Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Sự đệ quy

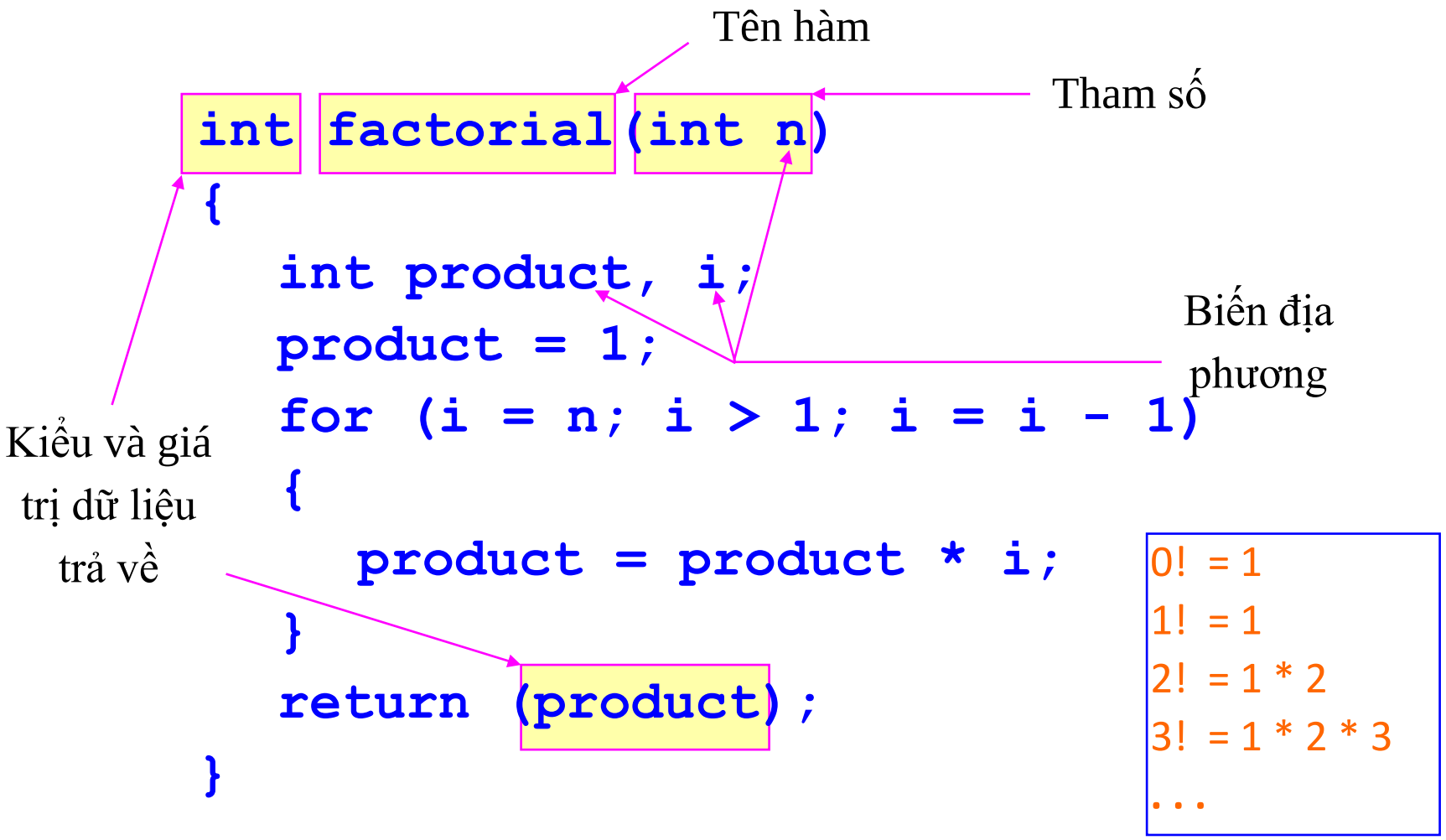
Định Nghĩa Sự Đệ Quy

- Một hàm có tính chất *đệ quy* (recursion) nếu nó gọi đến chính nó:

```
int foo(int x) {
  ...
  y = foo(...);
  ...
}
```

- Câu hỏi:
  - Giải thích cách làm việc của hàm đệ quy?
    - Sẽ được bàn đến hôm nay
  - Tại sao chúng ta viết hàm đệ quy?
    - Bạn sẽ thấy động lực

Hàm Giai Thừa



Hàm Giai Thừa Dùng Đệ Quy

- Định nghĩa của hàm giai thừa vốn đã có tính đệ quy:

  0! = 1! = 1; for n > 1, n! = n(n-1)!

```
int factorial(int n)
{
  int t;
  if (n <= 1)
    t = 1;
  else
    t = n * factorial(n - 1);
  return t;
}
```

0! is 1
1! is 1
n! is n * (n-1)!, for n>1

E.g.: 3! = 3 * 2!
= 3 * 2 * 1!
= 3 * 2 * 1

Ôn Lại: Căn Bản Về Hàm

- Sẽ không tốn sức để lần theo dấu vết của hàm đệ quy nếu bạn nhớ những kiến thức cơ bản về hàm:
  - Các tham số và biến được khai báo trong hàm là các tham số và biến *địa phương* của hàm
    - Được cấp phát bộ nhớ khi bắt đầu thực thi hàm
    - Được giải phóng khi thoát khỏi hàm
  - Các tham số được khởi tạo bằng cách *chép giá trị* của đối số trong lời gọi hàm

