

TT & CTDL

64IT5

Chương I: NHẬP MÔN

1.1 Mục tiêu

1.2 Phép phép

1.3 Thuật toán

1.4 Biến toàn cục & cục bộ

Vòng lặp:

1.5 Cấp phát
 a) Kiểu nguyên thủy (primitive types)
 b) Kiểu ^{lớp} _{array} record / class

Chương II: TÌM KIẾM & ĐỘ PHÚC TẠP TT

2.1 DL tìm vào

2.2 SX chọn

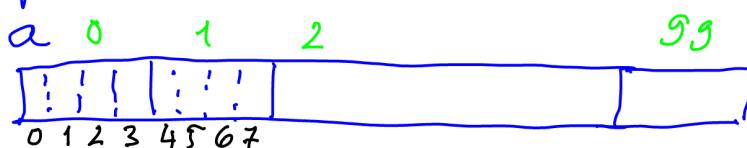
$t(n) \rightarrow O(n)$

$s(n) \rightarrow O(n)$

2.4 SX nổi bật

Chương III: CẤP PHÁT ĐỘNG

3.1 Cấp phát tĩnh



$a[0]$: byte #0 → byte #3

$a[1]$: byte #4 → byte #7

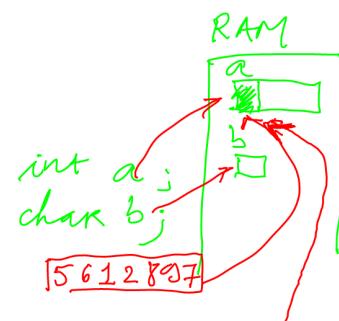
3.2 Còn trỏ

Cách truy cập: a) Dùng tên biến
 b) Dùng địa chỉ (còn trỏ)

pointer

Mảng tĩnh
 $\text{int } a[100];$

Symbol table	
tên	địa chỉ
a	562897
b	



Dùng tên biến: $a = a + 1;$

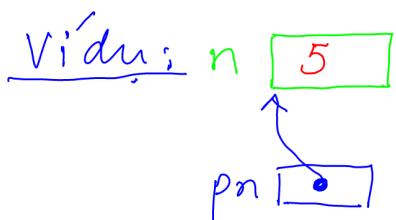
5612892

Dùng con trỏ: $\text{int} * \text{pa};$

sa

$\text{pa} = \&a;$ phép toán lấy địa chỉ

$\Rightarrow * \text{pa} = * \text{pa} + 1;$ phép toán lũy thừa tăng cấp



$\text{int} n;$

$n = 5;$

$\text{int} * \text{pn};$

$\text{pn} = \&n;$

$\text{cout} \ll * \text{pn};$

$n = 6;$

$\text{cout} \ll * \text{pn};$

$* \text{pn} = 11;$

$\text{cout} \ll * \text{pn};$

Giá trị NULL: (nullptr)

Ví dụ: n

np

np

$\text{if} (\text{np} == \text{nullptr}) \rightarrow$ chưa khởi tạo

$\Rightarrow \text{if} (\text{np})$

\rightarrow khởi tạo rồi

(có giá trị)

{ } ... ←

}

3.3 Cấp phát động:

$\text{int} * \text{np} = \text{nullptr};$

$\Rightarrow \text{np} = \text{new}(\text{int});$ kiểm

cấp phát động

tùm 2 việc:
- cấp phát RAM
- trả về địa

np

~~1 5~~

bổ nh
cấp ph
động

$* \text{np} = 5;$



$\Rightarrow \text{delete np;}$ trả bối nhớ cấp phát

$\text{np} = \text{nullptr};$

Ví dụ:

$\text{int} a = 5;$

$\text{int} * \text{p} = \&a;$

$\text{cout} \ll * \text{p};$

$\text{p} = \text{new}(\text{int});$

$\text{cout} \ll * \text{p};$

~~$\text{p} = \text{new}(\text{char});$~~

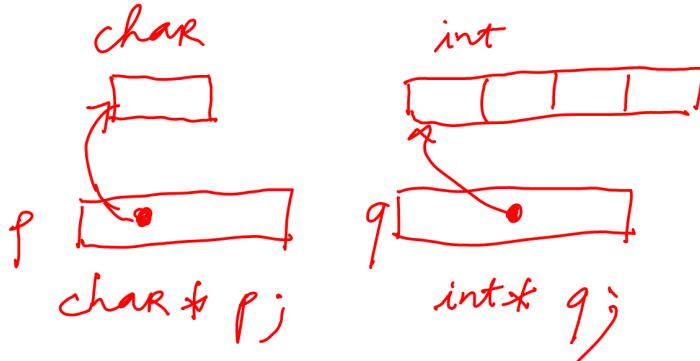
~~$* \text{p} = 7;$~~

~~$\text{cout} \ll * \text{p};$~~

~~$\text{p} = \&a;$~~

~~$\text{cout} \ll * \text{p};$~~

← orphan memory

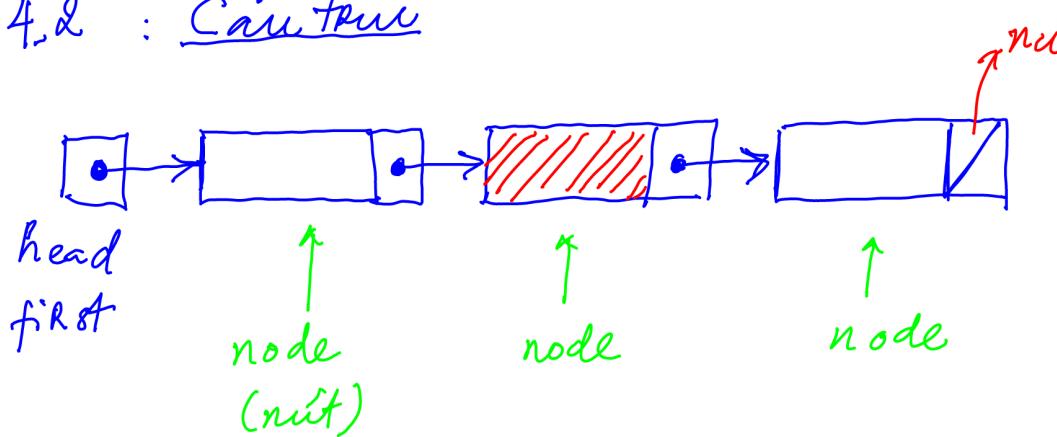


Chương IV: DANH SÁCH LIÊN KẾT

4.1 : Mục đích:

- a) Mảng array phải cẩn thận (có tiềm năng thiếu, hoặc thừa bộ nhớ), \rightarrow DLLK.
- b) DLLK tiện lợi như mảng:
 - * Tích hợp các phần tử có kiểu/cấu trúc gần nhau.
 - * Cho phép có các thao tác như: duyệt, chèn, xóa

4.2 : Cấu trúc



```
class Node {
    int msv;
    string ten;
    Node* next;
}
```

```
class SingleLinkedList {
    Node* head;
}
```

→ nội dung DL via phần tử của biến `next`

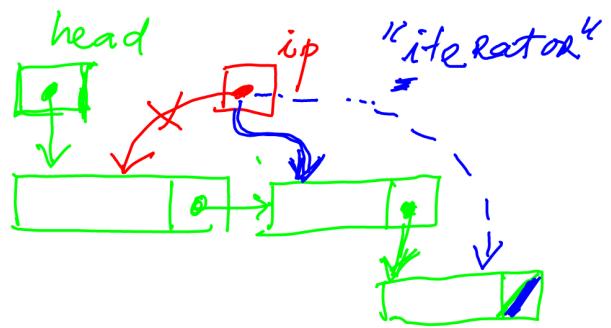
4.3 Duyệt

```

class SingleLinkedList {
    Node* head; ←
    void print(); ←
};

void
SingleLinkedList::print()
{
    Node* ip;
    ip = head; /* ip = this->head */
    while (ip)
    {
        cout << ip->ten << endl;
        ip = ip->next; ←
    }
}

```



Cách dùng

```
SingleLinkedList lop64TS;
```

```
:
```

```
lop64TS.print();
```

