

# Bài 12: Bubble Sort, Binary Search Binary Search Tree

Nguyễn Hoàng Anh

# Bubble Sort

- Thuật toán **sắp xếp nổi bọt** (Bubble Sort) hoạt động dựa trên nguyên tắc hoán đổi các phần tử liền kề để đưa phần tử lớn hơn về cuối dãy (hoặc phần tử nhỏ hơn về đầu dãy).
- Thuật toán gồm các bước sau:
  - 1. Duyệt qua danh sách từ đầu đến cuối.
  - 2. So sánh hai phần tử liền kề, nếu phần tử trước lớn hơn phần tử sau, thì hoán đổi vị trí.
  - 3. Lặp lại quá trình cho đến khi không còn sự hoán đổi nào xảy ra (mảng đã được sắp xếp).

# Bubble Sort

```
int arr[] = {5, 3, 8, 6, 2, -3}
```

0	1	2	3	4	5
5	3	8	6	2	-3

# Bubble Sort

Loop 1:

Input

5	3	8	6	2	-3
---	---	---	---	---	----

J

J+1

Output

3	5	8	6	2	-3
---	---	---	---	---	----

# Bubble Sort

Loop 1:

Input

3	5	8	6	2	-3
---	---	---	---	---	----

J

J+1

Output

3	5	8	6	2	-3
---	---	---	---	---	----

# Bubble Sort

Luợt 1:

Input

3	5	8	6	2	-3
---	---	---	---	---	----

J

J+1

Output

3	5	6	8	2	-3
---	---	---	---	---	----

# Bubble Sort

Loop 1:

Input

3	5	6	8	2	-3
			J	J+1	

Output

3	5	6	2	8	-3
---	---	---	---	---	----

# Bubble Sort

## Kết thúc Lượt 1:

## Input

3	5	6	2	8	-3
				J	J+1

## Output

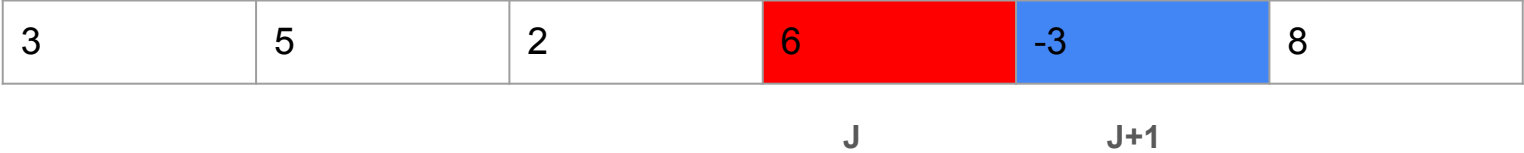
3	5	6	2	-3	8
---	---	---	---	----	---



# Bubble Sort

Kết thúc Lượt 2:

Input



Output



# Bubble Sort

Kết thúc Lượt 3:

Input



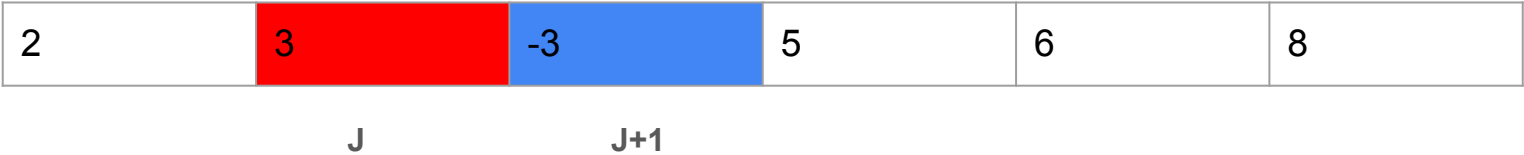
Output



# Bubble Sort

Kết thúc Lượt 4:

Input



Output



# Bubble Sort

Kết thúc Lượt 5:

Input



Output



# Bubble Sort

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

# Linear Search

- Thuật toán **tìm kiếm tuyến tính** (Linear Search) là phương pháp đơn giản nhất để tìm kiếm một phần tử trong mảng.
- Nguyên tắc hoạt động:
  - 1. Duyệt từng phần tử trong mảng từ trái sang phải.
  - 2. Nếu phần tử đang xét trùng với giá trị cần tìm, trả về vị trí của nó.
  - 3. Nếu duyệt hết mảng mà không tìm thấy, trả về kết quả không tồn tại.

