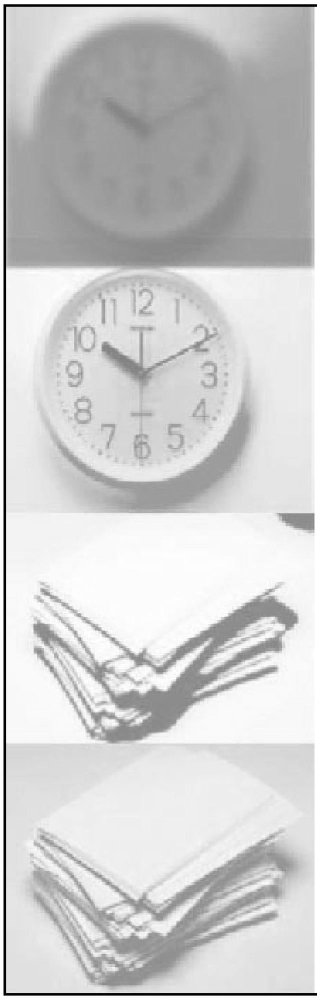
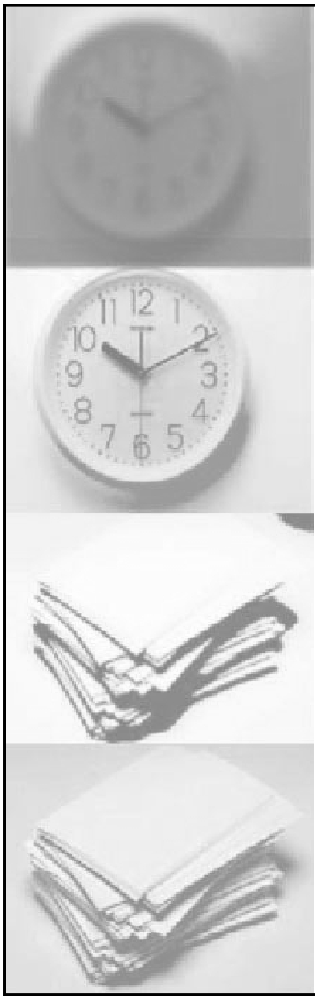

PHÂN TÍCH GIẢI
THUẬT



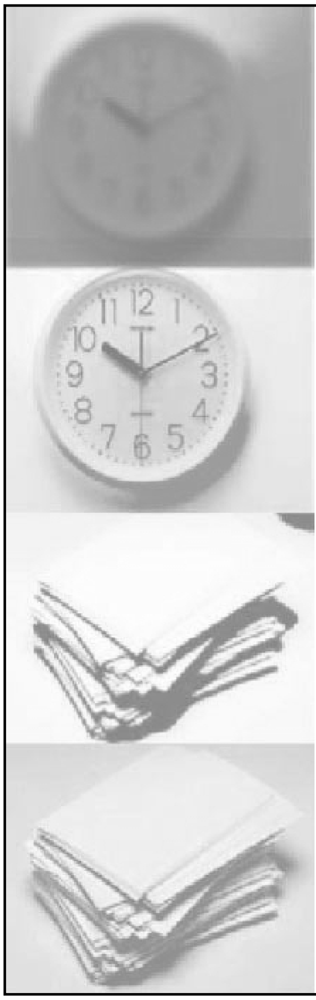
Mục tiêu

- Sau khi hoàn tất bài học này bạn cần:
  - Hiểu được sự cần thiết phải phân tích đánh giá giải thuật.
  - Biết các tiêu chuẩn để đánh giá một giải thuật.
  - Hiểu khái niệm độ phức tạp của giải thuật.
  - Vận dụng được các quy tắc để tính độ phức tạp của chương trình không gọi chương trình con, độ phức tạp của một chương trình có gọi các chương trình con không đệ quy.
  - Vận dụng được phương pháp thành lập phương trình đệ quy.



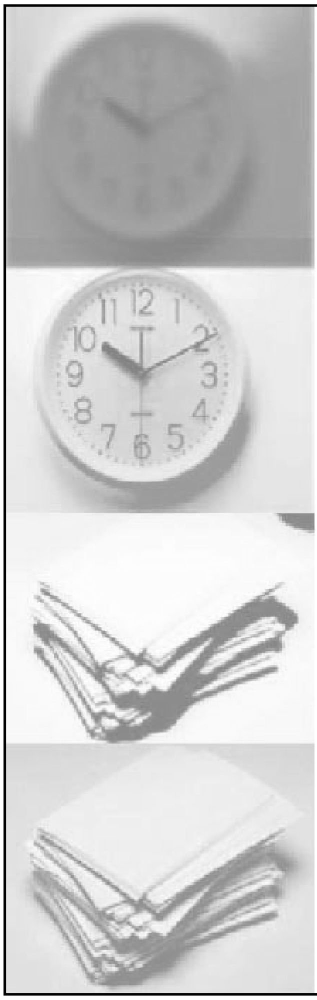
Mục tiêu (tt)

- Vận dụng được phương pháp truy hồi để giải phương trình đệ quy.
- Biết phương pháp đoán nghiệm để giải phương trình đệ quy.
- Vận dụng được việc giải phương trình đệ quy thuộc dạng phương trình tổng quát.
- Tổng hợp được vấn đề đánh giá giải thuật.



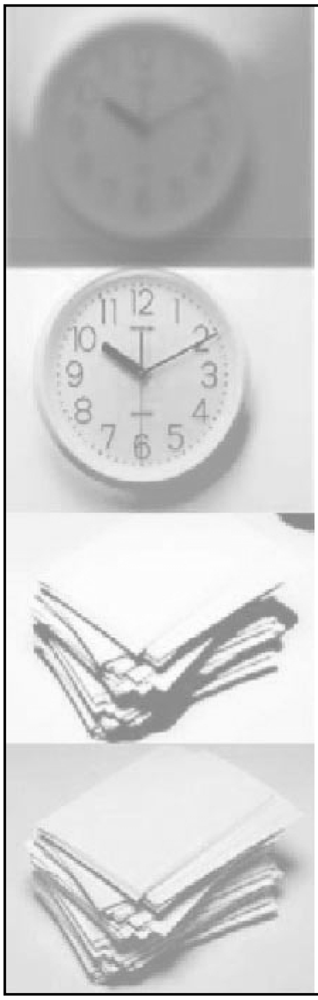
Sự cần thiết phải phân tích, đánh giá giải thuật

- Cần phải phân tích, đánh giá giải thuật để:
  - Lựa chọn một giải thuật tốt nhất trong các giải thuật để cài đặt chương trình giải quyết bài toán đặt ra.
  - Cải tiến giải thuật hiện có để được một giải thuật tốt hơn.



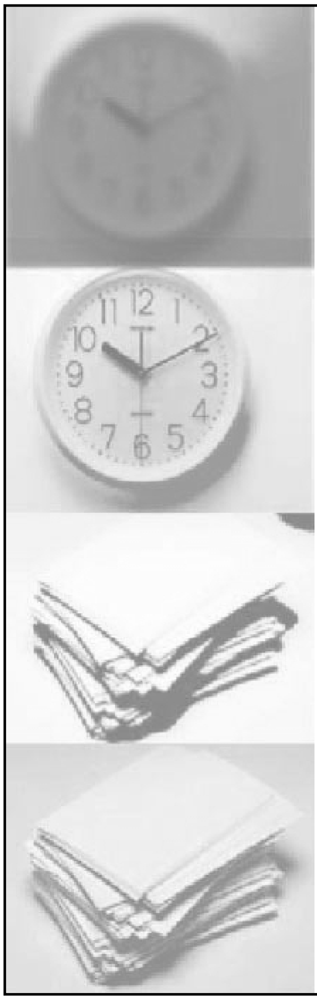
Tiêu chuẩn đánh giá một giải thuật là tốt

- Một giải thuật được xem là tốt nếu nó đạt các tiêu chuẩn sau:
  - Thực hiện đúng.
  - Tốn ít bộ nhớ.
  - Thực hiện nhanh.
- Trong khuôn khổ môn học này, chúng ta chỉ quan tâm đến tiêu chuẩn thực hiện nhanh.