4.4 Chèn (vào đầu ds)

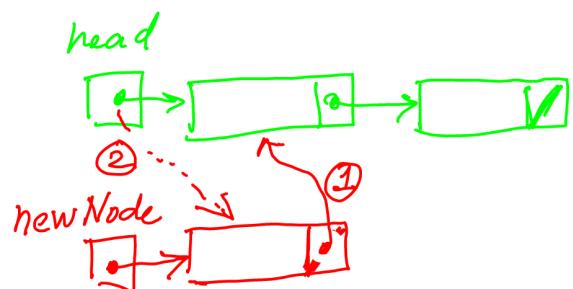
```
class SingleLinkedList
```

```
{
    Node* head;
```

```
    void print();
```

```
    void insert(Node* newNode);
```

```
;
SingleLinkedList();
```



```

void SingleLinkedList::insert(Node* newNode)
{
    if (!newNode) return;
    newNode->next = head;
    head = newNode;
}

```

kiểm tra
"edge cases"

```
SingleLinkedList::SingleLinkedList()
```

```
{
    head = nullptr;
}
```

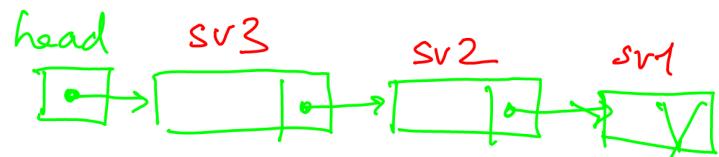
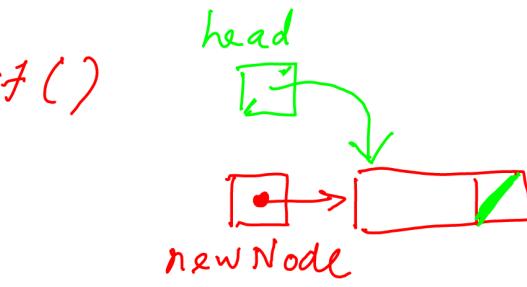
Ví dụ:

```
SingleLinkedList* lop64IT5 = new SingleLinkedList;
```

```
Node* sv1;
Node* sv2;
Node* sv3;
```

⋮
⋮

```
lop64IT5->insert(sv1);
lop64IT5->insert(sv2);
lop64IT5->insert(sv3);
```



```
class Node
```

```
{
    int mssv;
    string ten;
    Node* next;
}
```

```
Node( int m; string t);
```

```
};
```

```
Node::Node( int m; string t)
```

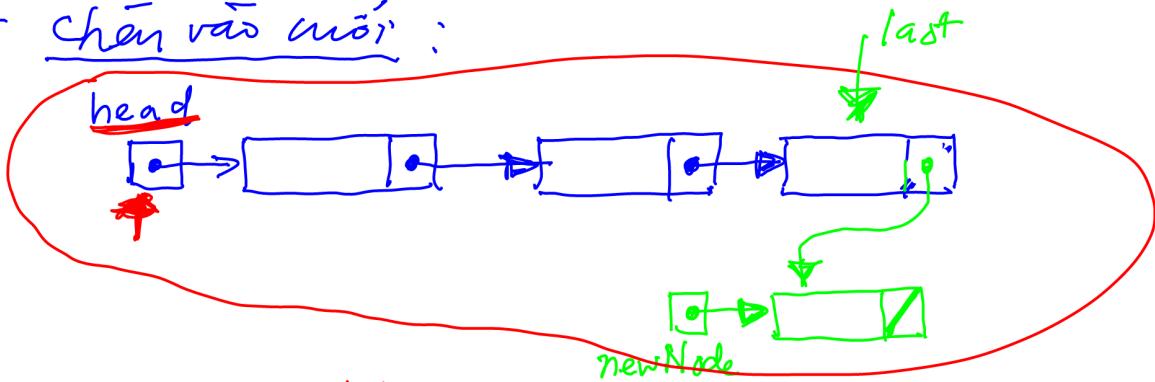
```
{
    mssv = m;
    ten = t;
}
```

sv1 = new Node(111,
"Anh Hài");

sv2 = new Node(222, "Cô Phuay");

sv3 = new Node(333, "Đinh Thúy");

4.5 Chèn vào cuối:



```
class SingleLinkedList
```

```
{ Node* head;
```

```
:
```

```
void insertLast (Node* newNode); ← Ⓛ
```

```
};
```

```
void SingleLinkedList::insertLast (Node* newNode)
```

```
{
```

```
Node* ip = head;
```

```
if (!head)
```

```
{ head = newNode;
    return;
}
```

```
Node* last = nullptr;
```

```
while (ip)
{
```

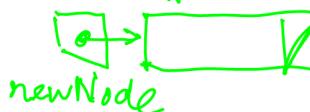
```
    last = ip;
```

```
    ip = ip->next;
}
```

```
last->next = newNode; ←
```

```
}
```

head



bool SingleLinkedList::isEmpty()

```
{
    return (!head);
}
```

```
class SingleLinkedList
```

```
{
```

```
Node* head;
```

```
Node* last;
```

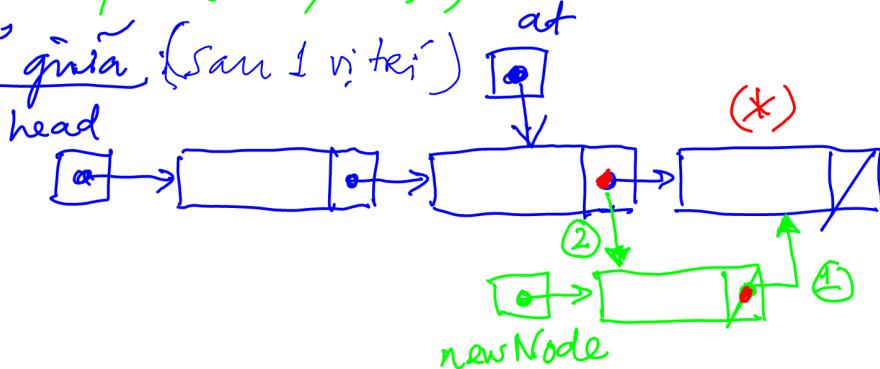
```
;
```

```
void insertLast (Node* newNode);
```

```
}
```


ds → insertAfter(sr1, sr2);

4.7 Chèn ở giữa (sam 1 vị trí)



class SingleLinkedList

{

 void insertAfter(Node* at, Node* newNode);

}

void SingleLinkedList::insertAfter(Node* at, Node* newNode)

{

 if (!head)

 head = newNode;

 last = newNode;

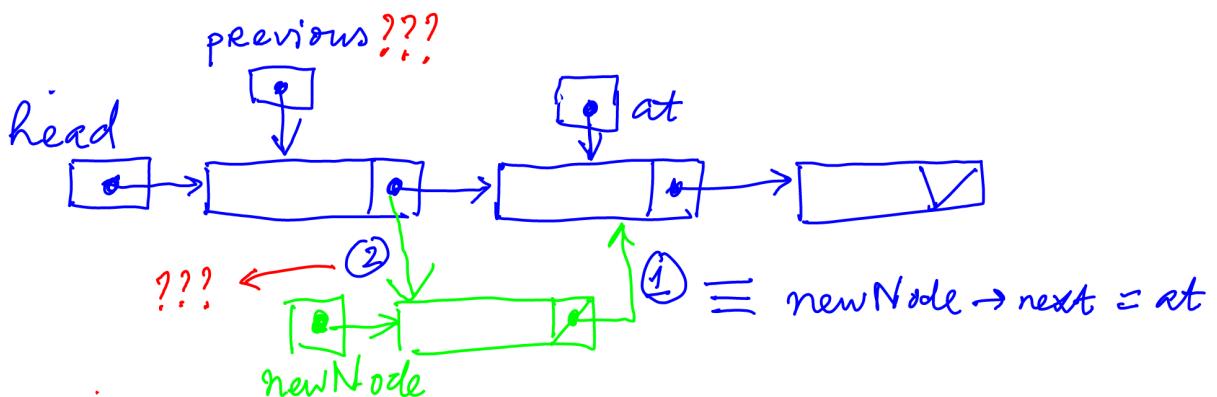
 return;

 newNode->next = at->next;

 at->next = newNode;