Hàm Giai Thừa

factorial(4) =

4 * factorial(3) =

4 * 3 * factorial(2) =

4 * 3 * 2 * factorial(1) =

4 * 3 * 2 * 1 = 24

Yêu Cầu ‘y’ hoặc ‘n’

```
char yes_or_no(void)
{
  char answer = 'X';
  while(answer != 'y' && answer != 'n')
  {
    printf ("Please enter 'y' or 'n':");
    scanf ("%c", &answer);
  }
  return answer;
}
```

Không Cần Vòng Lặp

```
char yes_or_no(void)
{
  char answer;
  printf ("Please enter 'y' or 'n':");
  scanf ("%c", &answer);
  if(answer != 'y' && answer != 'n')
    answer = yes_or_no( );
  return answer;
}
```

Phép Lặp Và Sự Đệ Quy

- *Bất cứ* thuật toán lặp nào cũng có thể thay bằng một thuật toán đệ quy và ngược lại.
- Có một số ngôn ngữ lập trình trong đó đệ quy là sự lựa chọn duy nhất.
- *Một số thuật toán bản thân nó đã được diễn đạt dưới dạng đệ quy một cách tự nhiên*:
  - Nhưng sẽ không hiệu quả nếu sử dụng đệ quy cho những ứng dụng (hoặc thuật toán) *đơn giản*

Khi Nào Dùng Đệ Quy?

- Các trường hợp khác có thể được giải quyết bằng cách chuyển về các trường hợp đơn giản hơn
  - Liên tục thực hiện phép chuyển đổi cho đến khi chuyển về trường hợp đơn giản không đệ quy.

Bài toán Tháp Hà Nội

Truyền thuyết kể rằng : Một nhà toán học Pháp sang thăm Đông Dương đến một ngôi chùa cổ ở Hà Nội thấy các vị sư đang chuyển một chồng đĩa qúy gồm 64 đĩa với kích thước khác nhau từ cột A sang cột C theo cách :

- Mỗi lần chỉ chuyển 1 đĩa .
- Khi chuyển có thể dùng cột trung gian B .
- Trong suốt qúa trình chuyển các chồng đĩa ở các cột luôn được xếp đúng (đĩa có kích thước bé được đặt trên đĩa có kích thước lớn) .

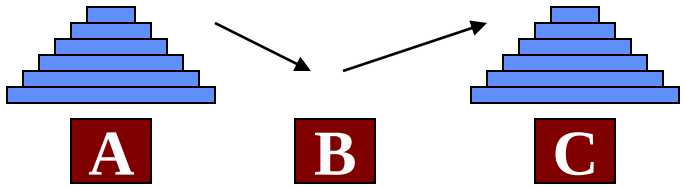
Khi được hỏi các vị sư cho biết khi chuyển xong chồng đĩa thì đến ngày tận thế !.

Với chồng gồm n đĩa cần $2^n - 1$ lần chuyển cơ bản (chuyển 1 đĩa)

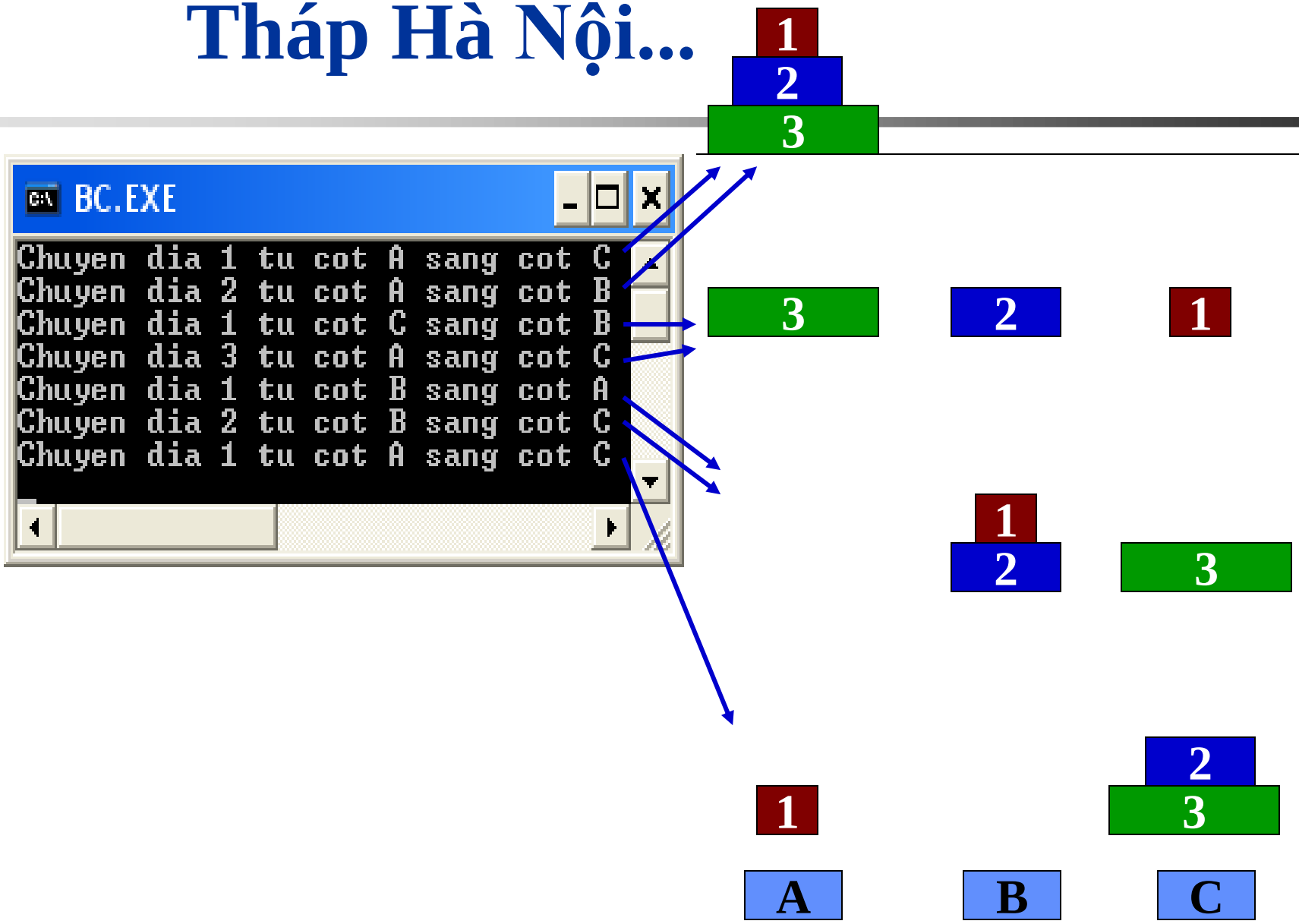
Giả sử thời gian để chuyển 1 đĩa là t giây thì thời gian để chuyển xong chồng 64 đĩa sẽ là :