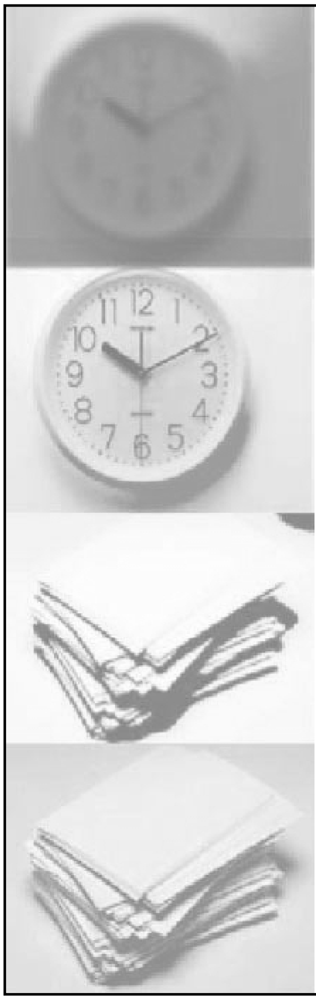
Thời gian thực hiện của chương trình

- Thời gian thực hiện một chương trình là một hàm của kích thước dữ liệu vào, ký hiệu T(n) trong đó n là kích thước (độ lớn) của dữ liệu vào.
- **Ví dụ :** Chương trình tính tổng của n số có thời gian thực hiện là T(n) = cn trong đó c là một hằng số.
- Thời gian thực hiện chương trình là một hàm không âm, tức là $T(n) \geq 0 \ \forall n \geq 0$.



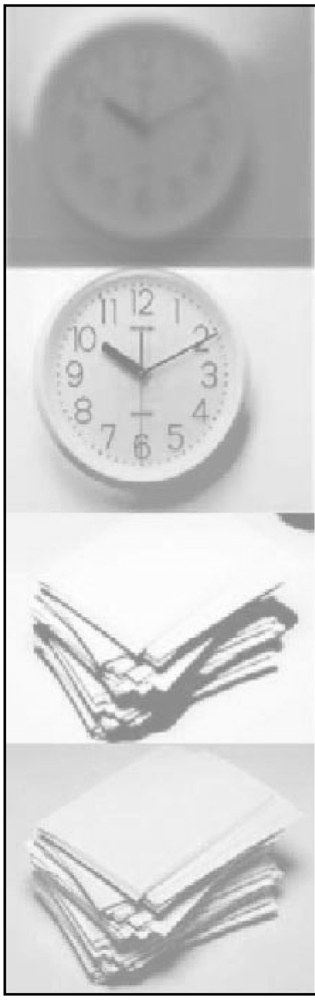
Đơn vị đo thời gian thực hiện

- Đơn vị của T(n) không phải là đơn vị đo thời gian bình thường như giờ, phút giây... mà thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng.
- **Ví dụ**: Khi ta nói thời gian thực hiện của một chương trình là T(n) = Cn thì có nghĩa là chương trình ấy cần Cn chỉ thị thực thi.



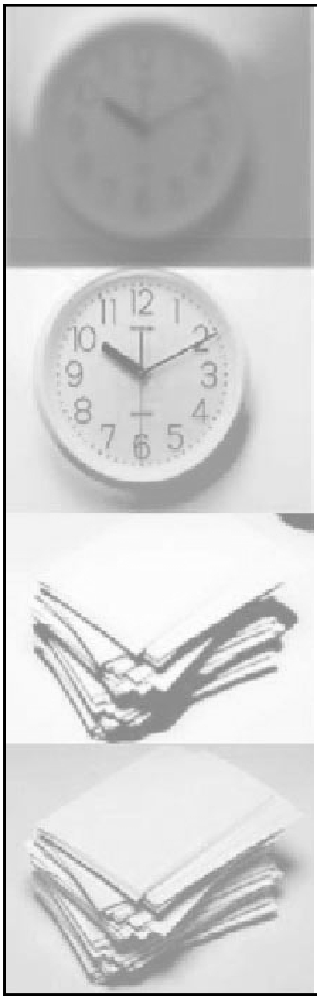
Thời gian thực hiện trong trường hợp xấu nhất

- Nói chung thì thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào.
- Vì vậy thường ta coi T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thước n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n.



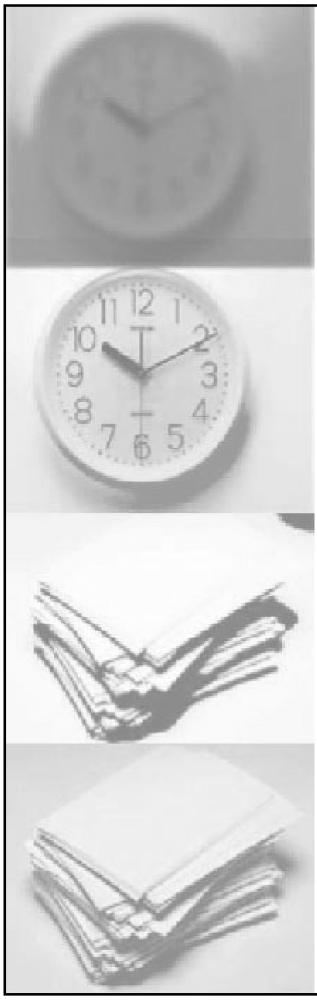
Tỷ suất tăng

- Ta nói rằng hàm không âm T(n) có tỷ suất tăng (growth rate) f(n) nếu tồn tại các hằng số C và N0 sao cho $T(n) \leq Cf(n)$ với mọi $n \geq N0$.
- Ta có thể chứng minh được rằng “Cho một hàm không âm T(n) bất kỳ, ta luôn tìm được tỷ suất tăng f(n) của nó”.



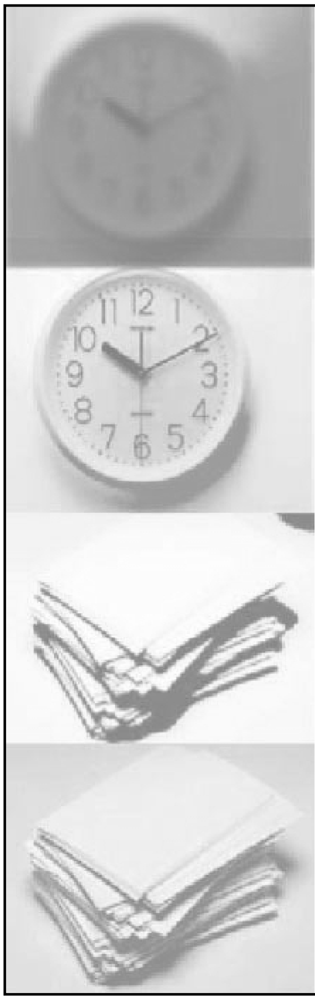
Tỷ suất tăng (tt)

- **Ví dụ 1:** Giả sử T(0) = 1, T(1) = 4 và tổng quát $T(n) = (n+1)^2$. Đặt N0 = 1 và C = 4 thì với mọi n ≥1 chúng ta dễ dàng chứng minh được rằng $T(n) = (n+1)^2 \leq 4n^2$ với mọi $n \geq 1$, tức là tỷ suất tăng của T(n) là n^2.
- **Ví dụ 2:** Tỷ suất tăng của hàm $T(n) = 3n^3 + 2n^2$ là n^3. Thực vậy, cho N0 = 0 và C = 5 ta dễ dàng chứng minh rằng với mọi $n \geq 0$ thì $3n^3 + 2n^2 \leq 5n^3$



