Chứng minh đệ quy (Induction) là một kỹ thuật toán học dùng để chứng minh các mệnh đề cho các đối tượng đệ quy như dãy số, cây, hoặc các cấu trúc tương tự. Còn đệ quy (recursion) là một kỹ thuật hoặc phương pháp giải quyết vấn đề được sử dụng trong toán học và khoa học máy tính.

Chapter 4
Induction and Recursion
Quy nạp và Đệ quy

Cả hai đều có một bước cơ sở và một bước suy diễn. (dựa trên nguyên tắc liên tục chia vấn đề thành các phần nhỏ hơn)

đệ quy là cách triển khai quy nạp trong lập trình vì:
-Cả hai đều có bước cơ sở và bước suy diễn.
-Cả hai đều sử dụng tính chất của bước trước để suy ra bước sau.
-Khi viết một hàm đệ quy, nếu chứng minh được bằng quy nạp toán học, thì ta có thể đảm bảo nó hoạt động đúng.

Objectives

- Mathematical Induction
- Strong Induction and Well-Ordering
- Recursive Definitions and Structural Induction
- Recursive Algorithms
- Program Correctness

4.1- Mathematical Induction

- Introduction
- Mathematical Induction
- Examples of Proofs by Mathematical Induction

Principle of Mathematical Induction

<u>Principle of Mathematical Induction</u>

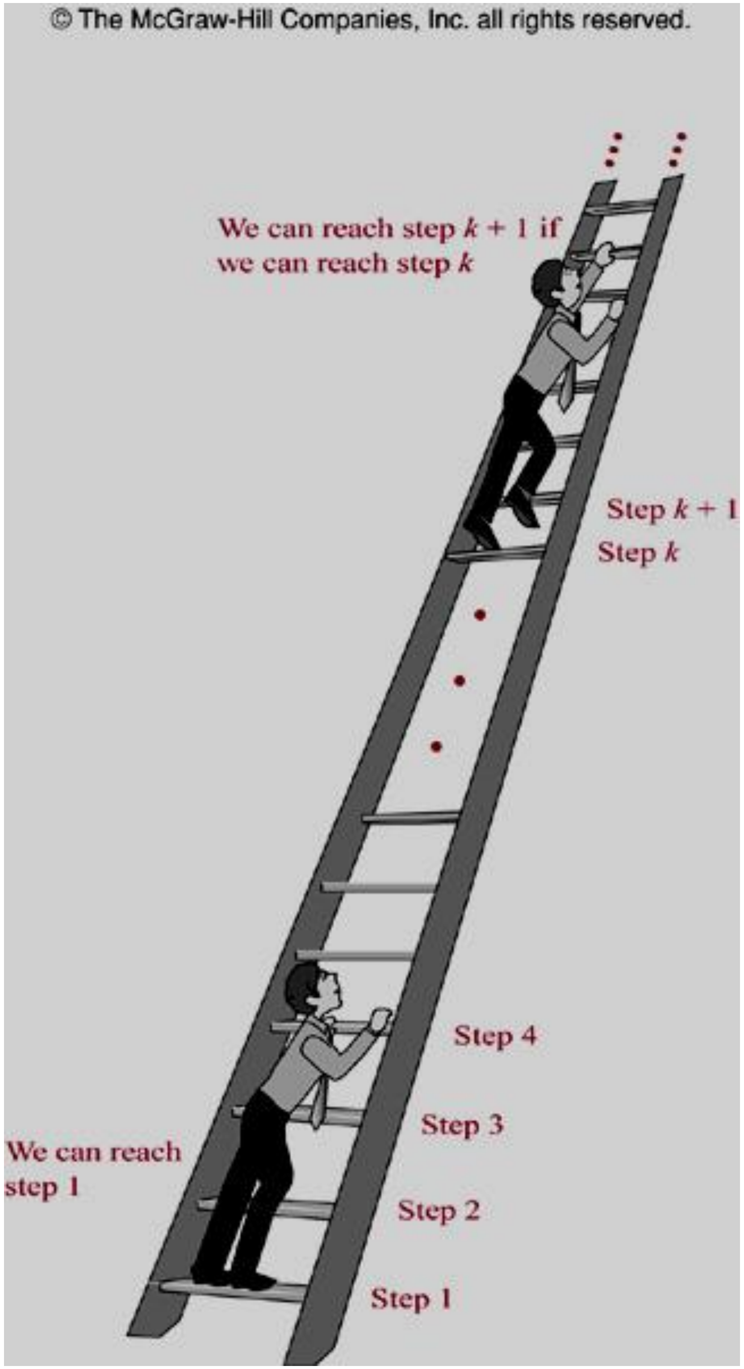
To prove P(n) is true for all possible integers n, where P(n) is a propositional function, we complete two step:

Basic step:

Verifying P(1) is true

Inductive step:

Show $P(k) \rightarrow P(k+1)$ is true for all $k>0$



Induction: Example 1

Prove that $1 + 2 + 3 + \ldots + n = n(n+1)/2$ for all integers $n>0$

Solution.