$T = (2^{64} - 1) * t \ S = 1.84 * 10^{19} * t \ S$

Với t = 1/100 s thì $T = 5.8 * 10^9$ năm = 5.8 tỷ năm .



Tháp Hà Nội...


BC.EXE
Chuyen dia 1 tu cot A sang cot C
Chuyen dia 2 tu cot A sang cot B
Chuyen dia 1 tu cot C sang cot B
Chuyen dia 3 tu cot A sang cot C
Chuyen dia 1 tu cot B sang cot A
Chuyen dia 2 tu cot B sang cot C
Chuyen dia 1 tu cot A sang cot C
1
2
3
A
B
C

Bài toán Tháp Hà Nội

```
1  // ThapHaNoi.cpp : Defines the entry point for the console application.
2  //
3
4  #include <stdio.h>
5
6  void PrintMovePlate(int n,char X,char Y)
7  {
8      printf("\nChuyen dia thu %d tu cot %c sang cot %c",n,X,Y);
9  }
10
11 void ThapHaNoi(int n,char X,char Y,char Z)
12 {
13     if(n>0)
14     {
15         ThapHaNoi(n-1,X,Z,Y);
16         PrintMovePlate(n,X,Z);
17         ThapHaNoi(n-1,Y,X,Z);
18     }
19 }
20
21 void main(void)
22 {
23     ThapHaNoi(3,'A','B','C');
24     return ;
25 }
26
```

Chuyển n đĩa từ cột X sang cột Z nhờ cột trung gian Y

(1) Chuyển n-1 đĩa từ cột X sang cột Y nhờ cột trung gian Z vì các đĩa bên trên là các đĩa nhỏ.

(2) Chuyển đĩa n (to nhất) từ cột X sang cột đích Z.

(3) Làm lại cho n-1 đĩa còn lại đang ở cột Y sang cột Z với X là cột trung gian.

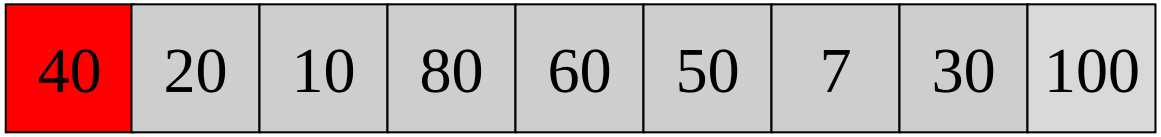
Quick sort - Example

Xếp mảng số nguyên sau:

40	20	10	80	100	50	7	30	60

Chọn phần từ chốt

Chọn phần tử so sánh: ví dụ như phần tử đầu tiên

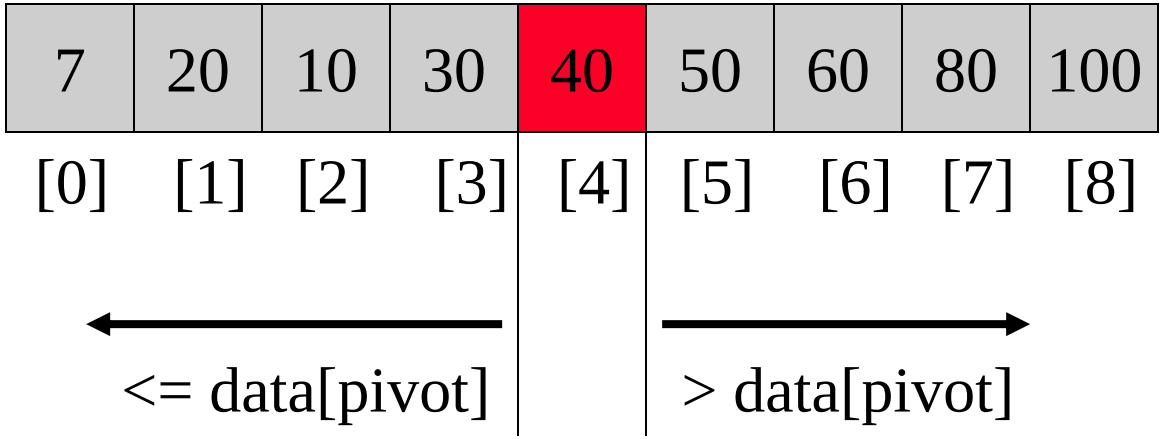


Chia mảng ra thành mảng nhỏ

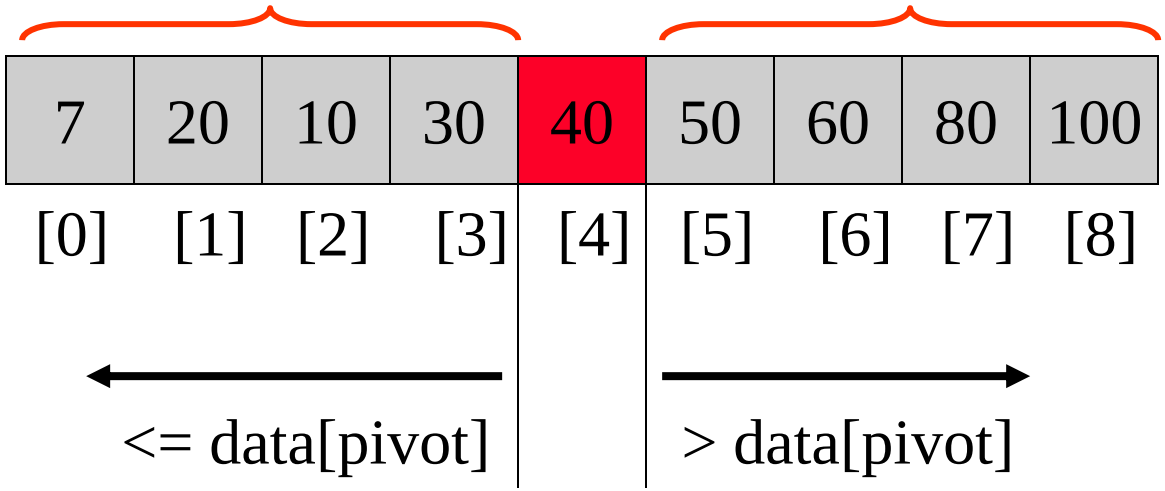
Cho trước phần tử trong mảng, chia mảng thành 2 phần:

1. Mảng gồm các phẩn tử <= phần tử so sánh
2. Mảng gồm các phẩn tử > phần tử so sánh

Partition Result

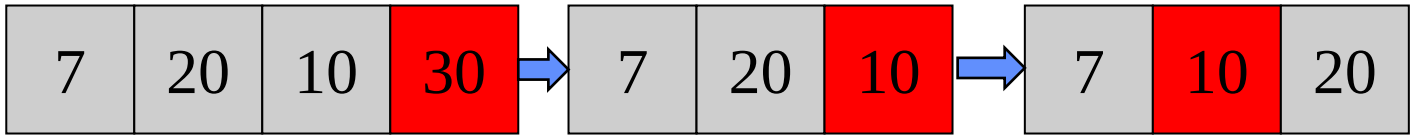


Recursion: Quicksort Sub-arrays



Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- What is best case running time?
  - Recursion:
    1. Partition splits array in two sub-arrays of size n/2
    2. Quicksort each sub-array
  - Depth of recursion tree? $O(\log_2 n)$
  - Number of accesses in partition? O(n)

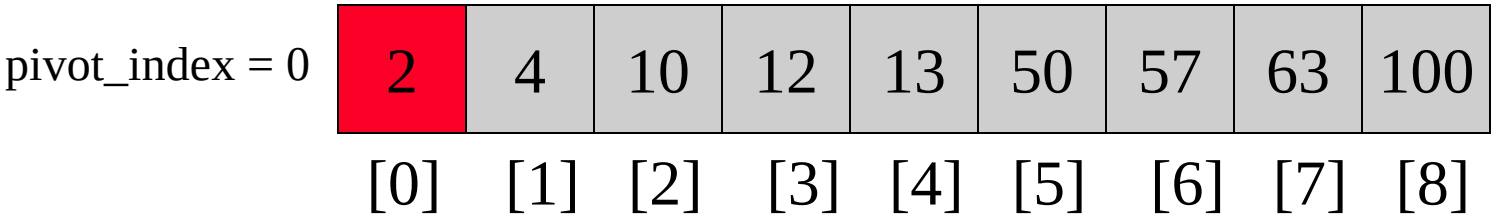


Quicksort Analysis

- Giả sử các phần tử phân bố đều ngẫu nhiên
- Thời gian ngắn nhất: $O(n \log_2 n)$
- Trường hợp xấu nhất?

Quicksort: trường hợp xấu nhất

- Giả thiết là các phần tử đã tự phân bố theo thứ tự tăng dần:



Cách khác chọn điểm chốt (pivot)

Chọn giá trị trung bình của 3 phần từ trong mảng:

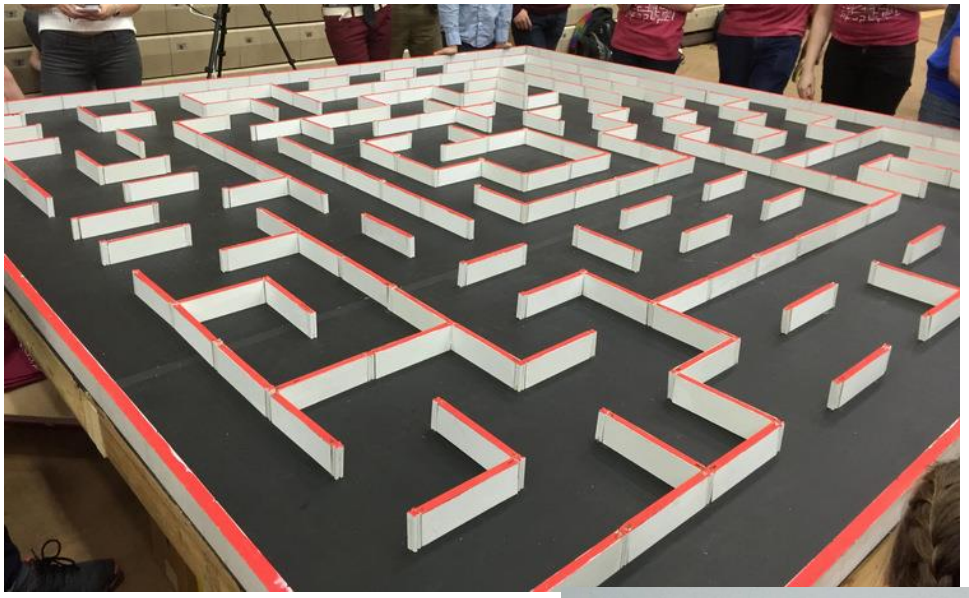
data[0], data[n/2], and data[n-1].