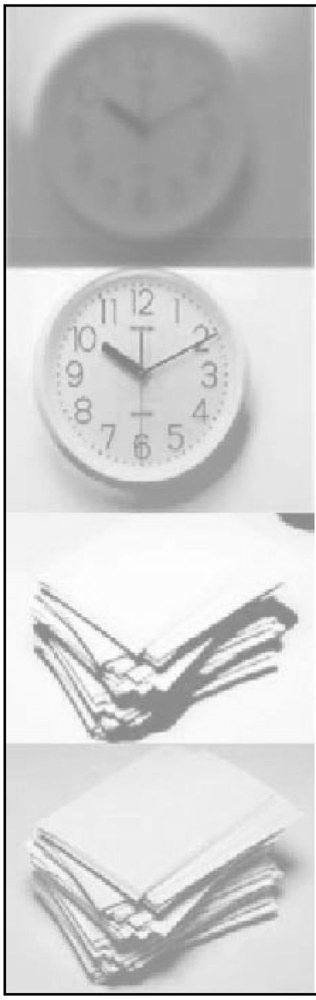
Khái niệm độ phức tạp của giải thuật

- Giả sử ta có hai giải thuật P1 và P2 với thời gian thực hiện tương ứng là T1(n) = $100n^2$ (với tỷ suất tăng là n^2) và T2(n) = $5n^3$ (với tỷ suất tăng là n^3).
- Khi n>20 thì T1 < T2. Sở dĩ như vậy là do tỷ suất tăng của T1 nhỏ hơn tỷ suất tăng của T2.
- Như vậy một cách hợp lý là ta xét tỷ suất tăng của hàm thời gian thực hiện chương trình thay vì xét chính bản thân thời gian thực hiện.
- Cho một hàm T(n), T(n) gọi là có độ phức tạp f(n) nếu tồn tại các hằng C, N0 sao cho $T(n) \le Cf(n)$ với mọi $n \ge N0$ (tức là T(n) có tỷ suất tăng là f(n)) và kí hiệu T(n) là O(f(n)) (đọc là “ô của f(n)”).



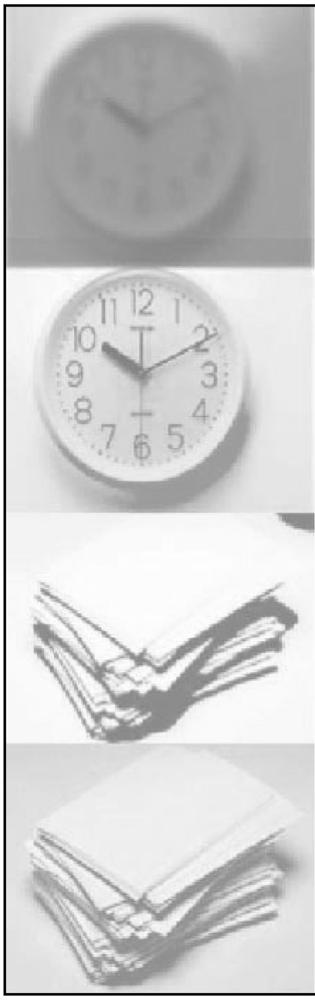
Khái niệm độ phức tạp của giải thuật (tt)

- Chú ý: O(C.f(n))=O(f(n)) với C là hằng số. Đặc biệt O(C)=O(1)
- Các hàm thể hiện độ phức tạp có các dạng thường gặp sau: **$\log_2 n$, n, $n\log_2 n$, n^2, n^3, 2^n, n!, n^n**.
- Ba hàm cuối cùng ta gọi là dạng hàm mũ, các hàm khác gọi là hàm đa thức.
- Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được, còn các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.
- Trong cách viết, ta thường dùng logn thay thế cho $\log_2 n$ cho gọn.



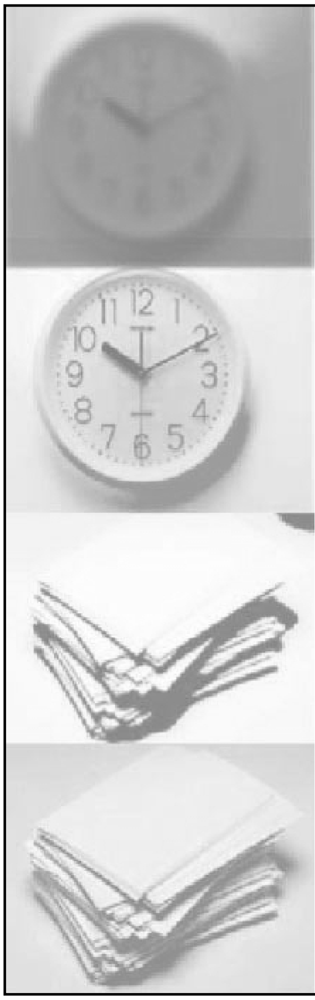
Phương pháp tính độ phức tạp

- Chúng ta sẽ nói đến phương pháp tính độ phức tạp (thời gian thực hiện) của:
  - Chương trình không gọi chương trình con.
  - Chương trình có gọi chương trình con không đệ quy.
  - Chương trình đệ quy
- Trước hết ta có hai quy tắc quan trọng là quy tắc cộng và quy tắc nhân
- **Quy tắc cộng**: Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O(f(n)), T2(n)=O(g(n)) thì thời gian thực hiện của đoạn hai chương trình đó **nối tiếp nhau** là T(n)=O(max(f(n),g(n))).
- **Quy tắc nhân**: Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1và P2 và T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện của đoạn hai đoạn chương trình đó **lồng nhau** là T(n) = O(f(n).g(n)).



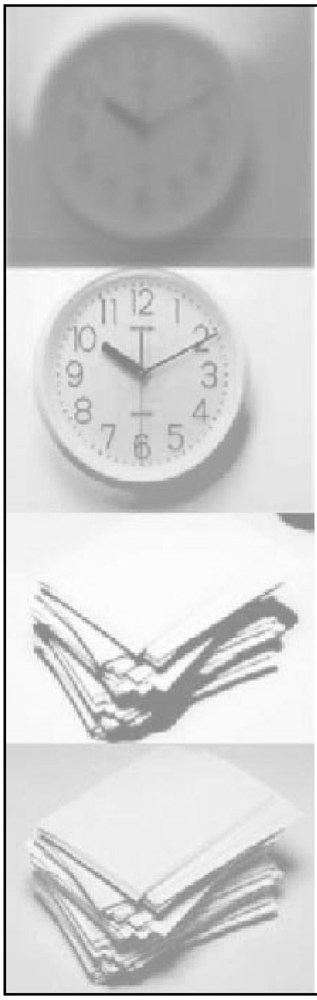
Qui tắc tổng quát để phân tích một chương trình không có chương trình con

- Thời gian thực hiện của mỗi lệnh gán, READ, WRITE là O(1)
- Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi các lệnh.
- Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau THEN hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1).
- Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.



Ví dụ 1:
Thủ tục sắp xếp “nổi bọt”

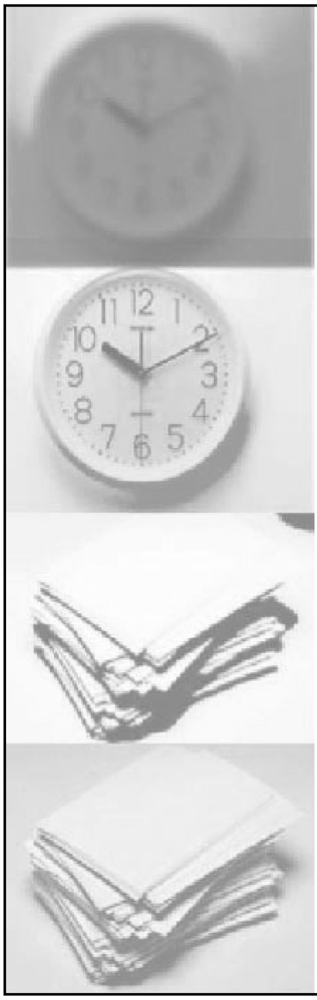
```
void BubbleSort(int a[n])
{      int i,j,temp;
/*1*/   for(i= 0; i<=n-2; i++)
/*2*/     for(j=n-1; j>=i+1;j--)
/*3*/       if (a[j].key < a[j-1].key) {
/*4*/          temp=a[j-1];
/*5*/          a[j-1] = a[j];
/*6*/          a[j]    = temp;
              }
 }
```



Tính thời gian thực hiện của thủ tục sắp xếp "nổi bọt"