}

4.8 Chèn giữa (traverse vị trí nào đó)



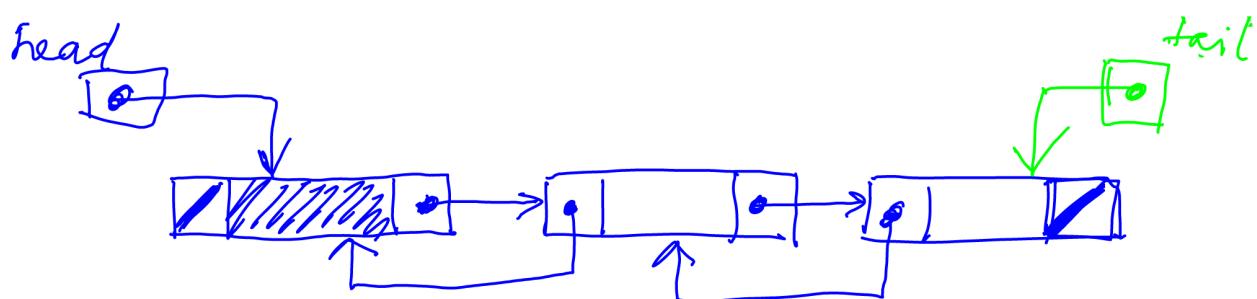
Làm sao để biết previous : Bằng cách duyệt ds

cho đến khi gặp at

Giải quyết = DANH SÁCH LIÊN KẾT KÉP

Chuỗi IV : DANH SÁCH LK KÉP

5.1 Cách tạo & Khai báo lớp



```

class Node
{
    int msv;
    string ten;
    Node* next;
    Node* previous;
}

```

class DoubleLinkedList

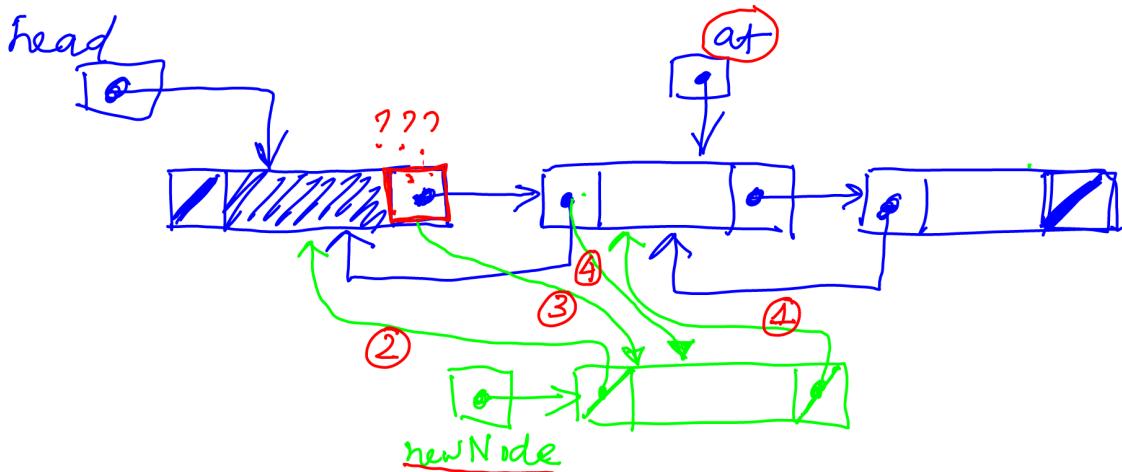
{

Node* head;

Node* tail;

}

5.2 Chèn giữa ds (tạo 1 vị trí nhất định)

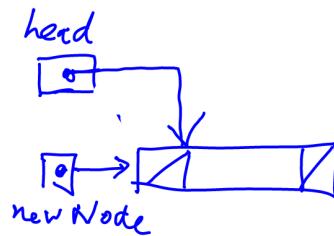


```

class DoubleLinkedList
{
    :
    void insertBefore(Node* at, Node* newNode);
};

void DoubleLinkedList::insertBefore(Node* at, Node* newNode)
{
    if (!head)
    {
        head = newNode;
        return;
    }
}

```



```

newNode->next = at;
newNode->previous = at->previous;

if (at->previous)
{
    at->previous->next = newNode;
}
at->previous = newNode;
}

```

5.3 Xoa

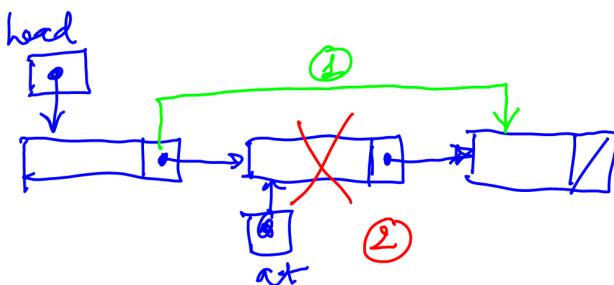
```

class SingleLinkedList
{
    :
    void remove (Node* at);
};

void SingleLinkedList::remove (Node* at)
{

```

NÊN ĐÙNG DLL KÉP



Khi xóa:

```

sv = ds->lookup(123);
ds->remove(sv);
delete sv;

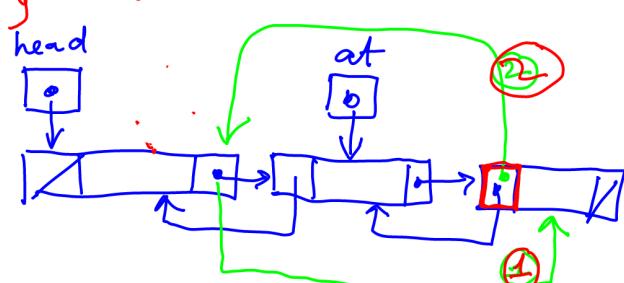
```

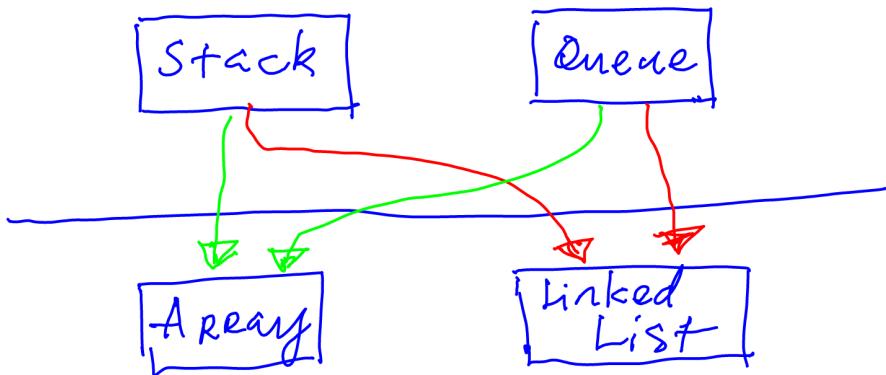
```

class DoubleLinkedList
{
    :
    void remove (Node* at);
};

void DoubleLinkedList::remove (...)
{
    if (at->previous)
    {
        at->previous->next
            = at->next;
    }
    if (at->next)
    {
        at->next->previous
            = at->previous;
    }
}

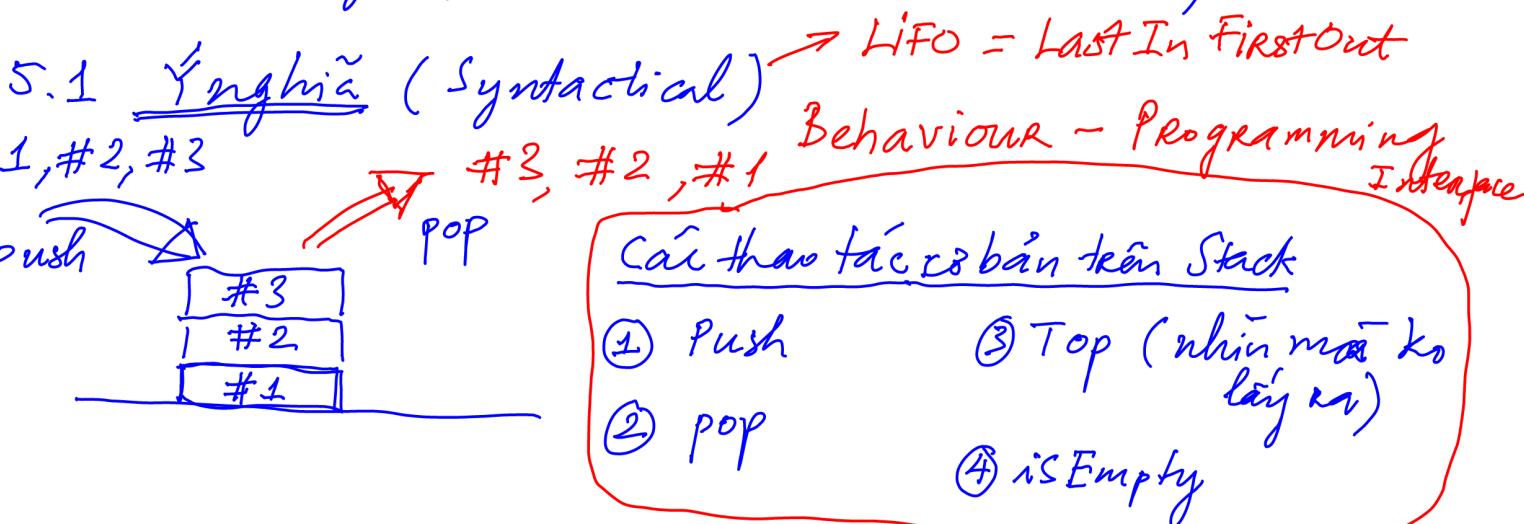
```





