



# Mảng và các thuật toán tìm kiếm, sắp xếp



# Nội dung

- Cấu trúc mảng
- Tìm kiếm tuyến tính, tìm kiếm nhị phân
- Các thuật toán sắp xếp đơn giản
- Mảng động, mảng nhiều chiều

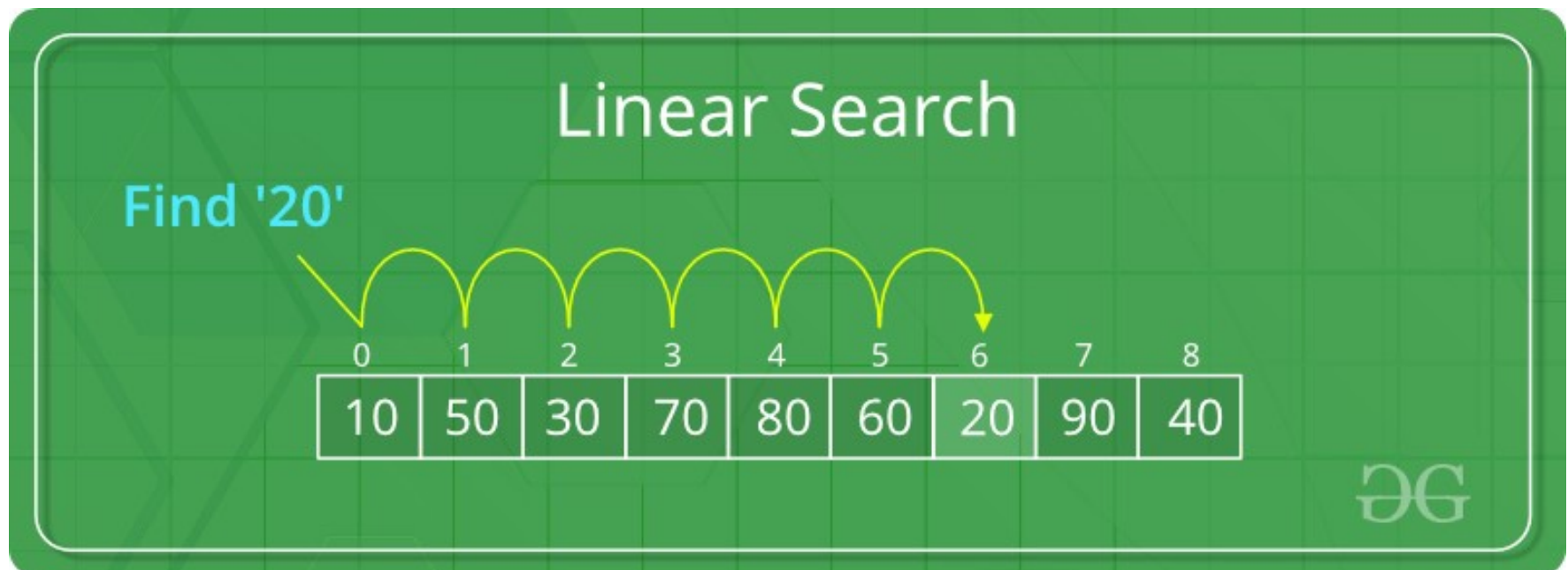
# Mảng

- Tập các phần tử cùng kiểu được lưu trữ liên tục trong bộ nhớ
- Truy cập ngẫu nhiên đến phần tử bất kỳ

Memory Location									
200	201	202	203	204	205	206	▪	▪	▪
U	B	F	D	A	E	C	▪	▪	▪
0	1	2	3	4	5	6	▪	▪	▪
Index									

# Tìm kiếm phần tử trên mảng

- Tìm một giá trị có nằm trong mảng cho trước hay không?



# Tìm kiếm

- Tìm kiếm tuyến tính tốc độ không cao
  - Số phép toán cần thiết = ???
- Cần thuật toán tìm kiếm hiệu quả hơn
- Giả sử mảng đã được sắp xếp, vậy có cách nào tìm nhanh hơn không?