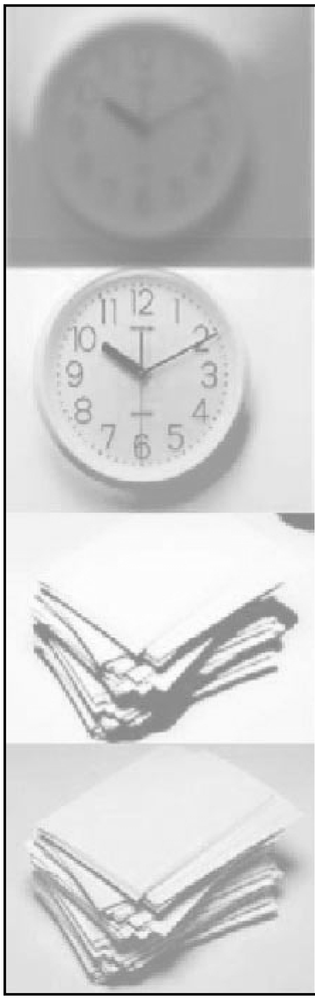
- Đây là chương trình sử dụng các vòng lặp xác định. Toàn bộ chương trình chỉ gồm một lệnh lặp {1}, lồng trong lệnh {1} là lệnh lặp {2}, lồng trong lệnh {2} là lệnh {3} và lồng trong lệnh {3} là 3 lệnh nối tiếp nhau {4}, {5} và {6}.
- Chúng ta sẽ tiến hành tính độ phức tạp theo thứ tự từ trong ra.
- Trước hết, cả ba lệnh gán {4}, {5} và {6} đều tốn O(1) thời gian, việc so sánh a[j-1] > a[j] cũng tốn O(1) thời gian, do đó lệnh {3} tốn O(1) thời gian.
- Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n-i).1) = O(n-i).
- Vòng lặp {1} có i chạy từ 1 đến n-1 nên thời gian thực hiện của vòng lặp {1} và cũng là độ phức tạp của giải thuật là

$$T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2)$$



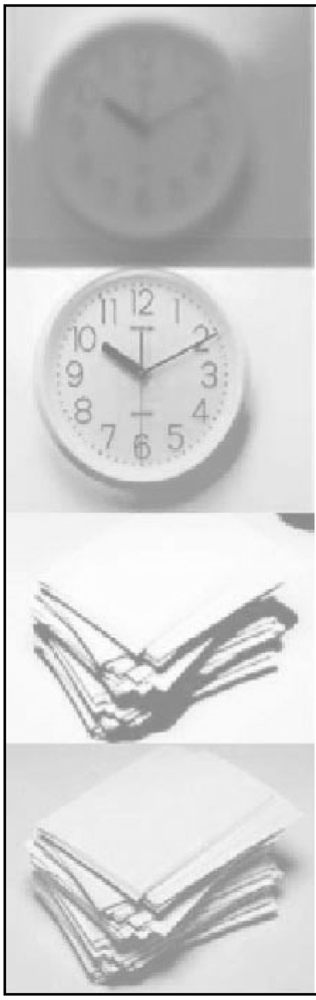
Tìm kiếm tuần tự

- Hàm tìm kiếm Search nhận vào một mảng a có n số nguyên và một số nguyên x, hàm sẽ trả về giá trị logic TRUE nếu tồn tại một phần tử a[i] = x, ngược lại hàm trả về FALSE.
- Giải thuật tìm kiếm tuần tự là lần lượt so sánh x với các phần tử của mảng a, bắt đầu từ a[1], nếu tồn tại a[i] = x thì dừng và trả về TRUE, ngược lại nếu tất cả các phần tử của a đều khác X thì trả về FALSE.



Tìm kiếm tuần tự (tt)

```
FUNCTION Search(a:ARRAY[1..n] OF
    Integer; x:Integer): Boolean;
VAR i:Integer; Found:Boolean;
BEGIN
{1}     i:=1;
{2}     Found:=FALSE;
{3}     WHILE(i<=n) AND (not Found) DO
{4}             IF A[i]=X THEN Found := TRUE
        ELSE I := i+1;
{5}     Search := Found;
END;
```

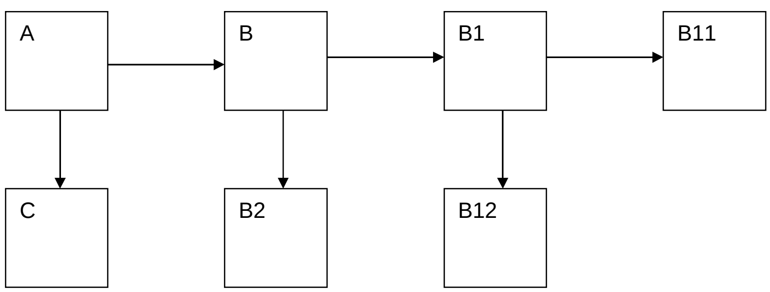


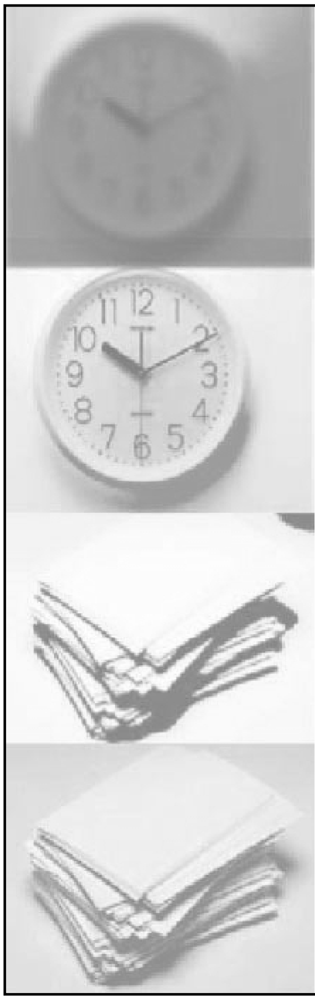
Tính độ phức tạp của hàm tìm kiếm tuần tự

- Ta thấy các lệnh {1}, {2}, {3} và {5} nối tiếp nhau, do đó độ phức tạp của hàm Search chính là độ phức tạp lớn nhất trong 4 lệnh này. Dễ dàng thấy rằng ba lệnh {1}, {2} và {5} đều có độ phức tạp O(1) do đó độ phức tạp của hàm Search chính là độ phức tạp của lệnh {3}. Lồng trong lệnh {3} là lệnh {4}. Lệnh {4} có độ phức tạp O(1).
- Lệnh {3} là một vòng lặp không xác định, nên ta không biết nó sẽ lặp bao nhiêu lần, nhưng trong trường hợp xấu nhất (tất cả các phần tử của mảng a đều khác x, ta phải xét hết tất cả các a[i], i có các giá trị từ 1 đến n) thì vòng lặp {3} thực hiện n lần, do đó lệnh {3} tốn O(n). Vậy ta có T(n) = O(n).

Độ phức tạp của chương trình có gọi chương trình con không đệ qui

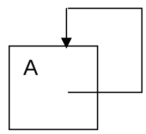
- Giả sử ta có một hệ thống các chương trình gọi nhau theo sơ đồ sau:



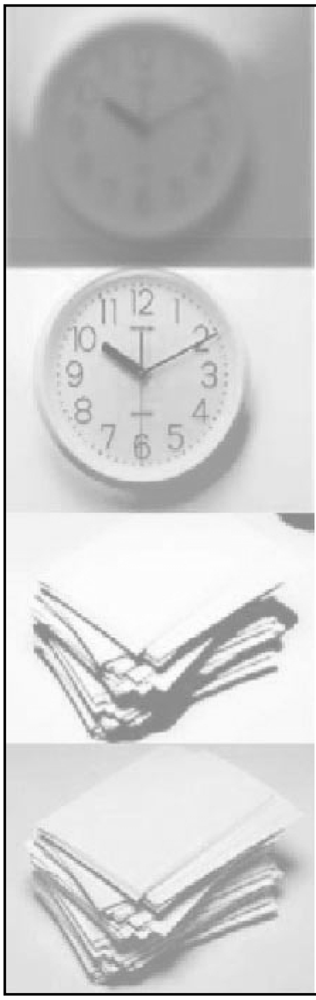


Phân tích
các chương trình đệ qui

- Có thể thấy hình ảnh chương trình đệ quy A như sau:



- Để phân tích các các chương trình đệ quy ta cần:
  - Thành lập phương trình đệ quy.
  - Giải phương trình đệ quy, nghiệm của phương trình đệ quy sẽ là thời gian thực hiện của chương trình đệ quy.