Let P(n)=" $1+2+3+\ldots+ n = n(n+1)/2$".

- Basic step: P(1) = "$1 = 1(1+1)/2$" → true
- Inductive step: With arbitrary $k>0$,

P(k) =" $1+ 2+\ldots+ k = k(k+1)/2$" is true.

We have

$$\underline{1+2+3+\ldots+k}+(k+1)= k(k+1)/2 + (k+1)$$
$$= [k(k+1)+ 2(k+1)]/2$$
$$= (k+1)(k+2)/2$$
$$= (k+1)((k+1)+1)/2$$

P(k+1)="$1+2+3+\ldots+k+1 = (k+1)(k+2)/2$" is true.

P(k) → P(k+1): true

Proved.

Example 2 p.316

- Conjecture a formula for the sum of the first n positive odd integers. Then prove your conjecture using mathematical induction.
- *Solution.*

  The sum of the first n positive odd integers for n=1, 2, 3, 4, 5 are:

  $1=1$, $1+3=4$, $1+3+5=9$,

  $1+3+5+7=16$, $1+3+5+7+9=25$.
- *Conjecture*: **$1+3+5+\ldots+(2n-1)=n^2$**.
- *Proof*. Let P(n)="**$1+3+5+\ldots+(2n-1)=n^2$**."

  - Basic step. P(1)="1=1" is true.

  - Inductive step. $(P(k) \rightarrow P(k+1))$ is true.

  Suppose P(k) is true. That is, "$1+3+5+\ldots+(2k-1)=k^2$"

  We have, $\underline{1+3+5+\ldots+(2k-1)}+(2k+1)=\underline{k^2}+2k+1 = (k+1)^2$.

  So, P(k+1) is true.

Proved.

Induction: Examples 2..13 – pages: 268..278

- $1+3+5+\ldots+(2n-1) = n^2$
- $2^0+2^1+2^2+2^3+\ldots+2^n = \sum 2^n = 2^{n+1}-1$
- $\sum ar^j = a + ar + ar^2 + \ldots + ar^n = (ar^{n+1}-a)/(r-1)$
- $n < 2^n$
- $2^n < n!$, $n>3$
- n^3-n is divisible by 3, n is positive integer
- The number of subsets of a finite set: a set with n elements has 2^n subsets.
- ....
- Let $H(j) = 1/1 + ½ + 1/3 + \ldots + 1/j$

  Prove that $H(2^n) \geq 1 + n/2$ for all $n \geq 0$

4.2- Strong Induction and Well-Ordering

Principle of Strong Induction

To prove P(n) is true for all positive integers n, where P(n) is a propositional function, two steps are performed:

Basic step:

Verifying P(1) is true

Inductive step:

Show $[P(1) \wedge P(2) \wedge \ldots \wedge P(k)] \rightarrow P(k+1)$ is true for all $k>0$

Bước quy nạp: Giả sử mệnh đề đúng với tất cả các giá trị từ 0 đến n (giả thuyết quy nạp mạnh), sau đó chứng minh nó đúng với n+1.

Strong Induction: Example 1

Prove that if n is an integer greater than 1, then n can be written as the product of primes

P(n) : n can be written as the product of primes

Basic steps: P(2) = true // 2=2 , product of 1 primes

P(4) = true // 4=2.2

Inductive step:

Assumption: P(j)=true for all positive j ≤ k

- Case k+1 is a prime ➔ P(k+1) =true
- Case k+1 is a composite ➔ k+1= ab, 2 ≤ a ≤ b<k+1

➔ P(k) is true

Well-Ordering

The validity of the Principle of Mathematical Induction follows from the Well-Ordering property of the set of non-negative integers.

Well-Ordering

Any nonempty set of non-negative integers has a least element.

Mọi tập hợp con không rỗng của tập số nguyên không âm N đều có phần tử nhỏ nhất.