# Tìm kiếm nhị phân

- Nếu khóa tìm kiếm = phần tử giữa mảng: **tìm thấy, kết thúc**. Nếu không,
- Xác định phần **nửa mảng** có khả năng chứa khóa tìm kiếm
- Tìm khóa trên nửa mảng bằng cách **lặp lại** các bước trên
- **Dừng** khi “mảng” còn 1 phần tử

# Binary Search

Search 23

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

$23 > 16$   
take 2<sup>nd</sup> half

L=0	1	2	3	M=4	5	6	7	8	H=9
2	5	8	12	16	23	38	56	72	91

$23 > 56$   
take 1<sup>st</sup> half

0	1	2	3	4	L=5	6	M=7	8	H=9
2	5	8	12	16	23	38	56	72	91


Found 23,  
Return 5

0	1	2	3	4	L=5, M=5	H=6	7	8	9
2	5	8	12	16	23	38	56	72	91

# Tìm kiếm nhị phân


- Áp dụng cho mảng đã sắp xếp
- Tốc độ cao
- Số phép toán (số lần chia) = ???





# Các thuật toán sắp xếp đơn giản

- Sắp xếp chọn (selection sort)
- Sắp xếp nổi bọt (bubble sort)
- Sắp xếp chèn (insertion sort)



# Sắp xếp chọn (selection sort)

- Duyệt mảng, tìm phần tử bé nhất (hoặc lớn nhất)
- Đổi chỗ với phần tử đầu tiên của mảng
- Lặp lại quá trình này với phần còn lại của mảng