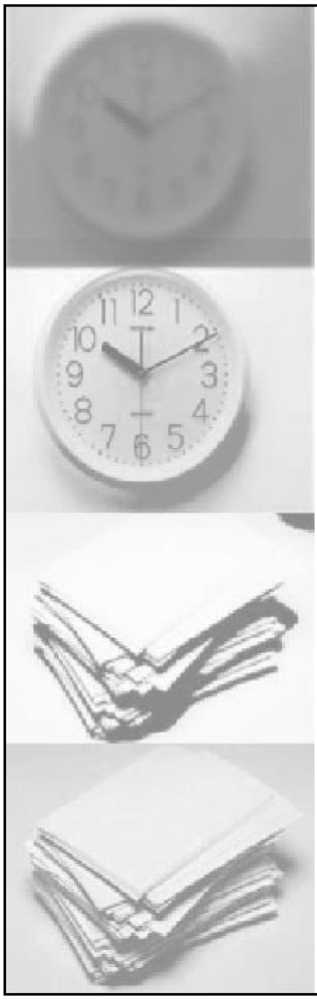


Chương trình đệ quy

- Chương trình đệ quy để giải bài toán kích thước n, phải có ít nhất một trường hợp dừng ứng với một n cụ thể và lời gọi đệ quy để giải bài toán kích thước k (k<n).
- **Ví dụ** : Chương trình đệ quy tính n!

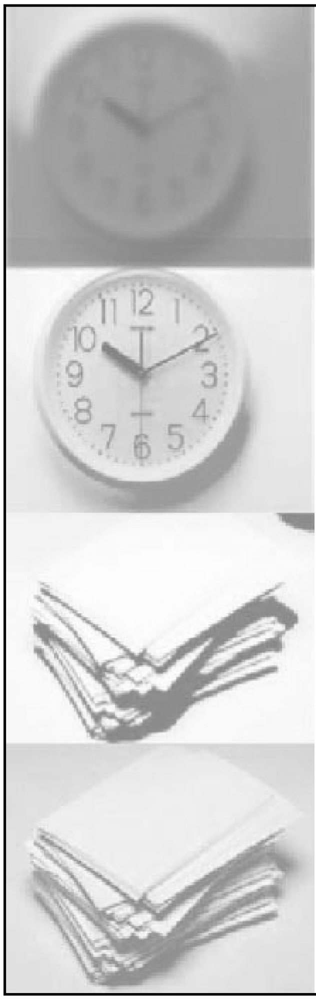
```
int Giai_thua(int n) {
    if (n==0) return 1;
    else return (n* Giai_thua(n-1));
};
```

- Trong ví dụ trên, n=0 là trường hợp dừng và k=n-1.



Thành lập phương trình đệ quy

- Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa T(n) và T(k), trong đó T(n) và T(k) là thời gian thực hiện chương trình có kích thước dữ liệu nhập tương ứng là n và k, với k < n.
- Để thành lập được phương trình đệ quy, ta phải căn cứ vào chương trình đệ quy.
- Ứng với trường hợp đệ quy dừng, ta phải xem xét khi đó chương trình làm gì và tốn hết bao nhiêu thời gian, chẳng hạn thời gian này là c(n).
- Khi đệ quy chưa dừng thì phải xét xem có bao nhiêu lời gọi đệ quy với kích thước k ta sẽ có bấy nhiêu T(k).
- Ngoài ra ta còn phải xem xét đến thời gian để phân chia bài toán và tổng hợp các lời giải, chẳng hạn thời gian này là d(n).

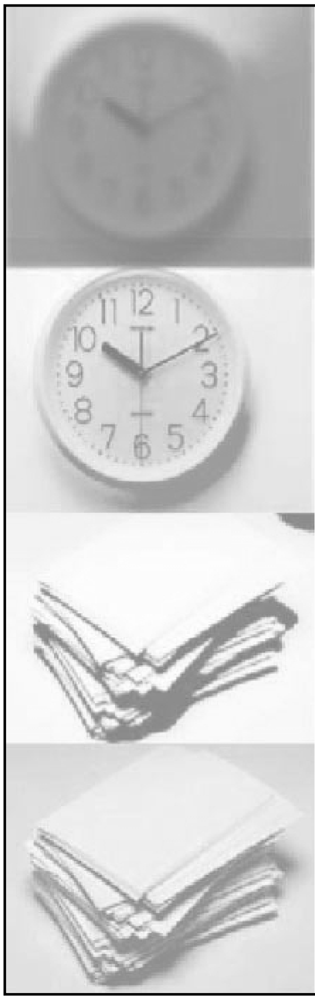


Thành lập phương trình đệ quy (tt)

- Dạng tổng quát của một phương trình đệ quy sẽ là:

$$T(n) = \begin{cases} C(n) \\ F(T(k)) + d(n) \end{cases}$$

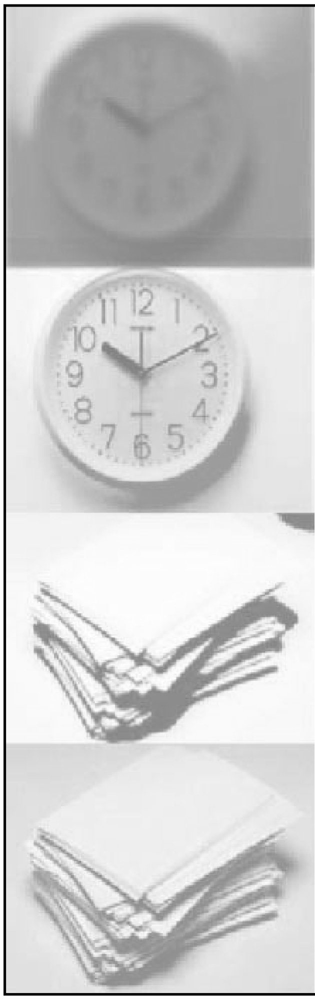
- C(n) là thời gian thực hiện chương trình ứng với trường hợp đệ quy dừng.
- F(T(k)) là một đa thức của các T(k).
- d(n) là thời gian để phân chia bài toán và tổng hợp các kết quả.



Ví dụ về phương trình đệ quy của chương trình đệ quy tính n!

- Gọi T(n) là thời gian tính n!.
- Thì T(n-1) là thời gian tính (n-1)!.
- Trong trường hợp n = 0 thì chương trình chỉ thực hiện một lệnh return 1, nên tốn O(1), do đó ta có T(0) = C_1.
- Trong trường hợp n>0 chương trình phải gọi đệ quy Giai_thua(n-1), việc gọi đệ quy này tốn T(n-1), sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và return tích số.
- Thời gian để thực hiện phép nhân và return là một hằng C_2. Vậy ta có phương trình:

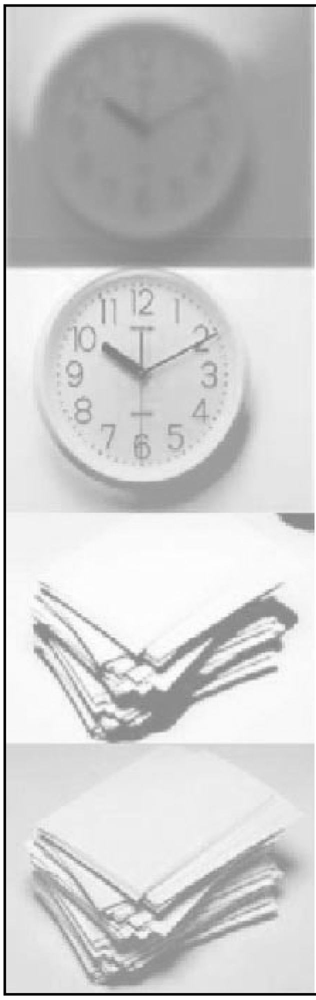
$$T(n) = \begin{cases} C_1 & \text{nêu} \quad n = 0 \\ T(n-1) + C_2 & \text{nêu} \quad n > 0 \end{cases}$$