WOP thường được dùng theo kiểu chứng minh phản chứng :

Giả sử ta muốn chứng minh một mệnh đề đúng với mọi $n \geq 0$.
Giả sử ngược lại rằng tập hợp S của tất cả các số mà mệnh đề sai là không rỗng.
Theo WOP, tồn tại số nhỏ nhất n_0 trong S. Nếu $n_0=0$, ta mâu thuẫn với bước cơ sở của quy nạp.
Nếu $n_0 > 0$, theo giả thuyết quy nạp, mệnh đề đúng với $n_0 - 1$ nhưng điều đó lại suy ra mệnh đề cũng đúng với n_0 , mâu thuẫn!
Do đó, tập S phải rỗng, nghĩa là mệnh đề đúng với mọi n. (quy nạp đúng)

4.3- Recursive Definition and Structural Induction

- Introduction
- Recursively Defined Functions
- Recursively Defined Sets and Structures
- Structural Induction
- Generalized Induction
- Recursive Algorithms

Recursion: Introduction

- Objects/ functions may be difficultly defined.
- Define an object/function in terms of itself
- Examples:

1. Định nghĩa Số tự nhiên:
   - 0 là một số tự nhiên
   - n > 0 là số tự nhiên nếu n – 1 là số tự nhiên
2. Định nghĩa n giai thừa
   - 0! = 1
   - Nếu n >0 thì n! = n* (n-1)!

Recursion: Introduction

- Objects/ functions may be difficultly defined.
- Define an object/function in terms of itself
- Examples:

$$2^n = \begin{cases} 1, n = 0 \\ 2.2^{n-1}, n > 0 \end{cases}$$

$$\sum_{i=0}^{n} i = \begin{cases} 0, n = 0 \\ n + \sum_{i=0}^{n-1} i, n > 0 \end{cases}$$

```
#include <stdio.h>
#include <conio.h>
double compute2n( int n)
{ if (n==0) return 1;
  return 2*compute2n(n-1);
}
void main()
{ clrscr();
  int n;
  scanf("%d",&n);
  double r=compute2n(n);
  printf("%lf",r);
  getch();
}
```

Recursively Defined Functions

- Recursive function

  Two steps to define a function with the set of nonnegative integers as its domain:
- Basis step: Specify the value of the function at zero.
- Recursive step: Give a rule for finding its value at an integer from its values at smaller integers
- Example: Find f(1), f(2), f(4),f(6) of the following function:

$$f(n) = \begin{cases} n, n < 3 \\ 3n + f(n-1), n \geq 3 \end{cases}$$

Recursively Defined Functions

- Example: Give the recursive definition of $\sum a_i$, i=0..k
- Basis step: $\sum a_i = a_0$, i=0
- Inductive step:

$$\boxed{a_0 + a_1 + \dots + a_{k-1}} + a_k$$

$$\left(\sum a_i ,\ i=0..k-1 \right)$$

$$\sum a_i = a_k + \left(\sum a_i ,\ i=0..k-1 \right)$$

Recursively Defined Functions

Definition 1: Fibonacci numbers $f(n)=\begin{cases} 1, n=0,1 \\ f(n-1)+f(n-2), n>1 \end{cases}$

Recursively Defined Functions

Definition 1: Fibonacci numbers

$$f(n)=\begin{cases} 1, n=0,1 \\ f(n-1)+f(n-2), n>1 \end{cases}$$

Recursively Defined Sets and Structures

Recursively Defined Structures (Binary Trees)

Recursively Defined Set (Strings)

- Example S= { 3,6,9,12,15, 18,21,…}

Step 1: $3 \in S$

Step 2: If $x \in S$ and $y \in S$ then $x+y \in S$

- **Definition 2**: The set $\sum^*$ of string over alphabet $\sum$ can be defined recursively by:

Basis step: $\lambda \in \sum^*$, λ is the empty string with no symbols

Recursive step: If $w \in \sum^*$ and $x \in \sum$ then $wx \in \sum^*$

Example: $\sum = \{0,1\}$ → $\sum^*$ is the set of string made by 0 and 1 with arbitrary length and arbitrary order of symbols 0 and 1

Recursively Defined Sets and Structures

- **Definition 3**: String Concatenation

Basis step: If $w \in \Sigma^*$ then $w.\lambda=w$, λ is the empty string

Recursive step: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$ then $w_1.(w_2x) = (w_1.w_2)x$

Example: $\Sigma = \{0,1\}$ ➔ Σ^* is the set of string made by 0 and 1 with arbitrary length and arbitrary order of symbols 0 and 1

4.4- Recursive Algorithms

- **Definition 1**: An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.