Encapsulation
(Information hiding)
programming interface
≡ contract

Chương VI : STACK (NGĂN XẾP)



5.2 Cài đặt dùng array:

```
#define STACK_MAX_SIZE 100
```

```
class Stack
```

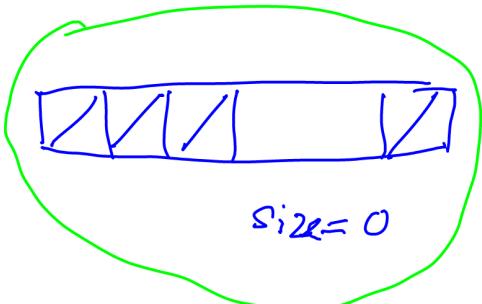
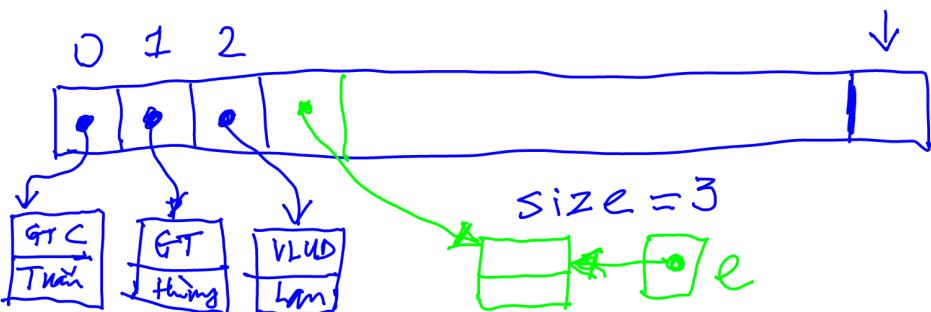
```
{
    StackElement* a[STACK_MAX_SIZE];
    int size;
```

```
void push(StackElement* e);
StackElement* pop();
StackElement* top();
bool isEmpty();
```

```
}
```

```
class StackElement
{
    string title;
    string author;
}
```

interface



```
void Stack::push(StackElement* e)
{
    if (size >= STACK_MAX_SIZE) return;
    a[size] = e;
    size++;
}
```

```
StackElement* Stack::pop()
```

```
{
    if (size <= 0) return nullptr;
    size--;
    return a[size];
}
```

```
StackElement* Stack::top()
```

```
{
    if (size <= 0) return nullptr;
    return a[size - 1];
}
```

```
bool Stack::isEmpty()
```

```
{
    return (!size); // return (size == 0)
}
```

5.3 Cài đặt dùng Linked List

```
class Stack
```

```
{
    LinkedList* l; // để đón các StackElement
```

 → interface (göi là phần 5.2)

```
};
```

NOTE: PHẢI KHÓI TẠO l TRƯỚC TỐI 1 DS LIÊN KẾT
(rỗng)

```
void Stack::push(StackElement* e)
```

```
{  
    l->insertLast(e);  
}
```

```
bool Stack::isEmpty()
```

```
{  
    return l->isEmpty();  
}
```

```
StackElement* Stack::pop()
```

```
{  
    StackElement* temp = l->tail;  
}
```

```
    l->remove(l->tail);
```

```
    return temp;
```

```
StackElement* Stack::top()
```

```
{  
    return l->tail;  
}
```

```
Stack::Stack()
```

```
{  
    l = new LinkedList();  
}
```

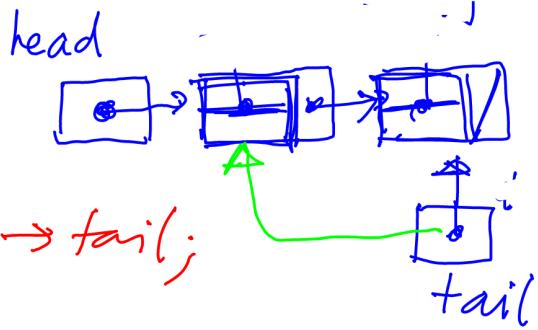
Về phía người dùng:

```
Stack* abc;
```

```
StackElement* kg; ←
```

```
kg = abc->pop();
```

```
cout << kg->title;
```



```
StackElement* s;  
s = new StackElement();  
s->title = "Tho TAK";  
s->author = "TARI";  
abc->push(s);
```

LÂM BÀI TRÊN ONLINE GDB

Bước 1: onlinegdb.com

Bước 2: Đăng ký người dùng (tạo account)
dùng email của trường.

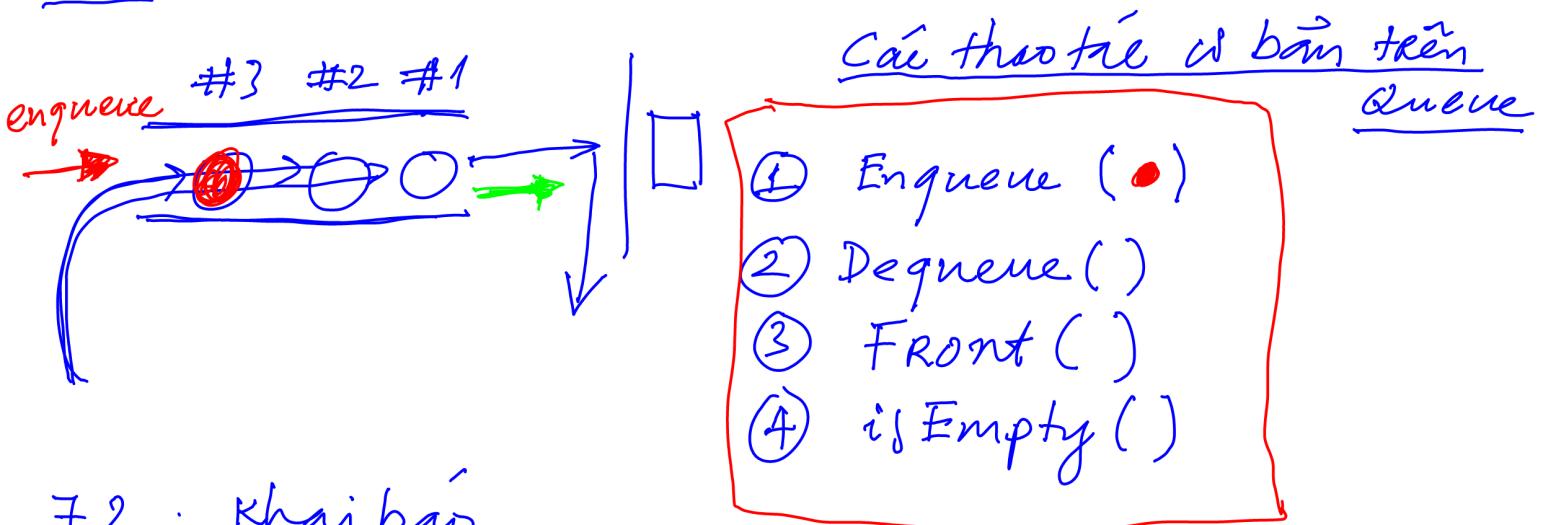
Phải verify email.