Giải thuật MergeSort

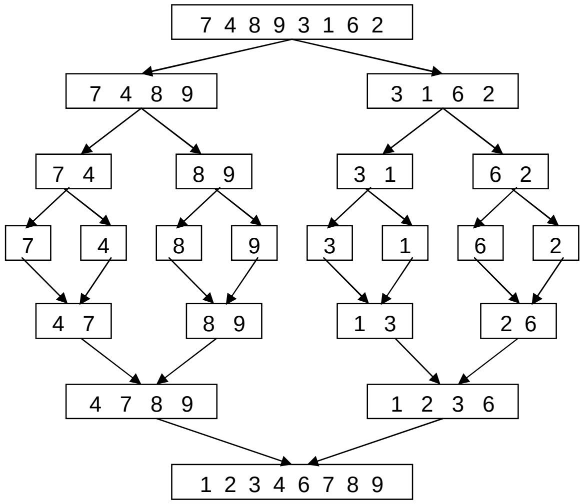
```
List MergeSort (List L; int n){
 List L1,L2
 if (n==1)  RETURN(L);
 else  {
     Chia đôi L thành L1 và L2, với độ dài n/2;

    RETURN(Merge(MergeSort(L1,n/2),MergeSort(L2,n/2)));
  };
};
```

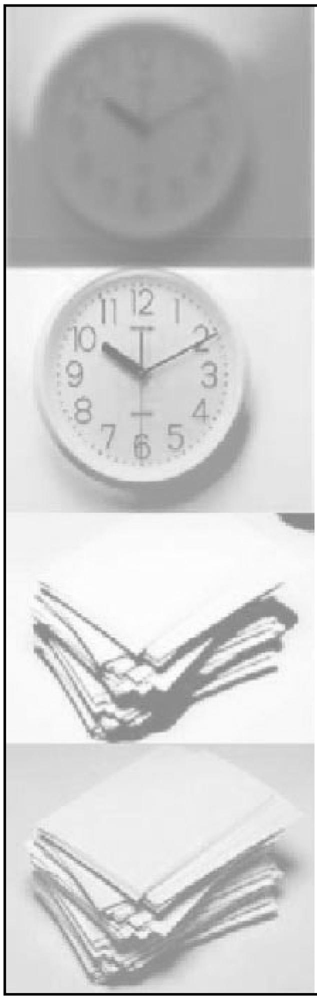
- Hàm MergeSort nhận một danh sách có độ dài n và trả về một danh sách đã được sắp xếp.
- Thủ tục Merge nhận hai danh sách đã được sắp L1 và L2 mỗi danh sách có độ dài , trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự.



Mô hình minh hoạ Mergesort

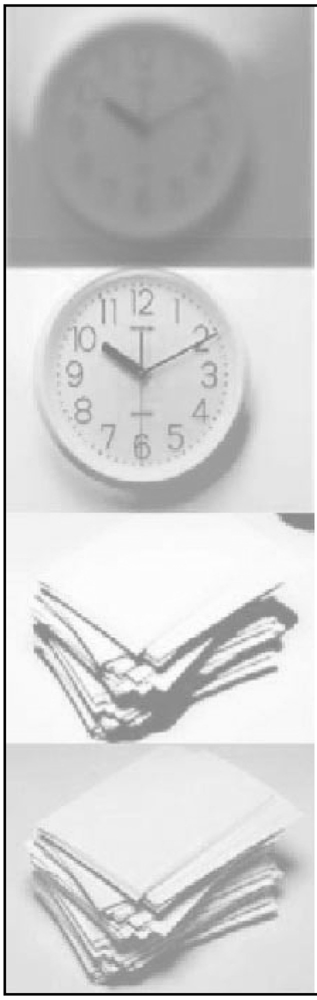
- Sắp xếp danh sách L gồm 8 phần tử 7, 4, 8, 9, 3, 1, 6, 2





Phương trình đệ quy của giải thuật MergeSort

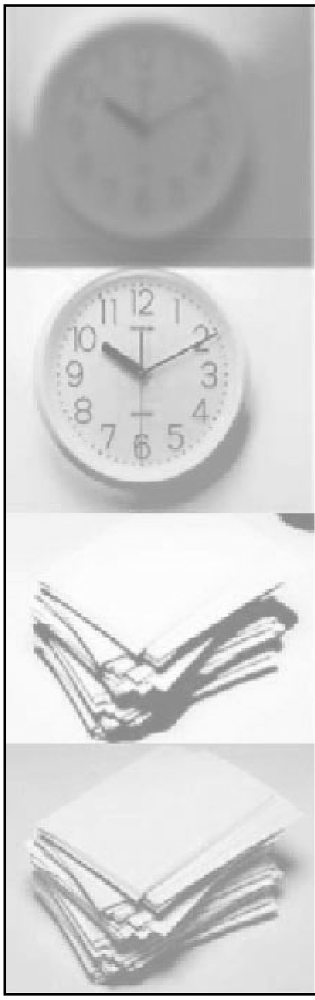
- Gọi T(n) là thời gian thực hiện MergeSort một danh sách n phần tử
- Thì T(n/2) là thời gian thực hiện MergeSort một danh sách n/2 phần tử.
- Khi L có độ dài 1 (n = 1) thì chương trình chỉ làm một việc duy nhất là return(L), việc này tốn O(1) = C1 thời gian.
- Trong trường hợp n > 1, chương trình phải thực hiện gọi đệ quy MergeSort hai lần cho L1 và L2 với độ dài n/2 do đó thời gian để gọi hai lần đệ quy này là 2T(n/2).



Phương trình đệ quy của giải thuật MergeSort (tt)

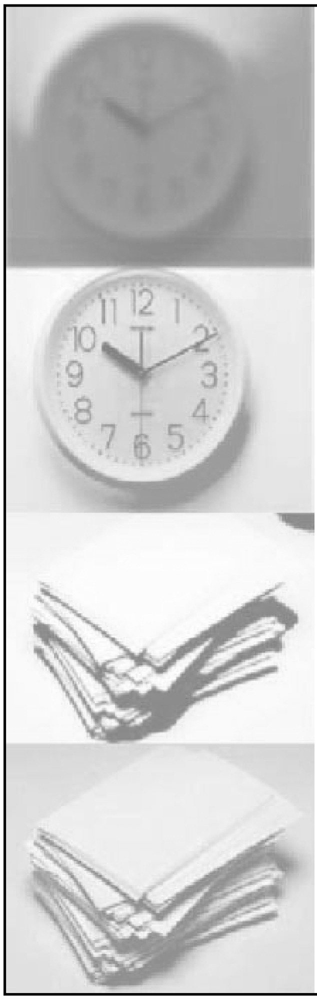
- Ngoài ra còn phải tốn thời gian cho việc chia danh sách L thành hai nửa bằng nhau và trộn hai danh sách kết quả (Merge).
- Người ta xác đinh được thời gian để chia danh sách và Merge là O(n) = C2n .
- Vậy ta có phương trình đệ quy như sau:

$$T(n) = \begin{cases} C_1 & \text{nêu } n = 1 \\ 2T\left(\frac{n}{2}\right) + C_2\, n & \text{nêu } n > 1 \end{cases}$$



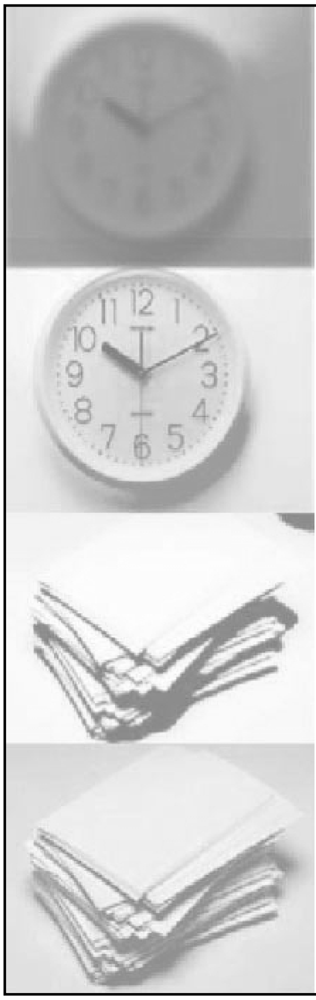
Giải phương trình đệ quy

- Có ba phương pháp giải phương trình đệ quy:
  - **Phương pháp truy hồi**.
  - **Phương pháp đoán nghiệm**.
  - **Lời giải tổng quát của một lớp các phương trình đệ quy**.