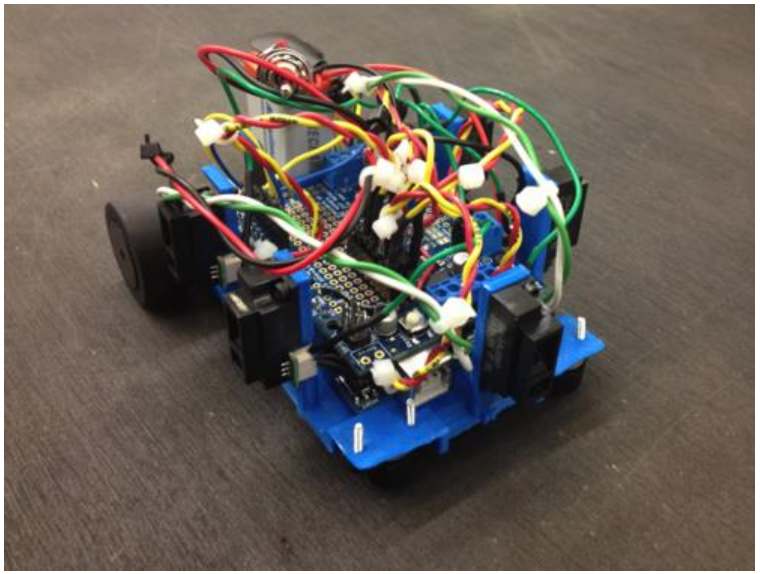
Sử dụng giá trị trung bình này.

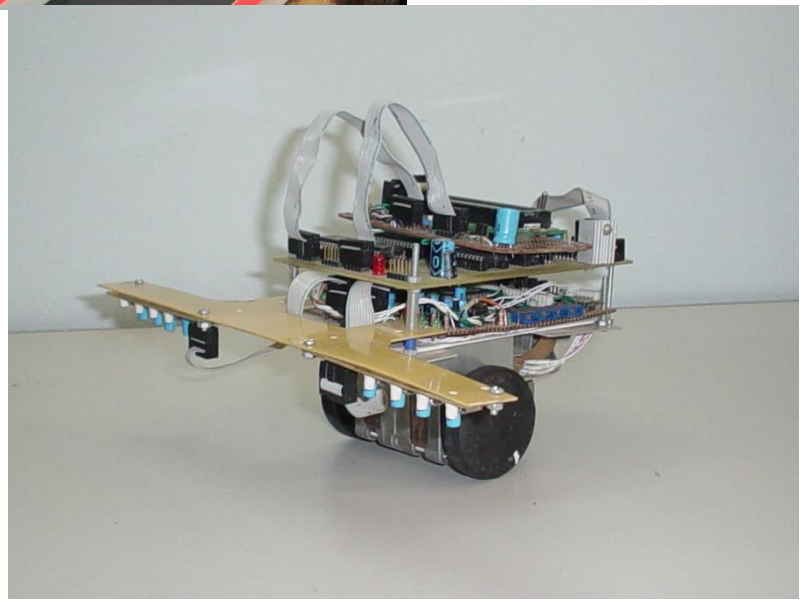
Tối ưu thuật toán Quicksort

- Tìm phần tử chốt
- Nếu kích thước nhỏ hơn và bằng 3:
  - 1 phần tử: không làm gì
  - Nếu có 2 phần từ:
    - if(data[first] > data[second]) swap them
  - Nếu có 3 phần tử : left as an exercise.