Example: Recursive algorithm for computing n!

procedure factorial (n: nonnegative integer)
if n=0 **then** factorial(n) :=1
else factorial(n) = n.factorial(n-1)

n!= 1 , n=0
n!= 1.2.3.4…n = n.(n-1)!, n>0

Recursive Algorithms…

Example: Recursive algorithm for computing a^n

procedure power (a: nonzero real number
n: nonnegative integer)
if n=0 **then** power(a,n) :=1
else power(a,n)=a.power(a,n-1)

$a^n = 1$, n=0
$a^n = a.a.a...a = a.a^{n-1}$, n>0

Recursive Algorithms…

Example: Recursive algorithm for computing $b^n \bmod m$
$m \geq 2$, $n \geq 0$, $1 \leq b < m$.

$b^n \bmod m = (b.(b^{n-1} \bmod m) \bmod m$

$b^0 \bmod m = 1$

Using division to improve performance: (n steps backward to 0 faster)

If n is even → $b^n = b^{n/2}.b^{n/2}$

→ $b^n \bmod m = ((b^{n/2} \bmod m). (b^{n/2} \bmod m)) \bmod m$

→ $b^n \bmod m = (b^{n/2} \bmod m)^2 \bmod m$

If n is odd→ $b^n = b.b^{\lfloor n/2 \rfloor}.b^{\lfloor n/2 \rfloor}$

→ $b^n \bmod m = ([(b^{\lfloor n/2 \rfloor} \bmod m)^2 \bmod m].(b \bmod m))\bmod m$

Algorithm: page 313

Recursive Algorithms…

Example: Recursive algorithm for computing b^n mod m

$m \geq 2$, $n \geq 0$, $1 \leq b < m$.

ALGORITHM 4 Recursive Modular Exponentiation.

procedure *mpower*(b, n, m: integers with $b > 0$ and $m \geq 2$, $n \geq 0$)
if $n = 0$ **then**
  return 1
else if n is even **then**
  return *mpower*$(b, n/2, m)^2$ **mod** m
else
  return (*mpower*$(b, \lfloor n/2 \rfloor, m)^2$ **mod** $m \cdot b$ **mod** m) **mod** m
{output is b^n **mod** m}

Recursive Algorithms…

Example: Recursive algorithm for computing gcd(a,b)

a,b: non negative integer, a < b

If a>b then swap a,b

gcd(a,b)=b , a=0

gcd (a,b) = gcd(b mod a, a)

Algorithm: page 313

Recursive Algorithms…

Example: Recursive algorithm for computing gcd(a,b)

a,b: non negative integer, a < b

ALGORITHM 3 A Recursive Algorithm for Computing gcd(a, b).

procedure $gcd(a, b$: nonnegative integers with $a < b$)
if $a = 0$ **then return** b
else return $gcd(b$ **mod** $a, a)$
{output is gcd(a, b)}

Recursive Algorithms…

Example: Recursive algorithm for the value x in the sequence

a_i, a_{i+1},…, a_j, sub-sequence of a_n.

$1 \le i \le n$, $1 \le j \le n$

$i>j \rightarrow$ location $=0$
$a_i=x \rightarrow$ location $= i$
location (i, j, x) = location (i+1, j, x)

Algorithm: page 363 – You should modify it.

Recursive Algorithms…

Example: Recursive algorithm for the value x in the sequence

a_i, a_{i+1},…, a_j, sub-sequence of a_n.

$1 \le i \le n$, $1 \le j \le n$

ALGORITHM 5 A Recursive Linear Search Algorithm.

procedure *search*(i, j, x: i, j, x integers, $1 \le i \le j \le n$)
if $a_i = x$ **then**
    return i
else if $i > j$ **then**
    return 0
else
    return *search*($i + 1, j, x$)
{output is the location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}

Recursive Algorithms…

Example: Recursive algorithm for binary searching the value x in the increasingly ordered sequence $a_i, a_{i+1}, \ldots, a_{j-1}$, sub-sequence of a_n. $1 \le i \le n$, $1 \le j \le n$

procedure binary-search(x, i, j)
if i>j **then** location=0
m= $\lfloor (i+j)/2 \rfloor$
if x= a_m **then** location =m
else if x< a_m **then** location= binary-search(x, i, m-1)
else location= binary-search(x, m+1, j)

Algorithm: page 363 – You should modify it.

