại diễn ra cho đến khi không còn xung đột nữa**



162

### (2.1) Dò tuyến tính (Linear Probing Method)

- Nếu chưa bị xung đột thì thêm nút mới vào địa chỉ qua hàm băm.
- Nếu bị xung đột thì hàm băm lại lần một- f1 sẽ xét địa chỉ kế tiếp, nếu lại bị xung đột thì hàm băm thì hàm băm lại lần hai - f2 sẽ xét địa chỉ kế tiếp.
- Quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm nút vào địa chỉ này.
- Khi tìm một nút có khoá key vào bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1, tìm nút khoá key trong khôi đặc chứa các nút xuất phát từ địa chỉ i.

### (2.1) Dò tuyến tính (Linear Probing Method)

- Hàm băm lại của phương pháp dò tuyến tính là **truy xuất địa chỉ kế tiếp**. Hàm băm lại lần i được biểu diễn bằng công thức sau:  

$$f(key)=(f(key)+i) \% M$$
 với **f(key)** là **hàm băm chính** của bảng băm.
- Lưu ý** địa chỉ dò tìm kế tiếp là địa chỉ 0 nếu đã dò đến cuối bảng

### (2.1) Dò tuyến tính (Linear Probing Method)

• Được cài đặt bằng <b>danh sách kèm có M nút</b> .	0	nullkey
• Mỗi <b>nút của bảng băm</b> là một mẫu tin có <b>một trường key</b> để chứa khoá của nút.	1	nullkey
• Khi khởi tạo bảng băm thì <b>tất cả trường key được gán NULLKEY</b> .	2	nullkey
• Khi thêm nút có khoá key vào bảng băm, <b>hàm băm f(key) sẽ xác định địa chỉ i</b> trong khoảng từ 0 đến M-1:	3	nullkey
	...	nullkey
	M-1	nullkey

Thêm các nút 32, 53, 22, 92, 17, 34, 24, 37, 56 vào bảng băm bằng phương pháp dò tuyến tính

0	NULL	0	NULL	0	NULL	0	NULL
1	NULL	1	NULL	1	NULL	1	NULL
2	32	2	32	2	32	2	32
3	53	3	53	3	53	3	53
4	22	4	22	4	22	4	22
5	92	5	92	5	92	5	92
6	NULL	6	34	6	34	6	34
7	17	7	17	7	17	7	17
8	NULL	8	NULL	8	24	8	24
9	NULL	9	NULL	9	37	9	37

166

Thêm các nút 32, 53, 22, 92, 17, 34, 24, 37, 56 vào bảng băm bằng phương pháp dò tuyến tính

0	NULL	0	NULL	0	NULL	0	NULL
1	NULL	1	NULL	1	NULL	1	NULL
2	32	2	32	2	32	2	32
3	53	3	53	3	53	3	53
4	22	4	22	4	22	4	22
5	92	5	92	5	92	5	92
6	NULL	6	34	6	34	6	34
7	17	7	17	7	17	7	17
8	NULL	8	NULL	8	24	8	24
9	NULL	9	NULL	9	37	9	37

167

Thêm các nút 32, 53, 22, 92, 17, 34, 24, 37, 56 vào bảng băm bằng phương pháp dò tuyến tính

0	NULL	0	56
1	NULL	1	NULL
2	32	2	32
3	53	3	53
4	22	4	22
5	92	5	92
6	34	6	34
7	17	7	17
8	24	8	24
9	37	9	37

168

### (2.1) Nhận xét - Linear Probing Method

- Bảng băm này chỉ tối ưu khi băm đều, nghĩa là trên bảng băm các khối đặc chứa vài phần tử và các khối phần tử chưa sử dụng xen kẽ nhau, tốc độ truy xuất lúc này có bậc 0(1).
- Trường hợp xấu nhất là băm không đều hoặc bảng băm đầy, lúc này hình thành một khối đặc có n phần tử, nên **tốc độ truy xuất lúc này có bậc 0(n)**.

### (2.2) Dò bậc hai (Quadratic Probing Method)

- Nếu chưa bị xung đột thì thêm nút mới vào địa chỉ i.
- Nếu bị xung đột thì hàm băm lại lần 1 -  $f_1$  sẽ xét địa chỉ cách  $i^2$ , nếu lại bị xung đột thì hàm băm lại lần 2  $f_2$  sẽ xét địa chỉ cách  $i - 2^2$ , ...,
- Quá trình cứ thế cho đến khi nào tìm được trống và thêm nút vào địa chỉ này.
- Khi tìm một nút có khóa key trong bảng băm thì xét nút tại địa chỉ  $i = f_i(\text{key})$ , nếu chưa tìm thấy thì xét nút cách  $i = 1^2, 2^2, \dots$ , quá trình cứ thế cho đến khi tìm được khóa (trường hợp tìm thấy) hoặc rơi vào địa chỉ trống (trường hợp không tìm thấy).