Phương pháp truy hồi

- Dùng đệ quy để thay thế bất kỳ T(m) với m < n vào phía phải của phương trình cho đến khi tất cả T(m) với m > 1 được thay thế bởi biểu thức của các T(1) hoặc T(0).
- Vì T(1) và T(0) luôn là hằng số nên chúng ta có công thức của T(n) chứa các số hạng chỉ liên quan đến n và các hằng số.
- Từ công thức đó ta suy ra nghiệm của phương trình.



Ví dụ 1 về giải phương trình đệ quy bằng phương pháp truy hồi

$$T(n) = \begin{cases} C_1 & \text{nêu} \quad n = 0 \\ T(n-1) + C_2 & \text{nêu} \quad n > 0 \end{cases}$$

T(n) = T(n-1) + C2

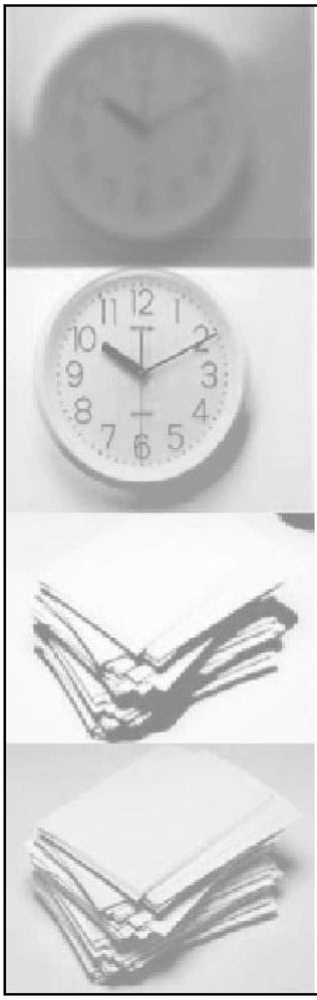
T(n) = [T(n-2) + C2] + C2 = T(n-2) + 2C2

T(n) = [T(n-3) + C2] + 2C2 = T(n-3) + 3C2

……

T(n) = T(n-i) + iC2

- Quá trình trên kết thúc khi n - i = 0 hay i = n.
- Khi đó ta có T(n) = T(0) + nC2 = C1 + nC2 = O(n)



Ví dụ 2 về giải phương trình đệ quy bằng phương pháp truy hồi

$$T(n) = \begin{cases} C_1 & \text{nêu } n = 1 \\ 2T\left(\frac{n}{2}\right) + C_2 n & \text{nêu } n > 1 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + C_2 n$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + C_2\frac{n}{2}\right] + C_2 n = 4T\left(\frac{n}{4}\right) + 2C_2 n$$

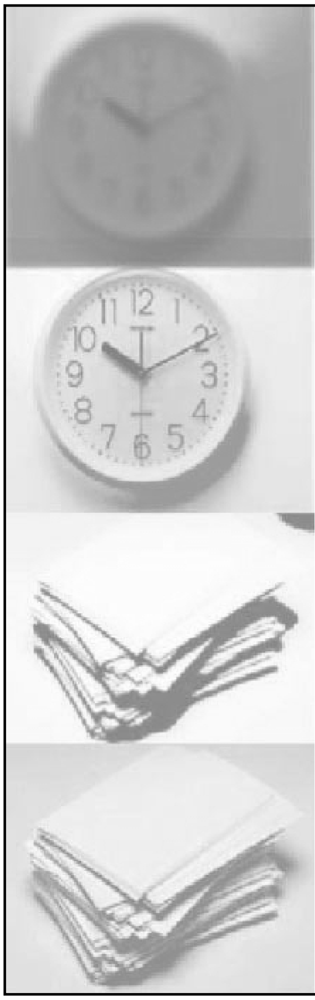
$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + C_2\frac{n}{4}\right] + 2C_2 n = 8T\left(\frac{n}{8}\right) + 3C_2 n$$

………………..

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + iC_2 n$$

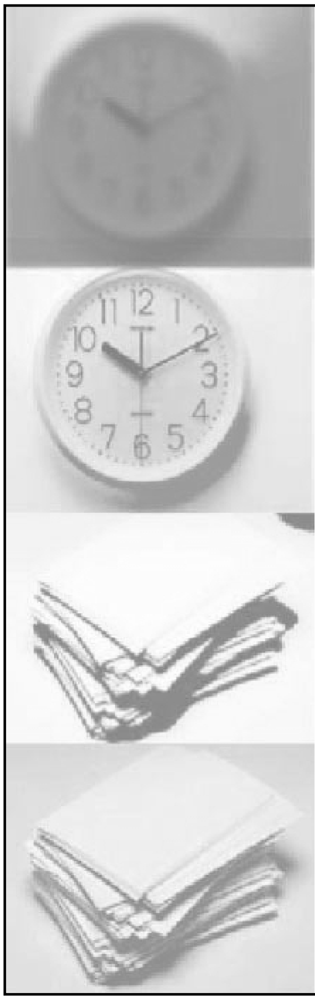
Quá trình suy rộng sẽ kết thúc khi $n/2^i = 1$ hay $2^i = n$ và do đó $i = \log n$. Khi đó ta có:

$$T(n) = nT(1) + \log n C_2 n = C_1 n + C_2 n\log n = O(n\log n).$$



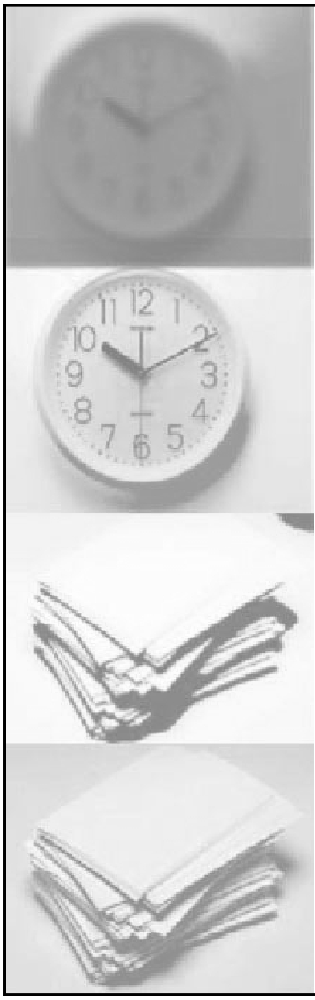
Lời giải tổng quát cho một lớp các phương trình đệ quy

- Trong mục này, chúng ta sẽ nghiên cứu các phần sau:
  - Bài toán đệ quy tổng quát.
  - Thành lập phương trình đệ quy tổng quát.
  - Giải phương trình đệ quy tổng quát.
  - Các khái niệm về nghiệm thuần nhất, nghiệm riêng và hàm nhân.
  - Nghiệm của phương trình đệ quy tổng quát khi d(n) là hàm nhân.
  - Nghiệm của phương trình đệ quy tổng quát khi d(n) không phải là hàm nhân.