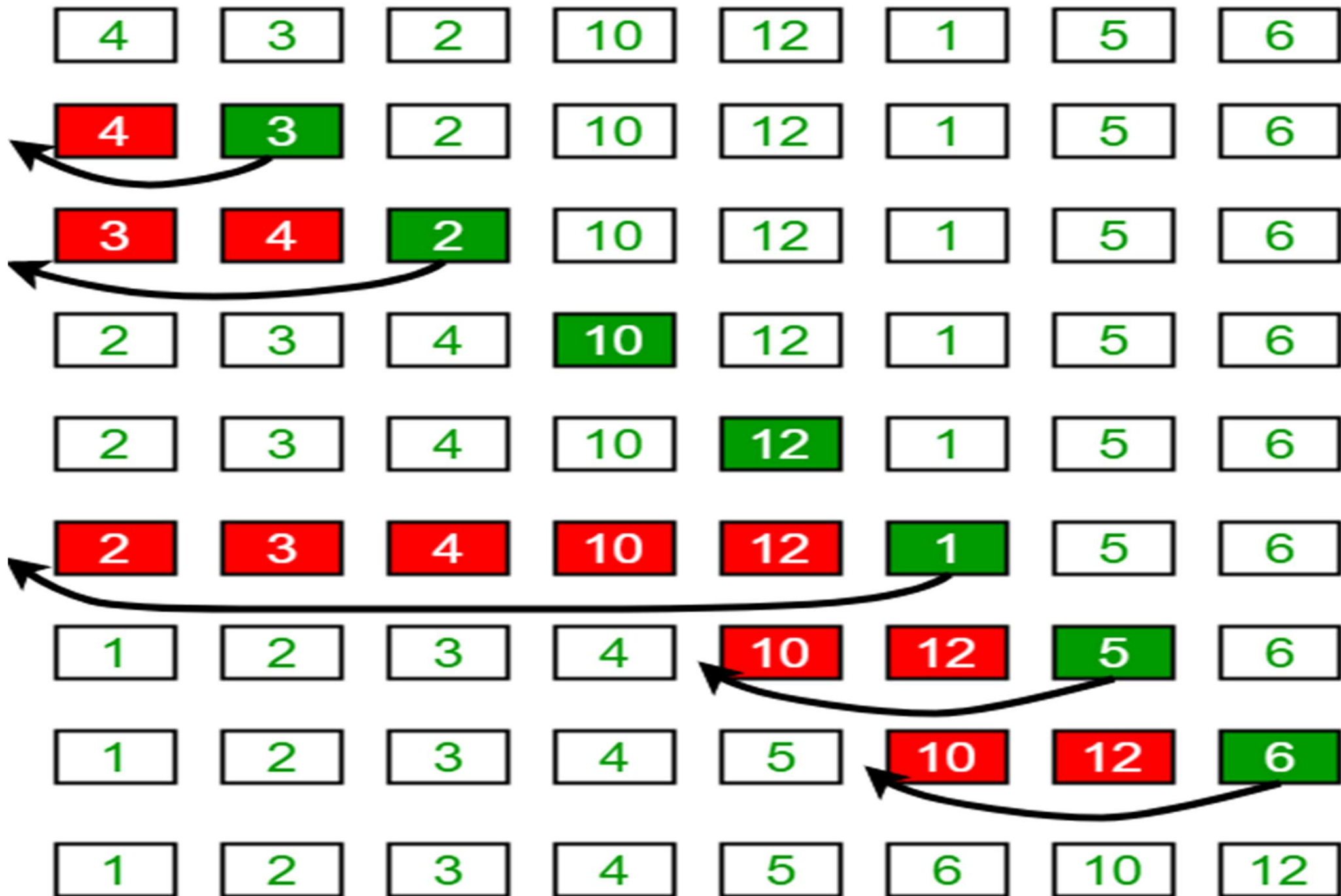
# Selection Sort



# Sắp xếp chèn

- Duyệt từ phần tử đầu tiên
- Với phần tử kế tiếp, chèn nó vào vị trí thích hợp và lùi các phần tử sau đó 1 vị trí
- Lặp lại cho đến khi kết thúc

## Insertion Sort Execution Example

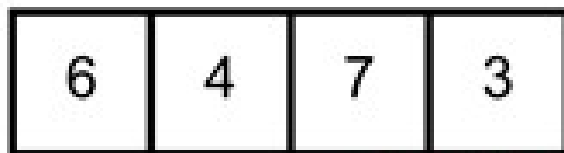
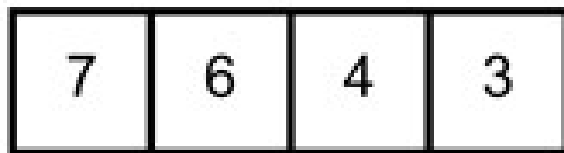




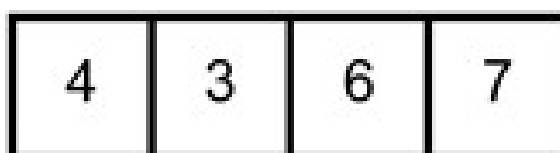
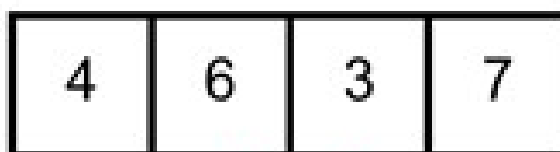
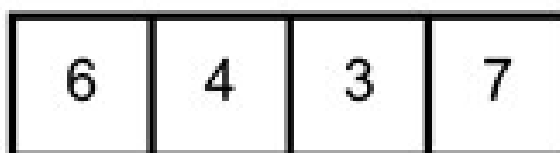
# Sắp xếp nổi bọt (bubble sort)

- Duyệt từ vị trí đầu đến cuối, so sánh 2 số cạnh nhau, nếu thứ tự ngược thì đổi chỗ
- Như vậy phần tử lớn nhất sẽ “nổi lên” vị trí cuối cùng
- Lặp lại các bước trên với mảng còn lại