# Linear Search

```
int arr[] = {13, 11, 15, 30, 18, 16, 21, 25, 20}
```

0	1	2	3	4	5	6	7	8
13	11	15	30	30	16	21	25	20

ID : 25

# Binary Search

- Thuật toán **tìm kiếm nhị phân** (Binary Search) hoạt động bằng cách **chia đôi mảng** để tìm kiếm, thay vì duyệt tuần tự như Linear Search.
- Nguyên tắc hoạt động:
  - 1. Sắp xếp mảng (tăng dần hoặc giảm dần).
  - 2. So sánh phần tử ở giữa mảng với giá trị cần tìm:
    - Nếu trùng → Trả về vị trí.
    - Nếu nhỏ hơn → Tiếp tục tìm trong nửa phải.
    - Nếu lớn hơn → Tiếp tục tìm trong nửa trái.
  - 3. Lặp lại bước 2 cho đến khi tìm thấy phần tử hoặc không còn phần tử nào để tìm.



# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3	4	5	6	7	8
11	13	15	16	18	20	21	30	30

ID : 27

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3	4	5	6	7	8
11	13	15	16	18	20	21	25	30



left



right

ID : 27

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3	4	5	6	7	8
11	13	13	16	18	20	21	25	30

↑  
left

↑  
mid

↑  
right

ID : 27



$$\text{mid} = (\text{left} + \text{right}) / 2 = (0 + 8) / 2 = 4$$

ID == arr[mid] ?

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3	4	5	6	7	8
11	13	15	16	18	20	21	25	30

 left                       right

ID : 27

$\text{left} = \text{mid} + 1 = 5$

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

5	6	7	8
20	21	25	30

↑ left                      ↑ right

ID : 27

$\text{left} = \text{mid} + 1 = 5$

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

5	6	7	8
20	21	25	30
↑ left	↑ mid		↑ right

ID : 27

$\text{mid} = (\text{left} + \text{right}) / 2 = (5 + 8) / 2 = 6$

$\text{ID} == \text{arr}[\text{mid}] ?$

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

7	8
25	30
↑ left	↑ right

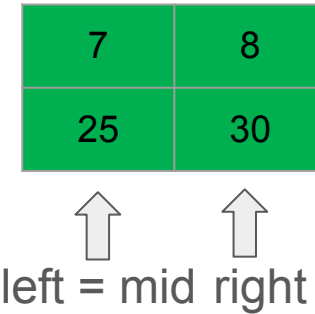
ID : 27

$\text{left} = \text{mid} + 1 = 7$

$\text{ID} == \text{arr}[\text{mid}] ?$

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```



ID : 27

$\text{mid} = (\text{left} + \text{right}) / 2 = (7 + 8) / 2 = 7$

$\text{ID} == \text{arr}[\text{mid}] ?$



# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

8
30



left = right

ID : 27

left = mid + 1 = 8 = right

ID == arr[mid] ?

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```



left = right = mid

ID : 27

$\text{mid} = (\text{left} + \text{right}) / 2 = (8 + 8) / 2 = 8$

ID == arr[mid] ?

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

5	6	7	8
20	21	25	30



right



left

ID : 27

$\text{right} = \text{mid} - 1$

$\text{right} < \text{left} \rightarrow \text{Kết thúc}$

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3	4	5	6	7	8
11	13	15	16	18	20	21	25	30

↑  
left

↑  
mid

↑  
right

ID : 14

$$\text{mid} = (\text{left} + \text{right}) / 2 = (0 + 8) / 2 = 4$$

ID == arr[mid] ?

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3	4	5	6	7	8
11	13	15	16	18	20	21	25	30