Số tên gọi: Vu Xuan Tai - 174164

Bước 3: Đăng link invite
để join vào lớp.

Chương VII : QUEUE (HÀNG ĐỢI)

7.1: Hành vi FIFO = First In First Out



7.2: Khai báo

```
class QueueElement
{
    string msv;
    string ten;
};

class Node
{
    QueueElement* d; ←
    Node* next;
};
```

```

class Queue
{
    private:
        Node* first;
        Node* last;
    }

    public:
        Queue();
        ~Queue();

        QueueElement* enqueue(QueueElement* e);
        QueueElement* dequeue();
        QueueElement* front();
        bool isEmpty();
}

```

design pattern
encapsulation

Contract

QueueElement* Queue::enqueue(QueueElement* e)

```

    Node* n = new Node();

```

```

    n->data = e;
    n->next = nullptr;
    if (last)
        last->next = n;
    else

```

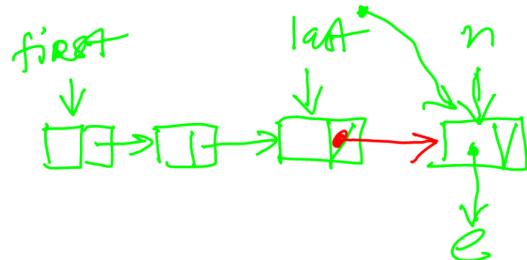
```

        first = n;
    }

```

last = n;

return e;



}

QueueElement* Queue::dequeue()

```

    if (!first) return nullptr;
    Node* n = first;

```



-

```
first = first->next;
if (!first) last = nullptr;
QueueElement* e = n->d;
delete n;
Return e;
}
```

```
QueueElement* Queue:: front()
{
    if (!first) return nullptr;
    return first->d;
}
```

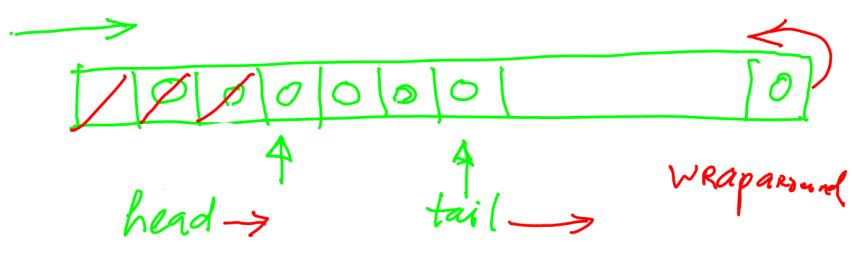
CÀI ĐẶT BẰNG ARRAY

```
#define QUEUE_CAPACITY 2048

class Queue
{
private:
    StackElement* a [QUEUE_CAPACITY];
    int head, tail, length;
public:
    Queue();
    ~Queue();
    StackElement* enqueue(StackElement* q);
    StackElement* dequeue();
    StackElement* front();
    bool isEmpty();
};
```

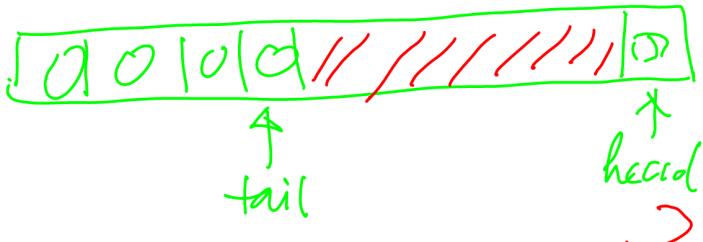
```
Queue :: Queue()
```

```
{ length = 0;  
head = -1; tail = -1;  
}
```



```
StackElement* Queue:: enqueue(StackElement* e)
```

```
{ if (!length)  
{ a[0] = e;  
head = 0;  
tail = 0; length = 1;  
return e;  
}
```



```
if (length >= QUEUE_CAPACITY)
```

```
{ return nullptr;  
}
```

```
tail++;
```

```
if (tail >= QUEUE_CAPACITY) tail = 0; //wraparound
```

```
a[tail] = e;
```

```
length++;
```

```
return e;
```

```
}
```

```
StackElement* Queue:: deQueue()
```

```
{ if (!length) return nullptr;
```

```
int h = head;
```

```
head++;
```

```
if (head >= QUEUE_CAPACITY) head = 0;
```

```
length--;
```

```
return a[h];
```

```
}
```

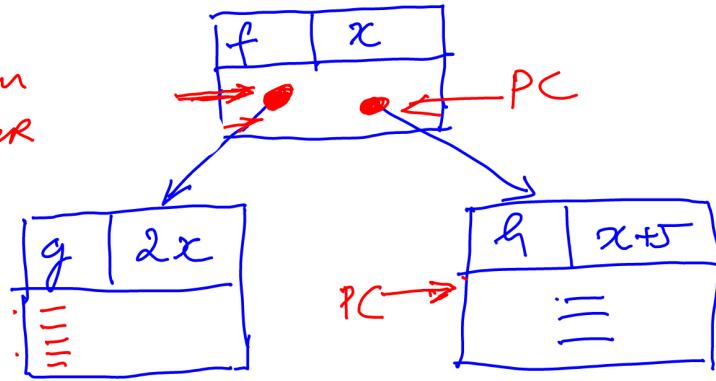
Chương VIII : ĐỀ QUY

① Gọi hàm đệ quy:

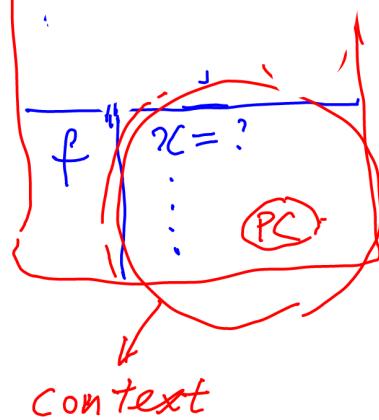
1.1. Gọi hàm khử đệ quy:

$$f(x) = g(2x) + h(x+5)$$

PC = Program Counter

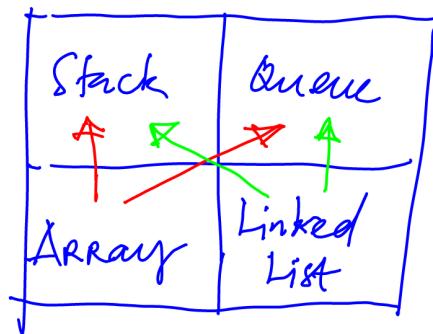


Stack



→ Single Linked-List

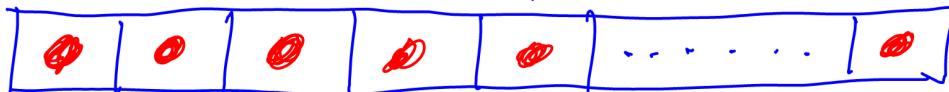
- Stack
- Queue



Chương IX: SẮP XẾP & TÌM KIẾM

A) TÌM KIẾM:

① Đầu vào: Có 1 mảng chứa dữ liệu, có 1 target
Tìm (viết) vị trí của phần tử có "dữ liệu" \equiv target



Nhận xét: Bài toán tìm kiếm là một loạt các phép so sánh ($= ?$)

Hiệu năng phụ thuộc vào # câu phép so sánh

Kết quả:
- Nếu tìm được - thi trả về vị trí (của phần tử đầu tiên)
- Nếu ko có thì trả lại: KO TÌM ĐƯỢC!!