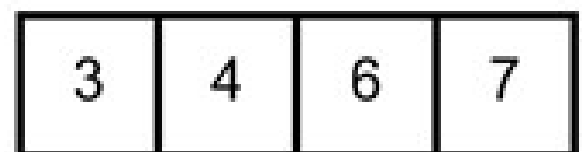
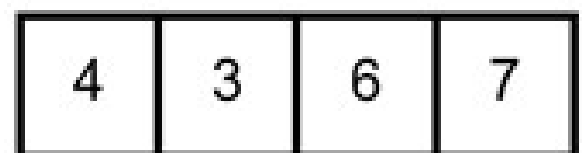
### First pass



### Second pass



### Third pass



# Độ phức tạp của thuật toán

- Độ phức tạp

- ☐ Số phép toán thực hiện
- ☐ Kích thước bộ nhớ

- Độ phức tạp của các thuật toán sắp xếp?

- ☐ Chọn
- ☐ Chèn
- ☐ Nổi bọt



# Mảng động

- Khai báo động số phần tử khi sử dụng
- Thuận tiện khi muốn dùng mảng nhưng chưa biết số phần tử thực tế
- Cần giải phóng bộ nhớ khi chấm dứt sử dụng

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

# Mảng nhiều chiều

- Có nhiều hơn một chỉ số, mảng của các mảng (ví dụ: ma trận)
- Có thể kết hợp với mảng động

```
const int Size = 9;  
int m[Size][Size];  
for (int i=0; i<Size; i++)  
    for (int j=0; j<Size; j++)  
        m[i][j] = (i+1) * (j+1);
```

# Mảng nhiều chiều

```
const int Size = 9;
int* m[Size];
for (int i=0; i<Size; i++) {
    m[i] = new int[i+1];
    for (int j=0; j<i+1; j++)
        m[i][j] = (i+1) * (j+1);
}
```

- Chú ý, vùng dữ liệu của toàn bộ “mảng” sẽ không liên tục trong bộ nhớ

# Tham số mảng

- Chương trình dịch cần biết thông tin về kích thước mảng để tính địa chỉ phần tử mảng

```
const int M = 8;  
const int N = 8;  
  
void print(int arr[M][N])  
{  
    int i, j;  
    for (i = 0; i < M; i++)  
        for (j = 0; j < N; j++)  
            cout << a[i][j] << " ";  
}
```

# Tham số mảng

- Có thể tham số hóa chiều thứ nhất

□ Tại sao?

```
const int N = 8;

void print(int arr[][N], int m)
{
    int i, j;
    for (i = 0; i < m; i++)
        for (j = 0; j < N; j++)
            cout << a[i][j] << " ";
}
```

# Bài tập/thực hành

- Cài đặt thuật toán tìm kiếm nhị phân
  - Nghĩ các ứng dụng khác của thuật toán tìm kiếm nhị phân
- Cài đặt các thuật toán sắp xếp chọn, chèn, nổi bọt
- Viết chương trình dùng mảng động, mảng nhiều chiều
- Tạo lớp mảng động; lớp mảng nhiều chiều

# Bài tập: Lớp mảng động Array

- Tạo lớp mảng động Array cho số nguyên với các chức năng và giao diện sau:
  - Khởi tạo với đối số là số phần tử của mảng
  - Hàm hủy giải phóng bộ nhớ động đã cấp cho mảng
  - Hàm thành viên `at(int i)` truy vấn phần tử thứ `i` của mảng
  - Hàm thành viên `size()` trả lại kích thước của mảng
- Nâng cao
  - Cải tiến Array thành template cho kiểu dữ liệu bất kỳ
  - Cài đặt toán tử truy vấn `[ ]`

# Ôn tập kiến thức C++

- Ôn tập về con trỏ
- Ôn tập về đệ qui
  - Cài đặt tìm kiếm nhị phân bằng đệ qui
  - Bài toán Tháp Hà Nội (Tower of Hanoi)





# Chuẩn bị bài

- Tìm hiểu về độ phức tạp thuật toán
- Tìm hiểu các thuật toán sắp xếp nhanh (quick sort), sắp xếp trộn (merge sort)