Recursive Algorithms…

ALGORITHM 6 A Recursive Binary Search Algorithm.

procedure *binary search*$(i, j, x: i, j, x$ integers, $1 \le i \le j \le n)$
$m := \lfloor (i + j)/2 \rfloor$
if $x = a_m$ **then**
    return m
else if $(x < a_m$ and $i < m)$ **then**
    return *binary search*$(i, m - 1, x)$
else if $(x > a_m$ and $j > m)$ **then**
    return *binary search*$(m + 1, j, x)$
else return 0
{output is location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}

Proving Recursive Algorithms Correct

- Using mathematical induction.
- Example: prove the algorithm that computes n! is correct.

procedure f (n: nonnegative integer)
  if n=0 **then** f(n) :=1
  else f(n) = n.f(n-1)

If n=0, first step of the algorithm tells us f(0)=1 ➔ true
Assuming f(n) is true for all n≥0
f(n)= 1.2.3….(n)
(n+1).f(n)= 1.2.3…n.(n+1) = (n+1)!
f(n+1) = (n+1)!
Conclusion: f(n) is true for all integer n, n≥0

More examples: Page 315

Recursion and Iteration

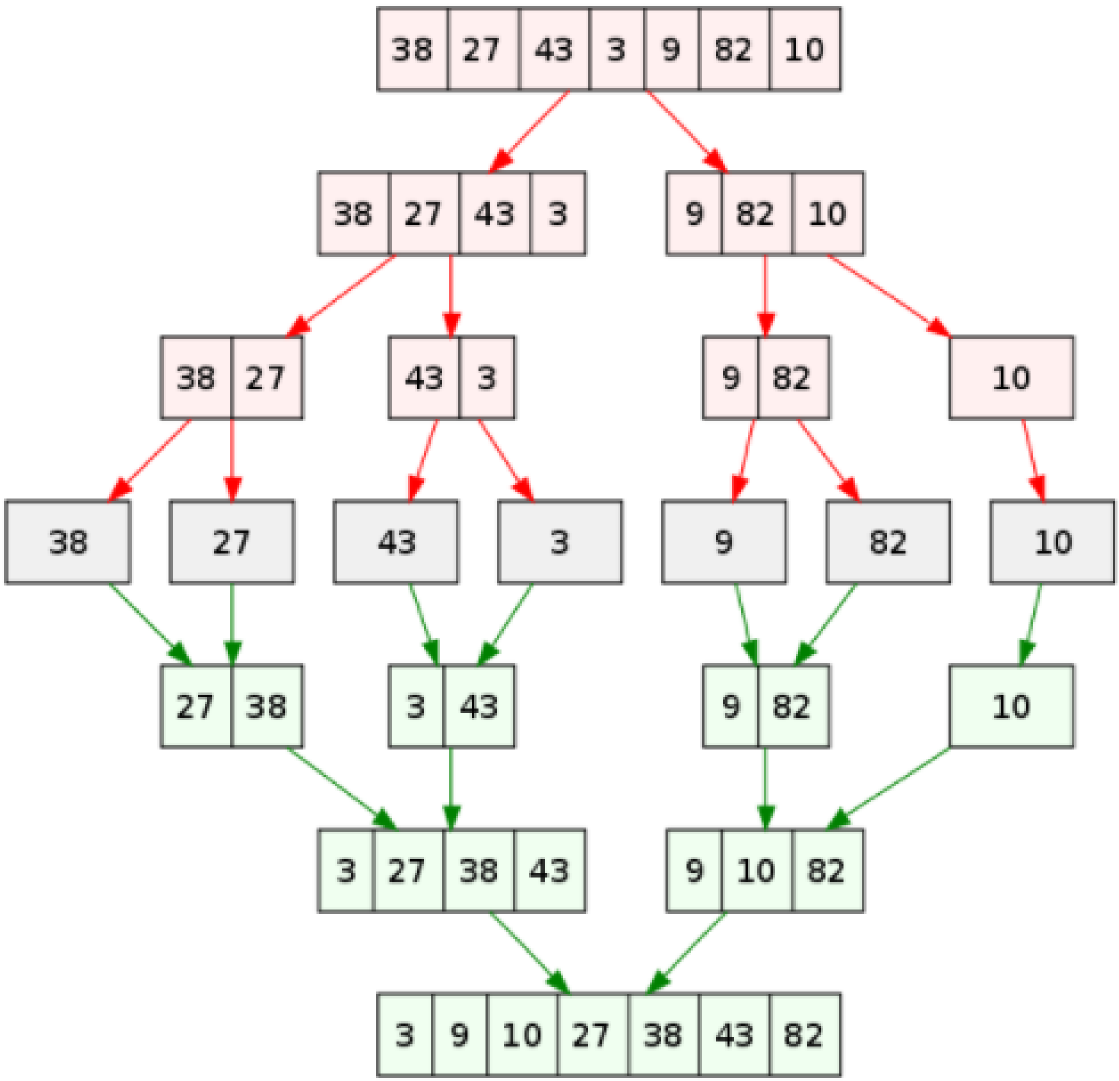
procedure rfibo (n: nonnegative integer)
If n=0 **then** rFibo(0)=0
Else if n=1 **then** rFibo(1)=1
Else rFibo(n) := rFibo(n-2) + rFibo(n-1)