② Thuật toán: Có 2 trường hợp:
- Dữ liệu bất kỳ (ngẫu nhiên)
- Dữ liệu đã được sắp xếp

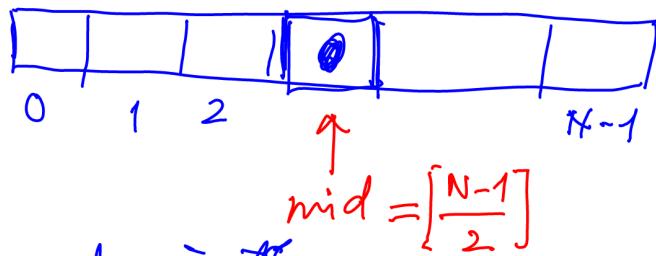
DL Ngẫu nhiên: (Duyệt mảng tuần tự)

(quá trình $<, >, =$)

Tìm kiếm tuần tự



DL đã được sắp xếp: Tìm kiếm nhị phân



b1 - Chọn mid nào đó:

chọn trung bình của đầu và cuối

b2 So sánh A[mid] với target:

* = : trả về chỉ số mid.

* target < A[mid] : lấy nửa bên trái

* target > A[mid] : _____ phải:

- Lặp lại b1 cho đến khi mảng a ra chúng ta o cùn phón
mico

CÁI ĐẤT:

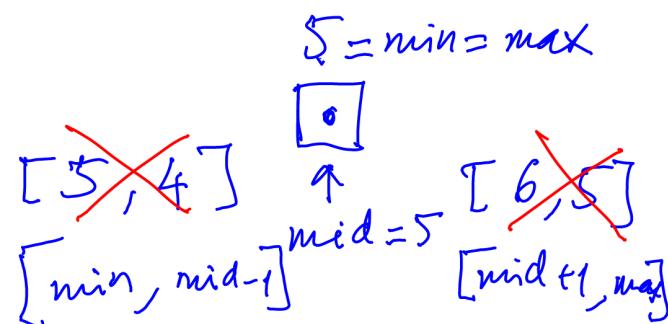
```

int binarySearch ( int A[], int target, int min, int max)
{
    if ( min > max) return -1; // Không tìm thấy

    int mid = (min + max) / 2;

    → if ( target == A[mid]) return mid;
    → if ( target < A[mid])
        return binarySearch (A, target, min, mid-1);
    else
        → return binarySearch (A, target, mid+1, max);
}

```

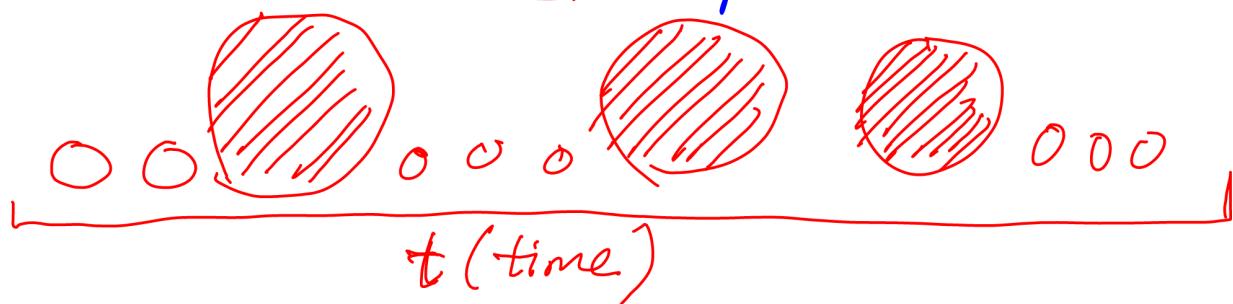


PHÂN TÍCH THUẬT TOÁN:

① Phân tích tìm kiếm toàn bộ:

với N là số phần tử trong dãy biến đầu vào (kích thước dãy
vào)

② Thời gian thực hiện là tổng chi phí của quá trình
tìm thuật toán (về thời gian) phụ thuộc vào
số lần thao tác chính trong thuật toán.



Ký hiệu thời gian thực hiện của thuật toán:

$T(N)$ ~~trong trường hợp xấu nhất~~ = N
ký hiệu chi phí top hill climb

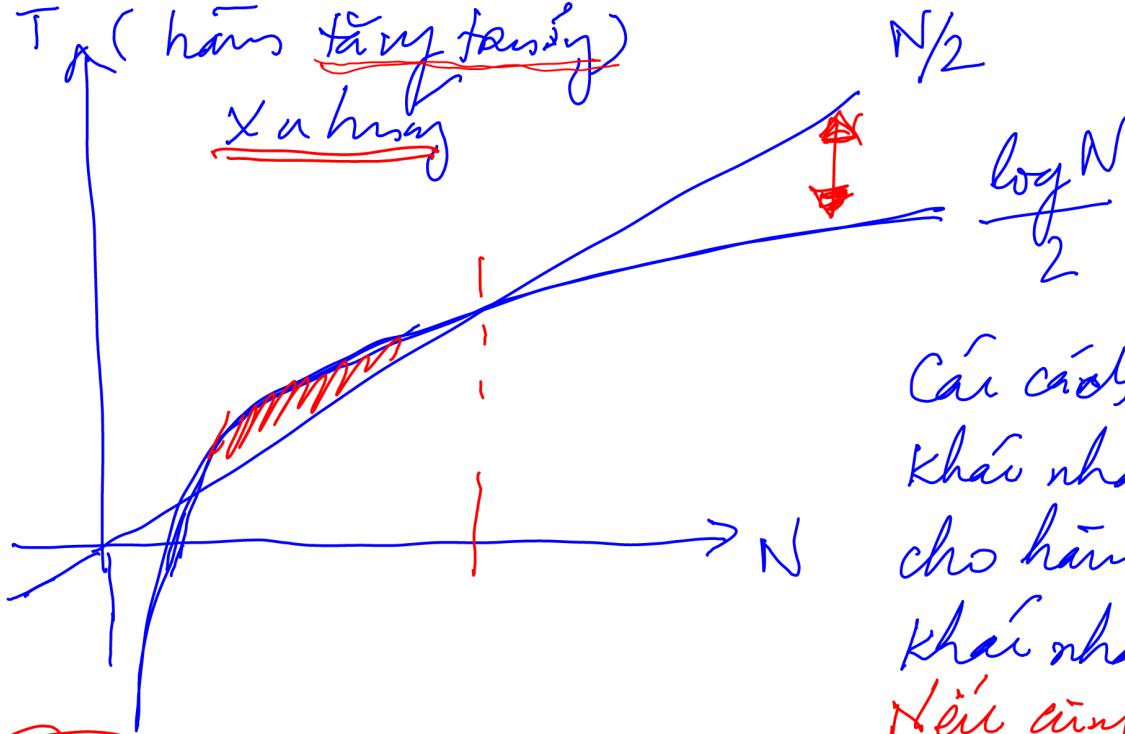
$$T(N)_{\text{worst}} = N \quad | \rightarrow T(N)_{\text{average}} = \frac{1+N}{2}$$

$$T(N)_{\text{best}} = 1 \quad | \quad = \frac{N}{2}$$

phân tích tìm kiếm nhị phân:

$$T(N)_{\text{worst}} = \log_2 N \quad | \rightarrow T(N)_{\text{average}} = \frac{\log_2 N}{2}$$

$$T(N)_{\text{best}} = 1 \quad |$$



LỚP ĐỘ PHÚC
TẤP TT

(cho mục đích phân loại)
trong kí hiệu $O(N)$

Các cách cài đặt
khác nhau sẽ
cho hàm tống tăng
khác nhau.
Nếu cùng thuật toán
thì xu hướng giống
nhau!!!