### (2.2) Dò bậc hai (Quadratic Probing Method)

- Hàm băm lại của phương pháp dò bậc hai là truy xuất các địa chỉ cách bậc 2.
- Hàm băm lại hàm  $f_i$  được biểu diễn bằng công thức sau:  

$$f_i(\text{key}) = (f(\text{key}) + i^2) \% M$$

với  $f(\text{key})$  là hàm băm chính của bảng băm.
- Nếu đã dò đến cuối bảng thì trở về dò lại từ đầu bảng.
- Bảng băm với phương pháp dò bậc hai **nên chọn số địa chỉ M là số nguyên tố**.

### (2.2) Dò bậc hai (Quadratic Probing Method)

- Khắc phục phương pháp dò tuyến tính **rải các nút không đều** → bảng băm với phương pháp dò **bậc hai rải các nút đều hơn**.
- Bảng băm trong trường hợp này được **cài đặt bằng danh sách** kê cổ M nút, **mỗi nút của bảng băm là một mẩu tin có một trường key để chứa khóa các nút**
- Khi khởi tạo bảng băm thì **tất cả trường key bị gán NULLKEY**.
- Khi thêm nút có khóa key vào bảng băm, **hàm băm  $f(\text{key})$  sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1**.

### Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36

$$f_i(\text{key}) = (f(\text{key}) + i^2) \% M$$

0	10	10%10 = 0
1	NULL	
2	NULL	
3	NULL	
4	NULL	
5	15	
6	16	
7	NULL	
8	NULL	
9	NULL	

### Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36

$$f_i(\text{key}) = (f(\text{key}) + i^2) \% M$$

0	10	0	10
1	NULL	1	20
2	NULL	2	NULL
3	NULL	3	NULL
4	NULL	4	NULL
5	15	5	15
6	16	6	16
7	NULL	7	NULL
8	NULL	8	NULL
9	NULL	9	NULL

$$20\%10 = 0 \rightarrow \text{Đúng độ}$$

Băm lại lần 1 cách vị trí  
đúng độ 1 đơn vị  $\rightarrow 1$

### Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36

$$fi(key) = (f(key) + i2) \% M$$

0	10	0	10	0	10
1	NULL	1	20	1	20
2	NULL	2	NULL	2	NULL
3	NULL	3	NULL	3	NULL
4	NULL	4	NULL	4	30
5	15	5	15	5	15
6	16	6	16	6	16
7	NULL	7	NULL	7	NULL
8	NULL	8	NULL	8	NULL
9	NULL	9	NULL	9	25

$30 \% 10 = 0 \rightarrow \text{Đúng đắn}$

Băm lại lần 1 cách vị trí  
dụng độ 1 đơn vị  $\rightarrow 1 \rightarrow$   
**Đúng đắn**

Băm lại lần 2 cách vị trí  
dụng độ 4 đơn vị  $\rightarrow 4$

### Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36

$$fi(key) = (f(key) + i2) \% M$$

0	10	0	10	0	10
1	NULL	1	20	1	20
2	NULL	2	NULL	2	NULL
3	NULL	3	NULL	3	NULL
4	NULL	4	NULL	4	30
5	15	5	15	5	15
6	16	6	16	6	16
7	NULL	7	NULL	7	NULL
8	NULL	8	NULL	8	NULL
9	NULL	9	NULL	9	25

$30 \% 10 = 0 \rightarrow \text{Đúng đắn}$

Băm lại lần 1 cách vị trí  
dụng độ 1 đơn vị  $\rightarrow 1 \rightarrow$   
**Đúng đắn**

Băm lại lần 2 cách vị trí  
dụng độ 4 đơn vị  $\rightarrow 4$

### Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36

$$fi(key) = (f(key) + i2) \% M$$

0	10	0	10	0	10
1	NULL	1	20	1	20
2	NULL	2	NULL	2	NULL
3	NULL	3	NULL	3	NULL
4	NULL	4	NULL	4	30
5	15	5	15	5	15
6	16	6	16	6	16
7	NULL	7	NULL	7	26
8	NULL	8	NULL	8	NULL
9	NULL	9	25	9	25

### Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36

$$fi(key) = (f(key) + i2) \% M$$

0	10	0	10	0	10	0	10
1	NULL	1	20	1	20	1	20
2	NULL	2	NULL	2	NULL	2	36
3	NULL	3	NULL	3	NULL	3	NULL
4	NULL	4	NULL	4	30	4	30
5	15	5	15	5	15	5	15
6	16	6	16	6	16	6	16
7	NULL	7	NULL	7	26	7	26
8	NULL	8	NULL	8	NULL	8	NULL
9	NULL	9	NULL	9	25	9	25

### (2.2) Quadratic Probing Method – Nhận xét

- Nên chọn số địa chỉ  $M$  là số nguyên tố. Khi khởi động bảng băm thì tất cả  $M$  trường key được gán NULL, biến toàn cục  $N$  được gán 0.
- Bảng băm đầy khi  $N = M-1$ , và nên dành ít nhất một phần tử trống trên bảng băm.
- Bảng băm này tối ưu hơn bảng băm dùng phương pháp dò tuyển tính do rải rác phần tử đều hơn, nếu bảng băm chưa đầy thì tốc độ truy xuất có bậc 0(1). Trường hợp xấu nhất là bảng băm đầy vì lúc đó tốc độ truy xuất chậm do phải thực hiện nhiều lần so sánh.

### (2.2) Băm kép -Double hashing Method

- Bảng băm này **dùng hai hàm băm khác nhau với mục đích để rải rác đều các phần tử trên bảng băm**.
  - Chúng ta **có thể dùng hai hàm băm bất kì**, ví dụ chọn hai hàm băm như sau:
- $f1(key) = key \% M$ .
- $f2(key) = (M-2)-key \% (M-2)$ .

## (2.2) Băm kép -Double hashing Method

- Khi thêm phần tử có khoá key vào bảng băm, thì  $i=f1(key)$  và  $j=f2(key)$  sẽ xác định địa chỉ  $i$  và  $j$  trong khoảng từ 0 đến  $M-1$ :
- Nếu chưa bị xung đột thì thêm phần tử mới tại địa chỉ  $i$ .
- Nếu bị xung đột thì hàm băm lại lần 1  $f1$  sẽ xét địa chỉ mới  $i+j$ , nếu lại bị xung đột thì hàm băm lại lần 2 là  $f2$  sẽ xét địa chỉ  $i+2j$ , ..., quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm phần tử vào địa chỉ này.

## (2.2) Băm kép -Double hashing Method

- Bảng băm cài đặt bằng danh sách kề có  $M$  phần tử, mỗi phần tử của bảng băm là một mẩu tin có một trường key để lưu khoá các phần tử.
- Khởi tạo bảng băm: tất cả trường key được gán NULL.
- Khi tìm kiếm một phần tử có khoá key trong bảng băm, hàm băm  $i=f1(key)$  và  $j=f2(key)$  sẽ xác định địa chỉ  $i$  và  $j$  trong khoảng từ 0 đến  $M-1$ , ..., quá trình cứ thế cho đến khi nào tìm được Xét phần tử tại địa chỉ  $i$ , nếu chưa tìm thấy thì xét tiếp phần tử  $i+j+2j$  (trường hợp tìm thấy) hoặc bị rơi vào địa chỉ trống (trường hợp không tìm thấy).

Thêm vào các khóa 10, 15, 16, 20, 30, 25, 26, 36 :

$$f1(key) = \text{key \%} M.$$

$$f2(key) = (M-2) - (\text{key \%} (M-2)).$$

0	10	0	10	0	10	0	10	0	10
1	NULL	1	NULL	1	NULL	1	20	1	20
2	NULL	2	NULL	2	30	2	30	2	26
3	NULL	3	NULL	3	NULL	3	NULL	3	36
4	NULL	4	20	4	NULL	4	NULL	4	20
5	15	5	15	5	15	5	15	5	15
6	16	6	16	6	16	6	16	6	16
7	NULL								
8	NULL	8	NULL	8	26	8	NULL	8	NULL
9	NULL	9	NULL	9	25	9	25	9	25

## Đề Minh Họa – Bảng băm

### Câu 3: (2 điểm)

Cho bảng băm A kích thước 13 phần tử và tập khóa  $K = \{10, 26, 52, 76, 13, 8, 3, 33, 60, 42\}$ , ta cần nạp các giá trị khóa K vào bảng băm A sử dụng hàm băm  $H(K) = K \% 13$ . Hãy vẽ bảng băm khi **thêm từng khóa K vào bảng A**, trong trường hợp xảy ra đụng độ, sử dụng phương pháp dò tuyển tính để giải quyết đụng độ.