procedure iFibo (n: nonnegative integer)
If n=0 **then** y:=0
Else if n=1 **then** y:=1
Else Begin
    x:=0 ; y:=1
    for i:= 2 **to** n
        Begin
            z:= x+y; x:= y; y:=z
        End
    End { iFibo(n) = z}

Recursive algorithm uses far more computation than iterative one

Merge Sort

Sơ đồ tiến trình của thuật toán merge sort cho mảng {38, 27, 43, 3, 9, 82, 10}.



Minh họa thuật toán sắp xếp merge sort

Merge Sort

```
mergeSort(arr[], l,  r)
If r > l
     1. Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa:
             middle m = (l+r)/2
     2. Gọi đệ quy hàm mergeSort cho nửa đầu tiên:
             mergeSort(arr, l, m)
     3. Gọi đệ quy hàm mergeSort cho nửa thứ hai:
             mergeSort(arr, m+1, r)
     4. Gộp 2 nửa mảng đã sắp xếp ở (2) và (3):
             merge(arr, l, m, r)
```

Mỗi lần gọi hàm merge() sẽ tạo ra hai mảng tạm (L[] và R[]), mỗi mảng chứa một phần của mảng ban đầu.
(đó là lí do tại sao có space complexity là O(n))

Merge Sort

Procedure mergesort $(L = a_1, a_2, \ldots, a_n)$

if $n > 1$ **then**

$m := \lfloor n/2 \rfloor$

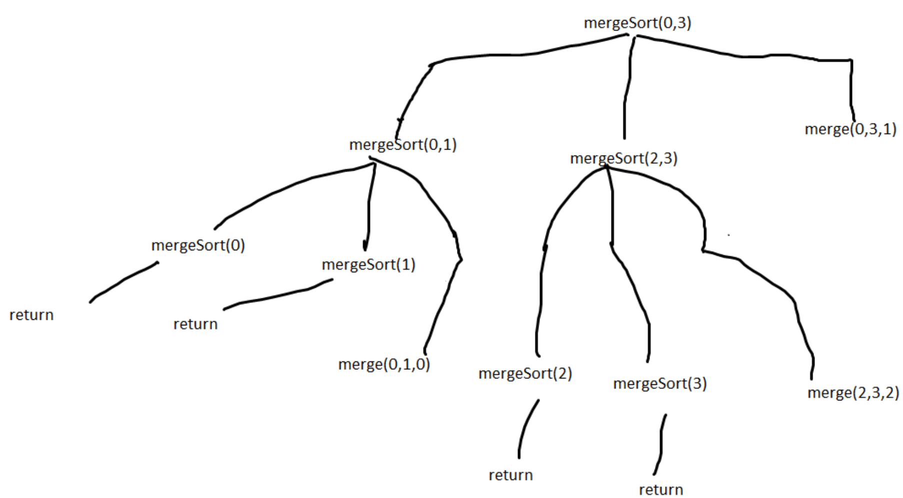
$L_1 = a_1, a_2, \ldots, a_m$

$L_2 = a_{m+1}, a_{m+2}, \ldots, a_n$

$L := merge(mergesort(L_1), mergersort(L_2))$

Print (L)

vd: sort mảng {38 27 43 10}



Merge Sort

Procedure mergesort $(L = a_1, a_2, \ldots, a_n)$
if $n > 1$ **then**
$m := \lfloor n/2 \rfloor$
$L_1 = a_1, a_2, \ldots, a_m$
$L_2 = a_{m+1}, a_{m+2}, \ldots, a_n$
$L := merge(mergesort(L_1), mergersort(L_2))$
Print (L)

Theorem

The number of comparisons needed to merge sort a list of n elements is $O(n \log n)$.

Fpt University

Thanks