Micromouse



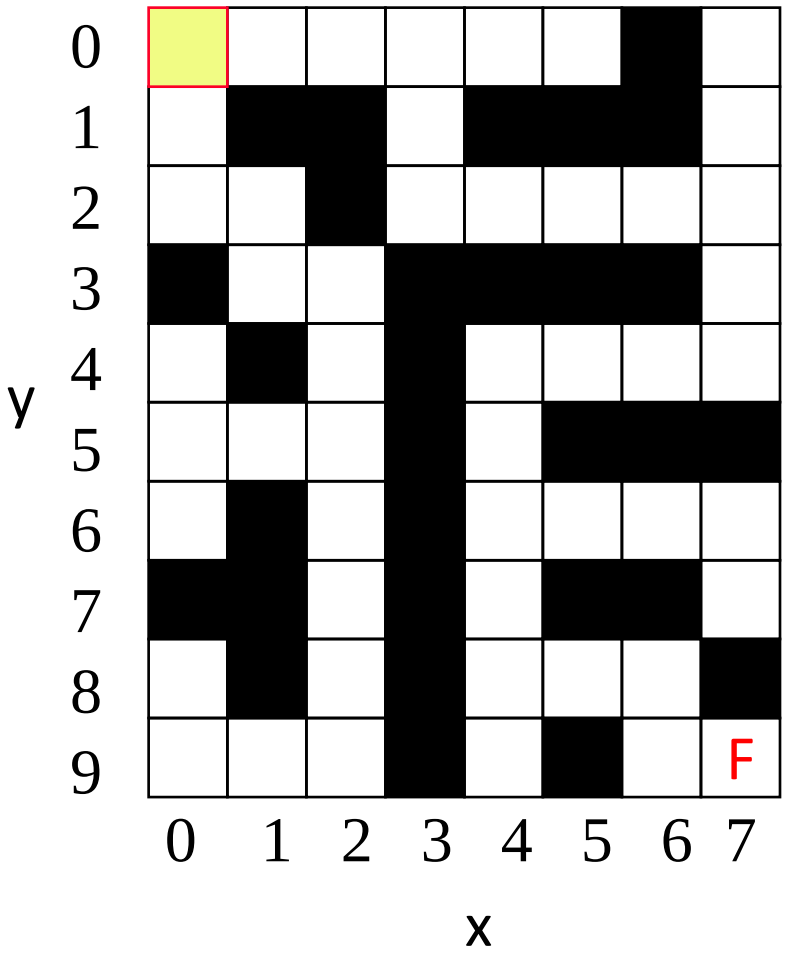




Ví Dụ

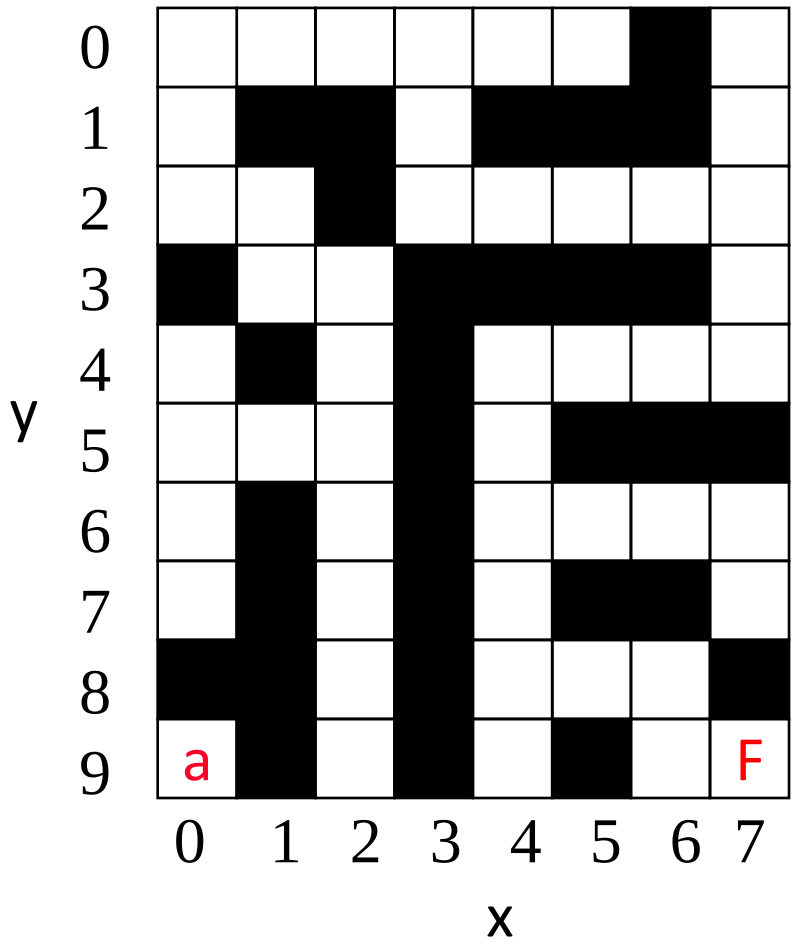
```
/* 'F' means finished!
   'X' means blocked
   ' ' means ok to move */
char maze[MAXX][MAXY];
/* start in yellow */
int x =0, y=0;
```

- Trừ khi bị chặn, robot có thể đi lên, xuống, trái, phải
- Bài toán: liệu có một đường đi tới đích?



