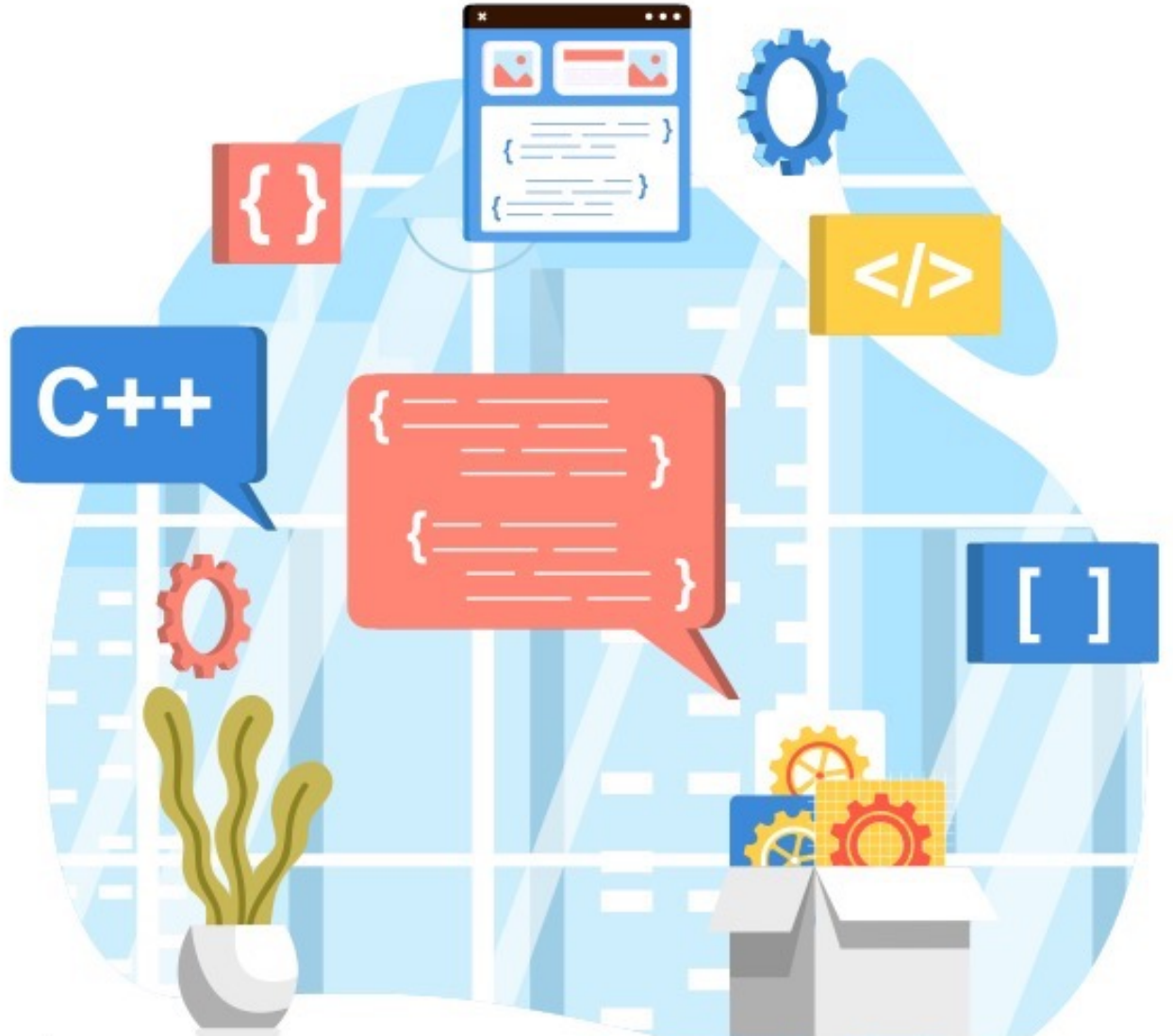
# CÁC GIẢI THUẬT TÌM KIẾM

VŨ NGỌC THANH SANG
KHOA CÔNG NGHỆ THÔNG TIN
ĐẠI HỌC SÀI GÒN



# I – Introduction

Sử dụng thuật toán tìm kiếm:

- Xác định xem một phần tử cụ thể có trong mảng/danh sách hay không
- Nếu dữ liệu đã được sắp xếp, tìm vị trí trong mảng/danh sách để chèn một phần tử mới.
- Tìm vị trí của để xóa một phần tử.

# II – Linear Search

Tìm kiếm tuyến tính (linear search) – tìm kiếm tuần tự (sequential search)

- Duyệt từ phẩn tử một, khi phần tử cần tìm được tìm thấy, trả về chỉ mục của phần tử đó.
- Giải thuật có thể được triển khai bằng cách sử dụng phương pháp lặp (iteration) hoặc đệ quy (recursion)

# II - Linear Search

# II - Linear Search

```
int seqSearch(const int array[], int length,
              const int& item){
    int loc;
    bool found = false;
    loc = 0;
    while (loc < length && !found)
        if (array[loc] == item)
            found = true;
        else
            loc++;
    if (found)
        return loc;
    else
        return -1;
} //end seqSearch
```

# II – Linear Search

Với mảng có độ dài n

Nếu phần tử được tìm kiếm (đích) không có trong mảng/danh sách:

→ n phép so sánh

Nếu phần tử được tìm kiếm có trong mảng/danh sách:

- Là phần tử đầu tiên → 1 phép so sánh (**trường hợp tốt nhất**)
- Là phần tử cuối cùng → n (**trường hợp xấu nhất**)
- Số lần so sánh trong **trường hợp trung bình**:

$$\frac{1 + 2 + \cdots + n}{n} = \frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2}$$

# III – Binary Search

Tìm kiếm nhị phân (binary search)

- Dựa trên nguyên lý chia để trị
- **Yêu cầu dữ liệu đã được sắp xếp theo thứ tự [tăng dần].**
- So sánh phần tử được tìm kiếm với phần tử ở giữa mảng/danh sách
- Nếu phần tử được tìm kiếm nhỏ hơn phần tử ở giữa, tiếp tục tìm kiếm với mảng con bên trái.
- Nếu phần tử được tìm kiếm lớn hơn phần tử ở giữa, tiếp tục tìm kiếm với mảng con bên phải.

# III – Binary Search

# III – Binary Search

```
int binarySearch(const int array[], int length,
                 const int& item){
    int first = 0;
    int last = length - 1;
    int mid;
    bool found = false;
    while (first <= last && !found){
        mid = (first + last) / 2;
        if (array[mid] == item)
            found = true;
        else if (array[mid] > item)
            last = mid - 1;
        else
            first = mid + 1;
    }
    if (found)
        return mid;
    else
        return -1;
} //end binarySearch
```

# III – Binary Search

Mỗi lần lặp làm giảm kích thước của mảng/danh sách tìm kiếm đi một nửa

Giả sử danh sách có N = 1024 = $2^{10}$ phần tử

→ Cần tối đa 11 lần lặp để tìm x

Mỗi lần lặp lại tạo ra hai phép so sánh chính

→ Trong trường hợp này, nhiều nhất là 22 phép so sánh chính

**Trường hợp tốt nhất:**

Phần tử được tìm kiếm bằng với phần tử đầu tiên → 1 phép so sánh

**Trường hợp xấu nhất:** $2\log_2 n + 2$ phép so sánh

**Trường hợp trung bình:** xấp xỉ $2\log_2 n$ phép so sánh

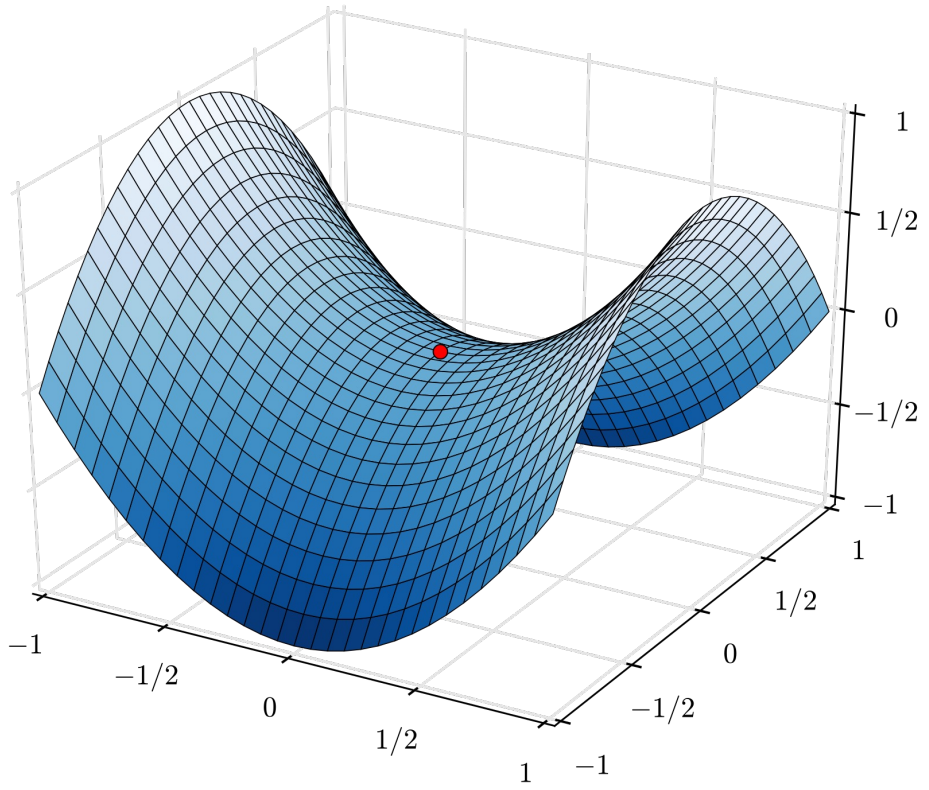
# EXERCISES

**Bài 1**: Viết chương trình sử dụng tìm kiếm tuyến tính và tìm kiếm nhị phân để tìm vị trí của một phần tử ở trong mảng.

**Bài 2**: Viết chương trình sử dụng tìm kiếm tuyến tính và tìm kiếm nhị phân sử dụng đệ quy để tìm vị trí của một phần tử ở trong mảng.

# EXERCISES

**Bài 3**: Một điểm yên ngựa (saddle point) trong mảng 2 chiều là một phần tử mà là giá trị lớn nhất trong hàng của nó và giá trị nhỏ nhất trong cột của nó. Viết chương trình xác định điểm yên ngựa (nếu có) của một mảng 2 chiều.



# THANK YOU FOR YOUR ATTENTIONS