"big O"
Order

Tìm kiếm truy tìm

$$T(N)_{\text{worst}} = N \Rightarrow O(N)_{\text{worst}} = N$$

$$T(N)_{\text{best}} = 1 \Rightarrow O(N)_{\text{best}} = 1$$

$$T(N)_{\text{average}} = \frac{N+1}{2} \Rightarrow O(N)_{\text{average}} = N$$

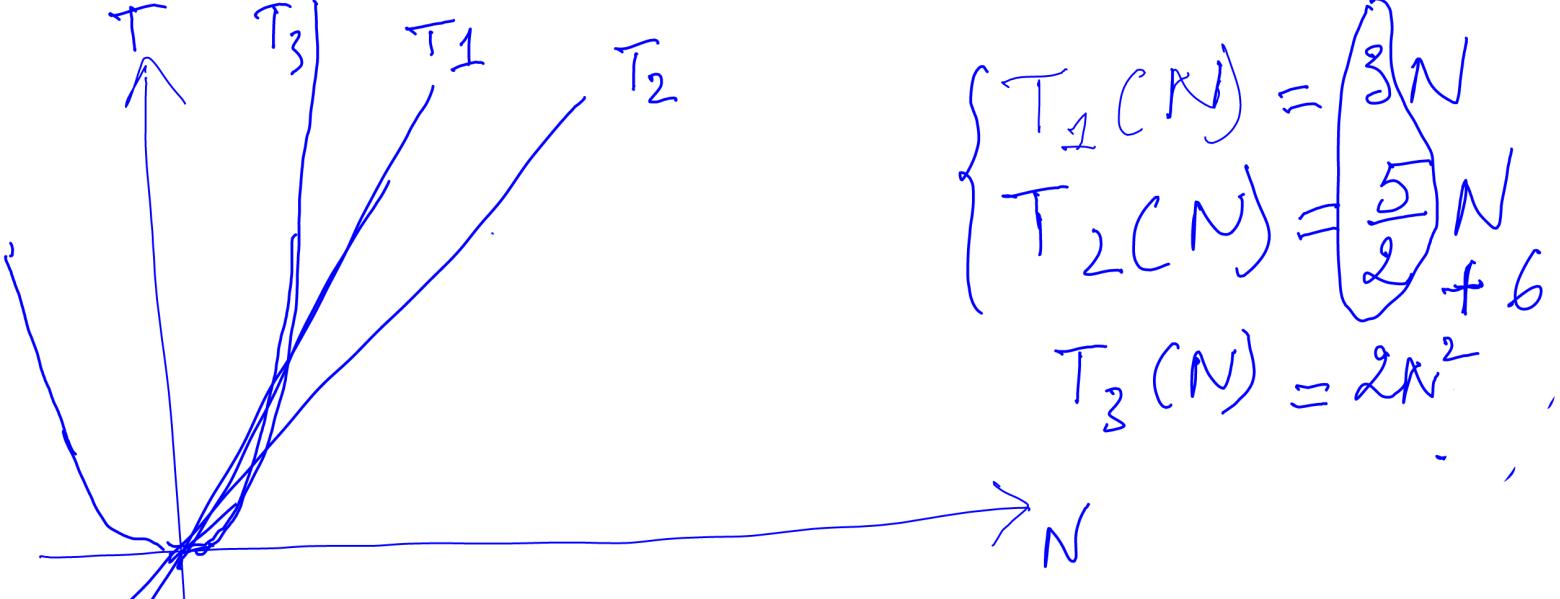
Tìm kiếm nhị phân

$$T(N)_{\text{worst}} = \log N$$

$$T(N)_{\text{best}} = 1$$

$$T(N)_{\text{average}} = \frac{\log N}{2}$$

$$\left| \begin{array}{l} O(N)_{\text{worst}} = \log N \\ O(N)_{\text{best}} = 1 \\ O(N)_{\text{average}} = \log N \end{array} \right.$$



$$\begin{cases} T_1(N) = 3N \\ T_2(N) = \frac{5}{2}N + 6 \end{cases}$$

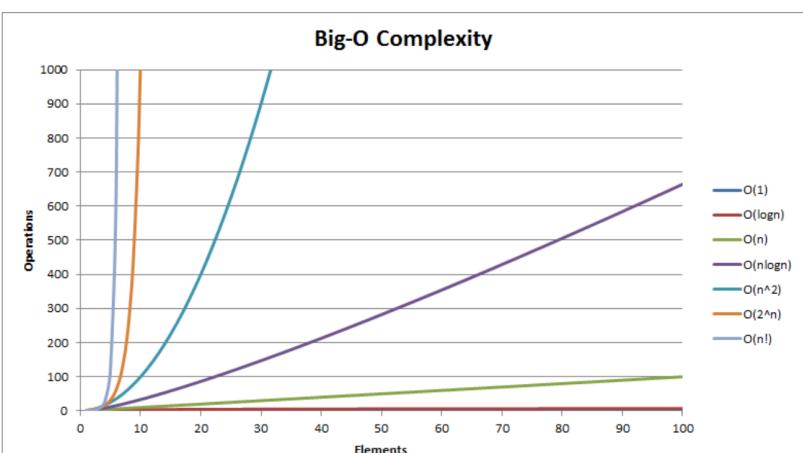
$$T_3(N) = 2N^2$$

$T(N) \rightarrow O(N)$

- b là các factors
- b là các phần đa thứ
và bài nhì hìn

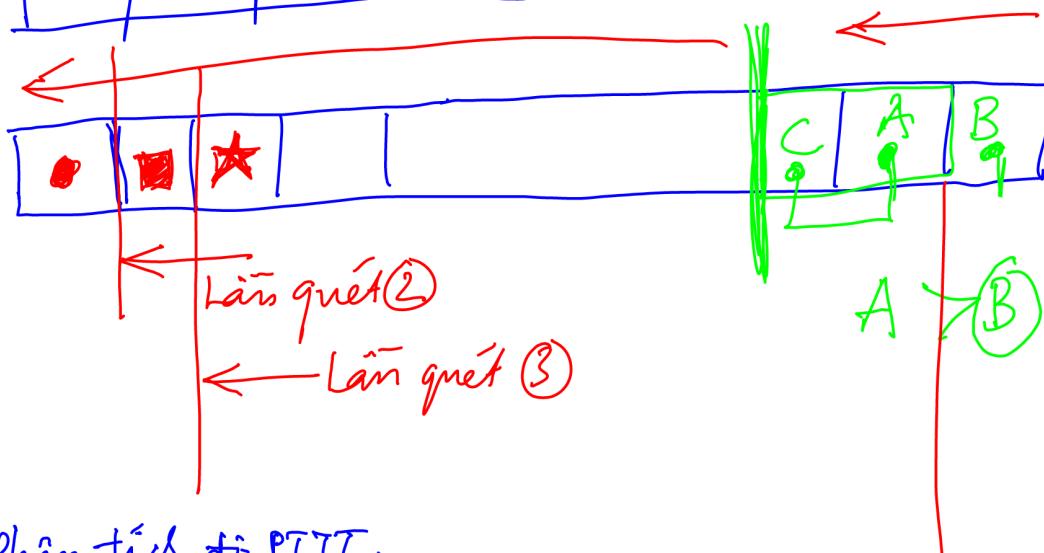
Các lớp phuc biến về DPTTT $O(N)$:

- ① $O(1)$: constant
 - ② $O(\log N)$: Logarithmic
 - ③ $O(N)$: Linear
 - ④ $O(N \log N)$: $N \log N$
 - ⑤ $O(N^2), O(N^3), \dots$: Polynomial
 - ⑥ $O(a^N)$: Exponential
 - ⑦ $O(N!)$: factorial
 - ⑧ $O(N^N)$: N^N
- ↓
 $O(2^N)$



(B) SẮP XẾP

SẮP XẾP NỘI BỘT



Phân tích độ PTTT:

N-1 lần scan.