## Đề Minh Họa – Bảng băm

### Câu 4 : (2 điểm)

Cho tập khóa  $K = \{12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5\}$  và hàm băm :

$$H(key) = (2 * key + 5) \% 11 \quad (\text{key: khoá cần băm}).$$

Anh / chị hãy thực hiện :

- Vẽ hình tóm lược việc lưu trữ từng khóa trong K vào bảng băm có kích thước  $M=11$ , dùng hàm băm H và phương pháp nối kết trực tiếp để xử lý xung đột. (1 điểm)
- Định nghĩa cấu trúc dữ liệu cho bảng băm ở câu a (1 điểm)

## Đề Minh Họa – Bảng băm

Câu 4 (1 điểm) : Giả sử cho bảng băm A kích thước 7 ô và tập khóa  $K = \{76, 93, 40, 47, 10, 55\}$ , ta cần nạp các giá trị khóa K vào bảng A sử dụng hàm băm  $H1(k) = k \% 7$ . Hãy vẽ bảng băm kết quả, trong trường hợp xảy ra đụng độ, hãy sử dụng phương pháp băm kép để xử lý, với hàm băm thứ 2 do anh / chị tự đề nghị.

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

### Câu 1 (4 điểm):

- a. Anh/ chị hãy viết hàm cài đặt thuật toán tìm kiếm giá trị x bằng phương pháp **tim nhị phân** trên mảng số nguyên có thứ tự tăng dần bằng ngôn ngữ C/C++ (2 điểm).

b. Trình bày các bước (vẽ từng bước) theo hàm đã cài đặt ở câu a để thực hiện việc tìm kiếm giá trị  $x = 10$  trên mảng số nguyên có giá trị như sau: **6 11 27 34 41 53** (2 điểm).

### Câu 2 (6 điểm):

- a. Anh/ chị hãy trình bày ý tưởng thuật toán nối bọt (Bubble Sort) để sắp xếp một mảng một chiều các số nguyên theo thứ tự tăng dần (2 điểm).

b. Viết hàm cài đặt thuật toán trong câu 2a bằng ngôn ngữ C/C++ (2 điểm).

- c. Trình bày các bước (vẽ từng bước) thực hiện việc sắp xếp theo hàm đã cài đặt ở câu 2b với mảng các số nguyên có giá trị như sau: **14 23 35 11 41 8** (2 điểm).

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

### Câu 2 (5 điểm)

Người ta muốn sử dụng danh sách liên kết đơn để quản lý danh sách các **phân số**, biết rằng mỗi phân số X đều có hai thành phần tử số (TS) và mẫu số (MS). Anh/Chị hãy thực hiện các yêu cầu sau đây:

- a. Định nghĩa các cấu trúc cần thiết để lưu danh sách theo mô tả trên (2 điểm).

b. Viết các hàm cần thiết để nhập một danh sách gồm N phân số từ bàn phím, biết rằng khi nhập lần lượt từng phân số sẽ thêm vào đầu danh sách (2 điểm).

- c. Viết hàm tính tổng các phân số có trong danh sách (1 điểm).

### Câu 3 (2 điểm)

Dựa vào các kiến thức đã học Anh/Chị hãy so sánh hai loại cấu trúc dữ liệu sau đây: mảng và danh sách liên kết.

188

## Đề Minh Họa – TÌM KIẾM SẮP XẾP

### Câu 1 (2 điểm):

- a. Anh/ Chị hãy trình bày ý tưởng của thuật toán tìm kiếm **nhi phân** (*Binary Search*) để tìm kiếm một số trên một mảng đã có thứ tự.

b. Trình bày các bước (vẽ từng bước) theo thuật toán tìm kiếm nhị phân thực hiện tìm kiếm giá trị  $x = 8$  trên mảng số nguyên có giá trị: **1 4 6 13 17 19**.

### Câu 2 (2.5 điểm):

- a. Anh/ chị hãy viết hàm cài đặt thuật toán **chọn trực tiếp** (*Selection Sort*) để sắp xếp một mảng các số nguyên có N phần tử theo chiều giảm dần bằng ngôn ngữ C/C++, cho biết độ phức tạp của giải thuật.

- b. Trình bày các bước (vẽ từng bước) áp dụng thuật toán ở câu 2.a để sắp xếp mảng số nguyên **{2, 7, 10, 9, 5, 3, 8, 40}** giảm dần.

189

## THANKS FOR LISTENING



## ANY QUESTIONS?

## BEST OF LUCK IN YOUR EXAMS



"Just stay calm. Try not to lose your head!"

190

## THANK YOU FOR LISTENING!