↑  
left

↑  
mid

↑  
right

ID : 14

```
right = mid - 1
```

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3
11	13	15	16



left



right

ID : 14

right = mid - 1

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3
11	13	15	16

↑      ↑                      ↑  
left   mid                      right

ID : 14

$\text{mid} = (\text{left} + \text{right}) / 2 = 1.5$

$\text{ID} == \text{arr}[\text{mid}] ?$

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3
11	13	15	16



left



right

ID : 14

left = mid + 1



# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3
11	13	15	16

↑      ↑  
left = mid   right

ID : 14

$\text{mid} = (\text{left} + \text{right}) / 2 = 2$

$\text{ID} == \text{arr}[\text{mid}] ?$

# Binary Search

```
int arr[] = { 11, 13, 15, 16, 18, 20, 21, 25, 30}
```

0	1	2	3
11	13	15	16



right



left

ID : 14

right = mid - 1

right < left

# Binary Search

```
int arr[10000] = { 13, 11, 15,..., 21, 25, 20}
```

0	1	2	...	9997	9998	9999
13	11	15	...	21	25	20

ID : 27

# Binary Search

```
int arr[10000] = { 11, 12, 13, ..., 999, 1000}
```

0	1	2	...	9998	9999
11	12	13	...	999	1000

ID : 800

- 10000 / 2 = 5000

5000 / 2 = 2500

2500 / 2 = 1250

1250 / 2 = 625

625 / 2 = 312

312 / 2 = 156

156 / 2 = 78
- 78 / 2 = 39

39 / 2 = 20

20 / 2 = 10

10 / 2 = 5

5 / 2 = 2

2 / 2 = 1

1 / 2 = 0

# Binary Search

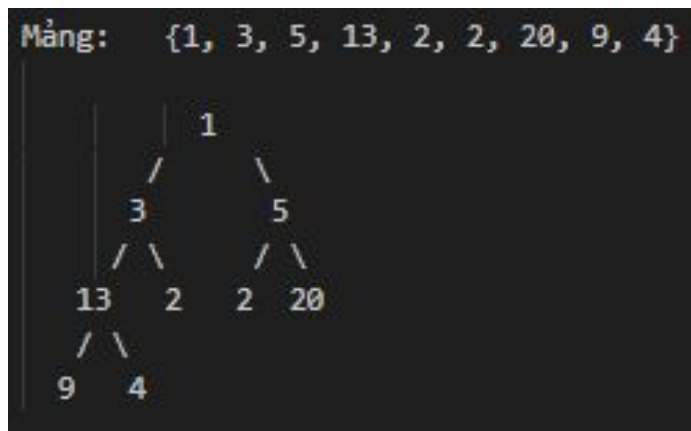
```
#define NO_FOUND -1

int binarySearch(int* arr, int l, int r, int x)
{
    if (r >= l)
    {
        int mid = (r + l) / 2;

        if (arr[mid] == x){
            return mid;
        }
        else if (arr[mid] > x){
            return binarySearch(arr, l, mid - 1, x);
        }
        else{
            return binarySearch(arr, mid + 1, r, x);
        }
    }
    return NO_FOUND;
}
```

# Binary Search Tree

- **Cấu trúc dữ liệu phân cấp** (Tree) là một cấu trúc dữ liệu phi tuyến tính, trong đó các phần tử (được gọi là nút, hay node) được tổ chức theo một **thứ bậc phân cấp**. Cây là một trong những cấu trúc dữ liệu quan trọng, được sử dụng rộng rãi trong khoa học máy tính để biểu diễn các quan hệ phân cấp, tìm kiếm, sắp xếp, và lưu trữ.



# Binary Search Tree

- **Cây Tìm Kiếm Nhị Phân** (BST - Binary Search Tree) là một cấu trúc dữ liệu dạng cây, trong đó:
  - Mỗi nút có tối đa 2 con (gọi là cây con trái và cây con phải).
  - Dữ liệu trong cây tuân theo quy tắc:
    - Nút con trái chứa giá trị nhỏ hơn nút gốc.
    - Nút con phải chứa giá trị lớn hơn nút gốc.
    - Quy tắc này áp dụng đệ quy cho toàn bộ cây.

# Binary Search Tree

```
int arr[] = {1, 3, 5, 13, 2, 2, 20, 9, 4}
```

0	1	2	3	4	5	6	7	8
1	3	5	13	2	2	20	9	4
1	2	2	3	4	5	9	13	20



# Binary Search Tree

```

/*****
 * @file    Binary_Search_Tree.c
 * @brief    Chuyển đổi danh sách liên kết thành cây nhị phân và tìm kiếm nhị phân.
 * @details  Chương trình xây dựng danh sách liên kết đơn, sau đó chuyển đổi danh sách
 *           thành cây nhị phân tìm kiếm (BST). Hỗ trợ thêm node, tìm điểm giữa danh sách,
 *           và thực hiện tìm kiếm nhị phân.
 * @version  1.0
 * @date     2024-11-12
 * @author   Anh Nguyen
 *****/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/**
 * @struct   Node
 * @brief    Cấu trúc của một node trong danh sách liên kết đơn.
 */
typedef struct Node
{
    int data;           /**< Giá trị của node */
    struct Node *next;  /**< Con trỏ đến node tiếp theo */
} Node;
```

# File operations

- Ngôn ngữ lập trình C cung cấp một số thư viện và hàm tiêu biểu để thực hiện các thao tác với file.
- File CSV (Comma-Separated Values) là một loại file văn bản được sử dụng để lưu trữ và truyền tải dữ liệu có cấu trúc dưới dạng bảng, trong đó các dữ liệu của các cột được phân tách bằng dấu phẩy (,) hoặc một ký tự ngăn cách khác.