- Lần 1 : $(N-1)$ phép so sánh

- Lần 2 : $(N-2)$

- Lần $(N-1)$: 1 phép so sánh

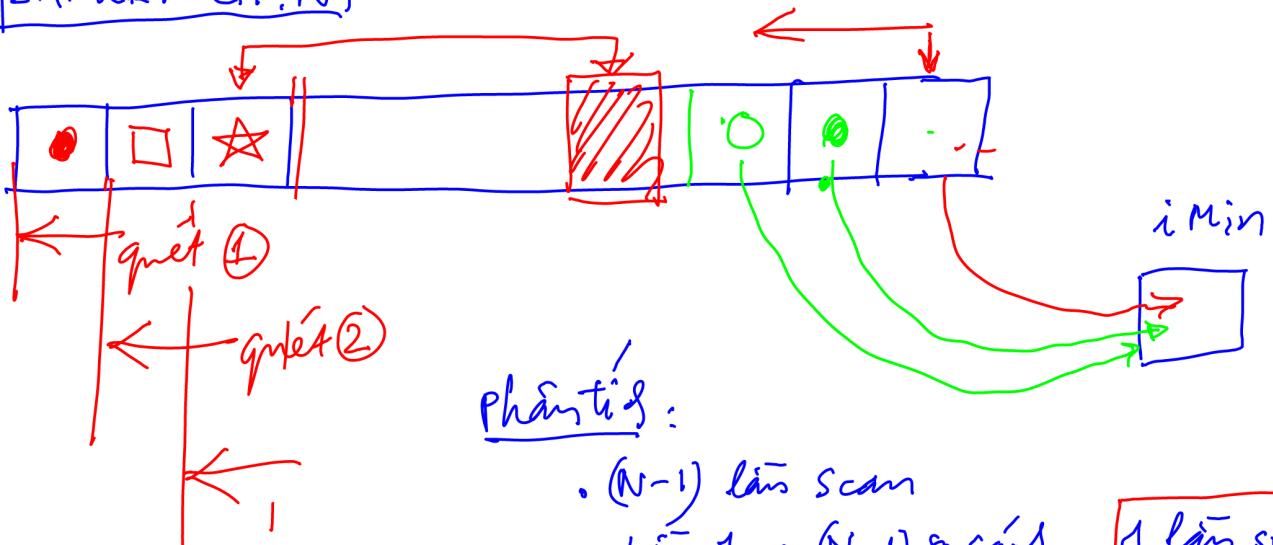
$$\begin{aligned} T(N) &= 1 + \dots + (N-1) \\ &\stackrel{\text{best}}{=} \frac{N(N-1)}{2} \\ &\stackrel{\text{worst}}{=} \frac{1}{2}N^2 - \frac{1}{2}N \end{aligned}$$

~~Bíu với phép swap~~

TH worst: $T_{swap}(N) = \frac{1}{2}N^2 - \frac{1}{2}N$

$$O(N) = N^2 \quad O(N^2)$$

SẮP XẾP CHỌN



Phân tích:

- $(N-1)$ lần scan

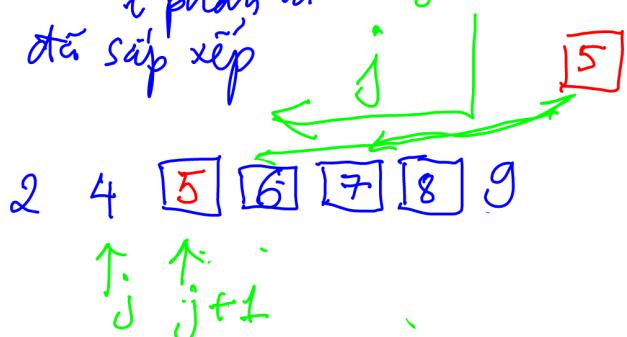
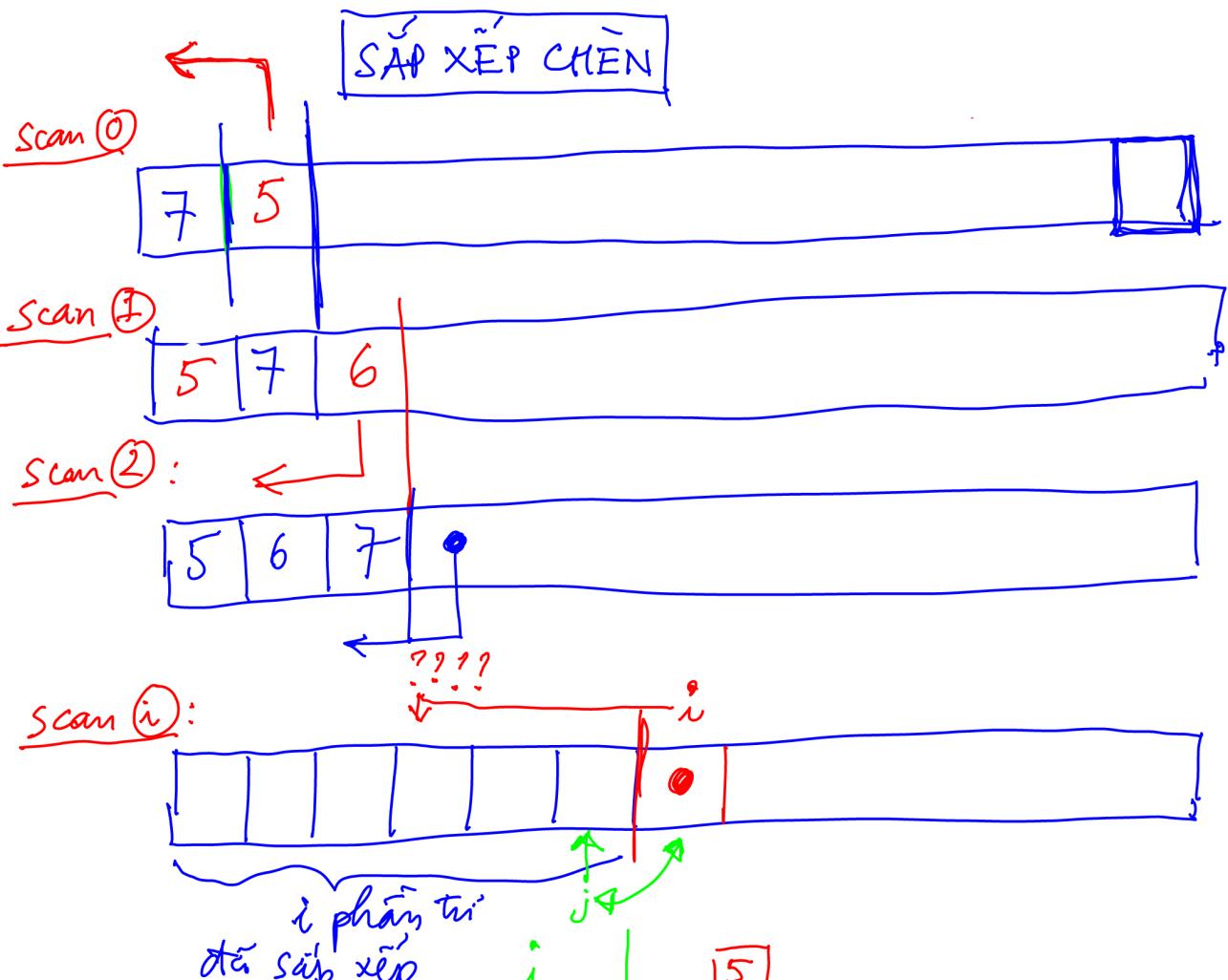
- Lần 1 : $(N-1)$ so sánh, 1 lần swap

- Lần 2 : $(N-2)$ — , 1 —

- Lần $(N-1)$: 1 — , 1 —

$$\begin{aligned}
 T(N) &= 1 + \dots + (N-1) \\
 &= \frac{1}{2} N^2 - \frac{1}{2} N \\
 &\Downarrow O(N^2)
 \end{aligned}$$

$$T_{\text{swap}}(N) = N-1$$



```

void insertionSort( int A[], int n )
{
    for( int i=1; i<n; i++ )
    {
        int value = A[i]; // lùn lại để chèn về sau
        for( int j=i-1; j>=0; j-- )
        {
            if (A[j]>value) A[j+1]=A[j]; // đẩy phải
            else break;
        }
    }
}

```

$A[j+1] = \text{value}; // \text{then}$

}

}

phänotyp:

best: $\text{scan}(i) : 1 \text{ so } \mathcal{O}(1)$

$$T(N) = 1 + \dots + 1 = \underbrace{(N-1)}_{\oplus}$$

worst: $\text{scan}(i) : i \text{ so } \mathcal{O}(N)$

$$T(N) = \frac{1}{2} N^2 - \frac{1}{2} N \Rightarrow \mathcal{O}(N^2)$$

average:

$\text{scan}(i) : i/2 \text{ so } \mathcal{O}(1)$

$$T(N) = 1 + \dots + \frac{1}{2} + \dots + \frac{N-1}{2}$$

$$= \frac{1}{4} N^2 - \frac{1}{4} N \Rightarrow \mathcal{O}(N^2)$$