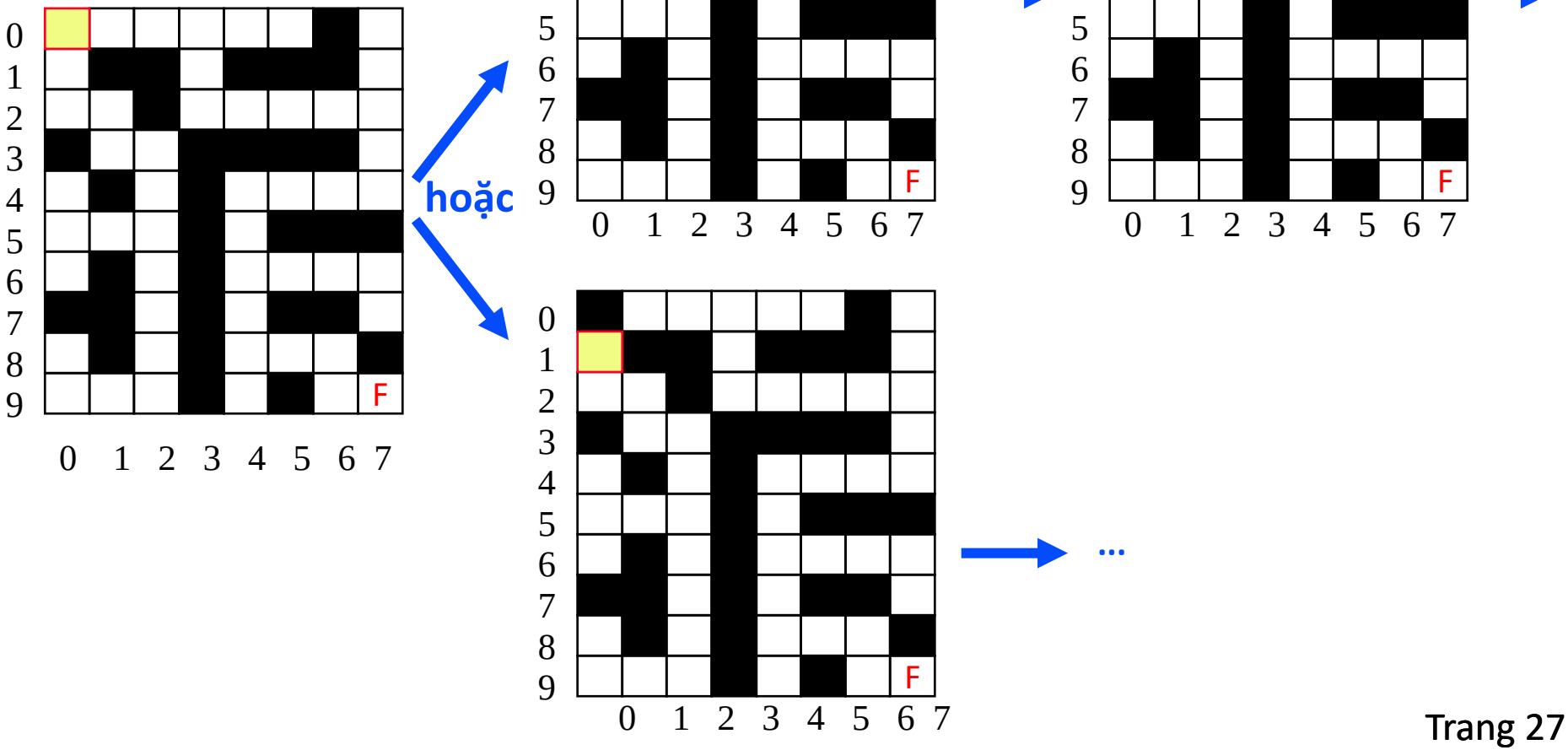
Các Trường Hợp Đơn Giản

- Giả sử robot đang ở vị trí có tọa độ (x,y)
  - if maze[x][y] == 'F'
    - then "yes!"
  - if no place to go
    - then "no!"



Chuyển Về Đơn Giản Hơn

Không cần thiết phải đến một ô hai lần



Hàm Trợ Giúp

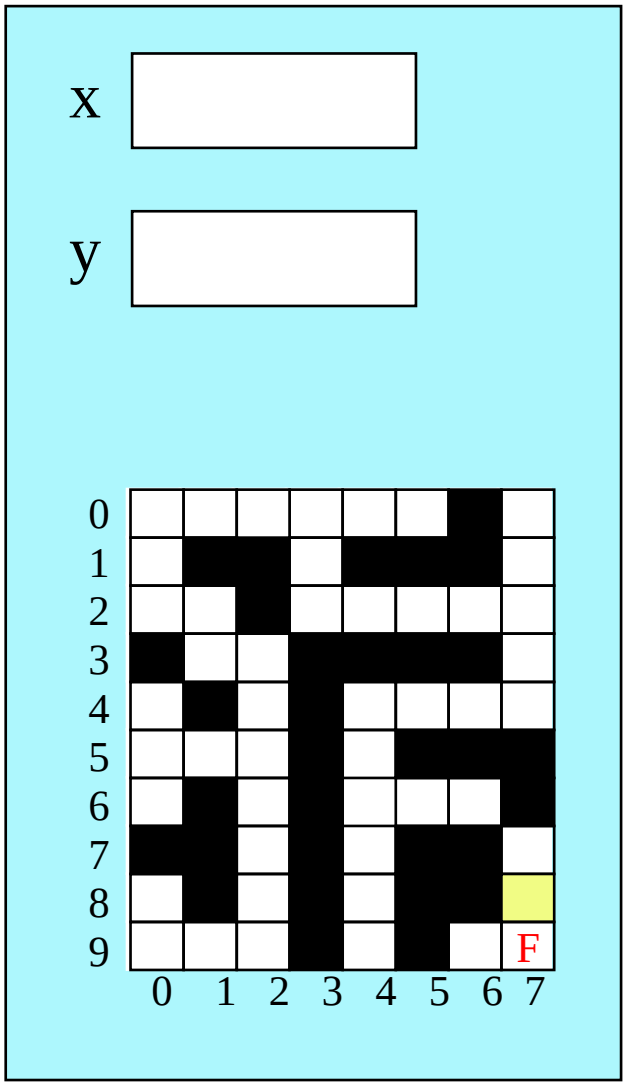
```
/* Returns true if <x,y> is a legal move
   given the maze, otherwise returns false */
int legal_mv (char m[MAXX ][MAXY], int x, int y)
{
  return(x >= 0 && x <= MAXX && y >= 0 &&
         y <= MAXY && m[x][y] != 'X');
}
```

Một Giải Pháp Tao Nhã

```
/* Returns true if there is a path from <x,y> to an
   element of maze containing 'F' otherwise returns
   false */
int is_path (char m[MAXX][MAXY ], int x, int y) {
  if (m [x][y] == 'F')
    return(TRUE);
  else {
    m[x][y] = 'X';
    return((legal_mv(m,x+1,y) && is_path(m,x+1,y)) ||
           (legal_mv(m,x-1,y) && is_path(m,x-1,y)) ||
           (legal_mv(m,x,y-1) && is_path(m,x,y-1)) ||
           (legal_mv(m,x,y+1) && is_path(m,x,y+1)))
  }
}
```