# File operations

Giả sử bạn có một bảng thông tin về nhân viên với các cột sau:

- Họ và tên
- Tuổi
- Địa chỉ
- Số điện thoại

file.csv

Họ và tên	Tuổi	Địa chỉ	Số điện thoại
John Doe	30	123 Main St	555-1234
Jane Smith	25	456 Oak St	555-5678
Bob Johnson	40	789 Pine St	555-8765

# File operations

- Để mở một file, bạn có thể sử dụng hàm `fopen()`. Hàm này trả về một con trỏ `FILE`, và cần được kiểm tra để đảm bảo file đã mở thành công.

```
FILE *file = fopen(const char *file_name, const char *access_mode);
```

# File operations

Chế độ	Mô tả
r	Mở file với chế độ chỉ đọc file. Nếu mở file thành công thì trả về địa chỉ của phần tử đầu tiên trong file, nếu không thì trả về NULL.
rb	Mở file với chế độ chỉ đọc file theo định dạng binary. Nếu mở file thành công thì trả về địa chỉ của phần tử đầu tiên trong file, nếu không thì trả về NULL.
w	Mở file với chế độ ghi vào file. Nếu file đã tồn tại, thì sẽ ghi đè vào nội dung bên trong file. Nếu file chưa tồn tại thì sẽ tạo một file mới. Nếu không mở được file thì trả về NULL.
wb	Mở file với chế độ ghi vào file theo định dạng binary. Nếu file đã tồn tại, thì sẽ ghi đè vào nội dung bên trong file. Nếu file chưa tồn tại thì sẽ tạo một file mới. Nếu không mở được file thì trả về NULL.

# File operations

Chế độ	Mô tả
a	Mở file với chế độ nối. Nếu mở file thành công thì trả về địa chỉ của phần tử cuối cùng trong file. Nếu file chưa tồn tại thì sẽ tạo một file mới. Nếu không mở được file thì trả về NULL.
ab	Mở file với chế độ nối dưới định dạng binary. Nếu mở file thành công thì trả về địa chỉ của phần tử cuối cùng trong file. Nếu file chưa tồn tại thì sẽ tạo một file mới. Nếu không mở được file thì trả về NULL.
r+	Mở file với chế độ đọc và ghi file. Nếu mở file thành công thì trả về địa chỉ của phần tử đầu tiên trong file, nếu không thì trả về NULL.
rb+	Mở file với chế độ đọc và ghi file dưới định dạng binary. Nếu mở file thành công thì trả về địa chỉ của phần tử đầu tiên trong file, nếu không thì trả về NULL.

# File operations

Chế độ	Mô tả
w+	Mở file với chế độ ghi và đọc file. Nếu file đã tồn tại thì trả về địa chỉ của phần tử đầu tiên của file. Nếu file chưa tồn tại thì sẽ tạo một file mới.
wb+	Mở file với chế độ ghi và đọc file dưới định dạng binary. Nếu file đã tồn tại thì trả về địa chỉ của phần tử đầu tiên của file. Nếu file chưa tồn tại thì sẽ tạo một file mới.
a+	Mở file với chế độ nối và đọc file. Nếu file đã tồn tại thì trả về địa chỉ của phần tử cuối cùng của file. Nếu file chưa tồn tại thì sẽ tạo một file mới.
ab+	Mở file với chế độ nối và đọc file dưới định dạng binary. Nếu file đã tồn tại thì trả về địa chỉ của phần tử cuối cùng của file. Nếu file chưa tồn tại thì sẽ tạo một file mới.

# File operations

## Đọc File

Tên hàm	Mô tả
fscanf()	Sử dụng chuỗi được định dạng và danh sách đối số biến để lấy đầu vào từ một File
fgets()	Copy nội dung trong File vào mảng dùng để lưu trữ với tối đa số lượng phần tử của mảng hoặc tới khi gặp ký tự xuống dòng.
fgetc()	Lấy giá trị tại địa chỉ hiện tại của file, sau đó di chuyển tới địa chỉ tiếp theo. Kiểu trả về là char
fread()	Đọc một số lượng byte được chỉ định từ File .bin



# File operations

## Ghi File

Tên hàm	Mô tả
fprintf()	Ghi chuỗi vào File, và có thể thêm danh sách các đối số
fputs()	Ghi chuỗi vào File
fputc()	Ghi một ký tự vào File
fwrite()	Ghi một số byte được chỉ định vào File .bin

# File operations

Một số hàm khác:

- `fclose()`: Đóng File đã mở
- `feof()`: Để kiểm tra địa chỉ hiện tại có phải ký tự cuối cùng của File hay chưa

# Makefile

[https://docs.google.com/document/d/1orn\\_c-s46cFZCu2LZfVu9ztrV-fL6kTA/edit?usp=sharing&ouid=112023306400142991703&rtpof=true&sd=true](https://docs.google.com/document/d/1orn_c-s46cFZCu2LZfVu9ztrV-fL6kTA/edit?usp=sharing&ouid=112023306400142991703&rtpof=true&sd=true)

# Quy Tắc Code Trong AUTOSAR Classic

<https://docs.google.com/document/d/18XAQT0KdeCmdF8uYe8dmj9tAPqMBjpXr/edit?usp=sharing&oid=112023306400142991703&rtpof=true&sd=true>