Ví Dụ

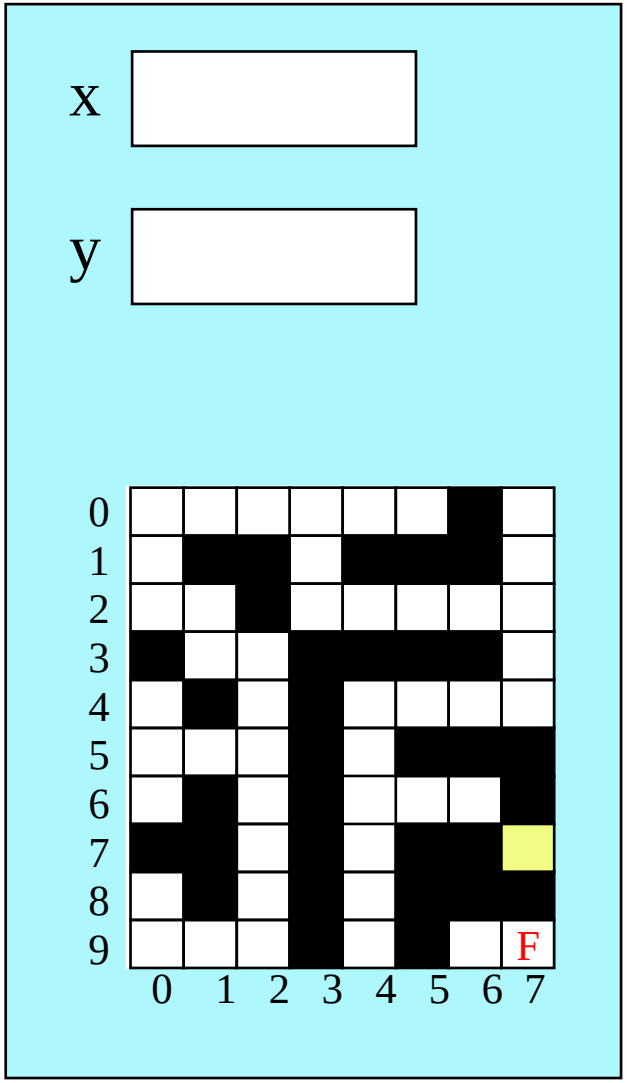
```
is_path(maze, 7, 8)
```



Ví Dụ

```
is_path(maze, 7, 8)
```

```
is_path(maze, 7, 8)
```

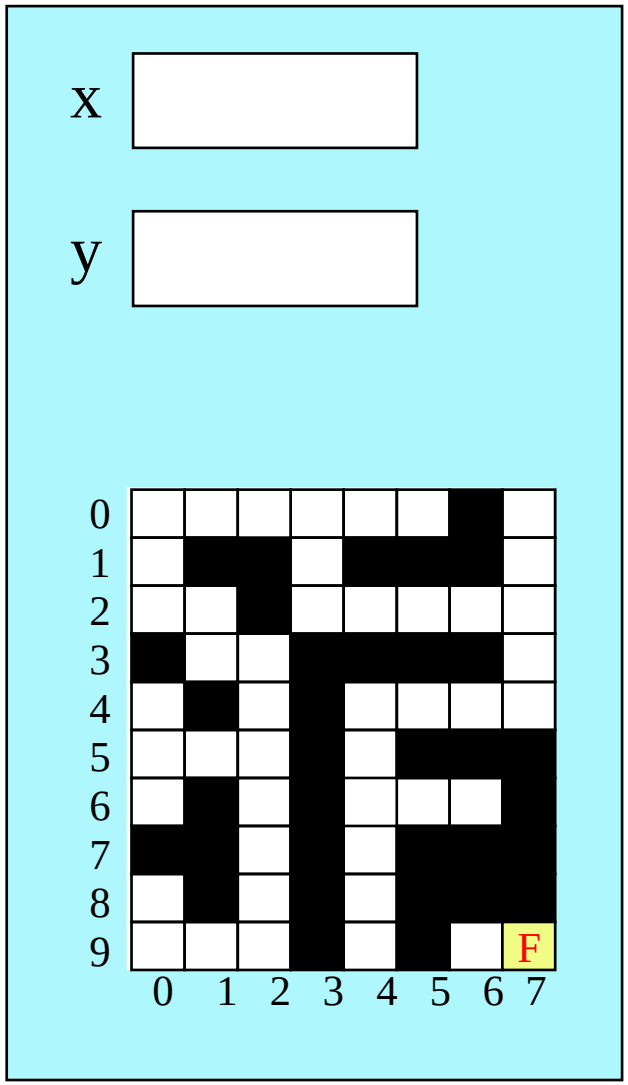


Ví Dụ

```
is_path(maze, 7, 8)
```

```
is_path(maze, 7, 8)
```

```
is_path(maze, 7, 8)
```



Tóm Lược Về Đệ Quy

- Đệ quy là một trong các kỹ thuật lập trình:
  - Nó hoạt động bởi vì cách hàm được gọi và cách các biến cục bộ hoạt động trong C
  - Mỗi khi hàm được gọi, tất cả mọi thứ đều có một bản sao mới
- Đệ quy không chỉ là một kỹ thuật lập trình, nó còn là một cách tiếp cận trong việc giải quyết các vấn đề.
- Cần thời gian và công sức để trở nên thành thạo kỹ thuật này.
- Đệ quy là cách tiếp cận tự nhiên đối với rất nhiều cấu trúc dữ liệu.