Bài toán đệ quy tổng quát

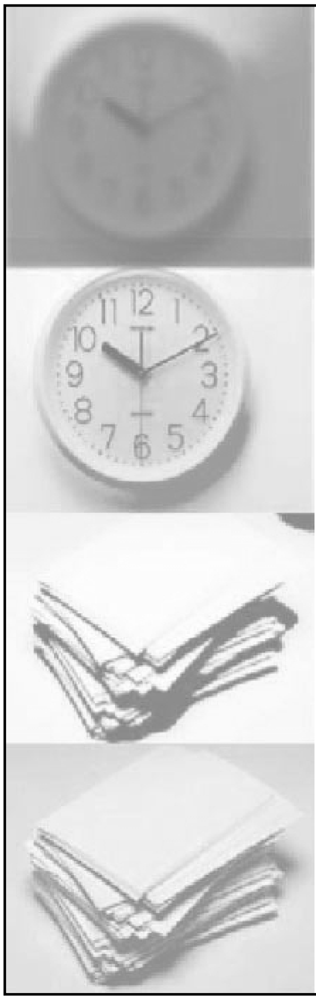
- Để giải một bài toán kích thước n, ta chia bài toán đã cho thành a bài toán con, mỗi bài toán con có kích thước n/b. Giải các bài toán con này và tổng hợp kết quả lại để được kết quả của bài toán đã cho.
- Với các bài toán con chúng ta cũng sẽ áp dụng phương pháp đó để tiếp tục chia nhỏ ra nữa cho đến các bài toán con kích thước 1. Kĩ thuật này sẽ dẫn chúng ta đến một giải thuật đệ quy.
- Giả thiết rằng mỗi bài toán con kích thước 1 lấy một đơn vị thời gian
- Giả thiết thời gian để chia bài toán kích thước n thành các bài toán con kích thước n/b và tổng hợp kết quả từ các bài toán con để được lời giải của bài toán ban đầu là d(n).



Thành lập phương trình đệ quy tổng quát

- Nếu gọi T(n) là thời gian để giải bài toán kích thước n
- Thì T(n/b) là thời gian để giải bài toán con kích thước n/b.
- Khi n = 1 theo giả thiết trên thì thời gian giải bài toán kích thước 1 là 1 đơn vị, tức là T(1) = 1.
- Khi n lớn hơn 1, ta phải giải đệ quy a bài toán con kích thước n/b, mỗi bài toán con tốn T(n/b) nên thời gian cho a lời giải đệ quy này là aT(n/b).
- Ngoài ra ta còn phải tốn thời gian để phân chia bài toán và tổng hợp các kết quả, thời gian này theo giả thiết trên là d(n). Vậy ta có phương trình đệ quy:

$$T(n) = \begin{cases} 1 & \text{neu } n = 1 \\ aT\left(\frac{n}{b}\right) + d(n) & \text{neu } n > 1 \end{cases}$$



Giải phương trình đệ quy tổng quát

$$T(n) = aT\left(\frac{n}{b}\right) + d(n)$$

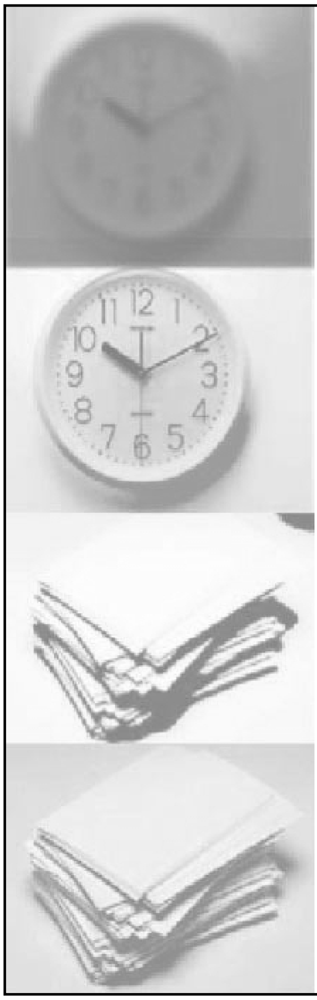
$$T(n) = a\left[aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b}\right)\right] + d(n) \quad = a^2T\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$T(n) = a^2\left[aT\left(\frac{n}{b^3}\right) + d\left(\frac{n}{b^2}\right)\right] + ad\left(\frac{n}{b}\right) + d(n)$$

$$= a^3T\left(\frac{n}{b^3}\right) + a^2d\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

....

$$T(n) = a^iT\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j d\left(\frac{n}{b^j}\right)$$



Giải phương trình đệ quy tổng quát (tt)

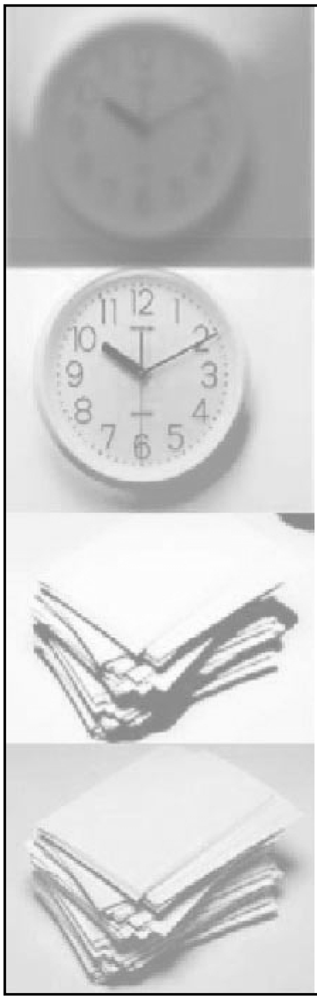
$$T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j d\left(\frac{n}{b^j}\right)$$

Giả sử $n = b^k$, quá trình suy rộng trên sẽ kết thúc khi $i = k$. Khi đó ta được:

$$T\left(\frac{n}{b^i}\right) = T\left(\frac{n}{b^k}\right) = T\left(\frac{b^k}{b^k}\right) = T(1) = 1$$

Thay vào trên ta có:

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right)$$



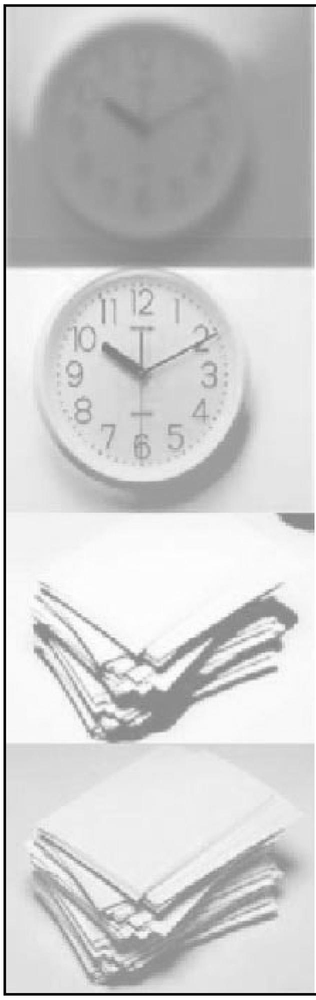
Nghiệm thuần nhất và nghiệm riêng

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right)$$

Nghiệm thuần nhất
$a^k = n^{\log_b a}$

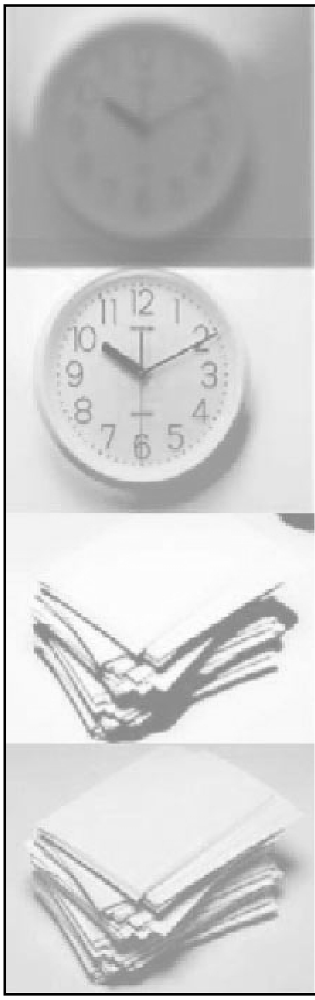
Nghiệm riêng

Nghiệm của phương trình là: MAX(NTN,NR).



Hàm nhân

- Một hàm f(n) được gọi là **hàm nhân** (multiplicative function) nếu f(m.n) = f(m).f(n) với mọi số nguyên dương m và n.
- **Ví dụ:**
  - Hàm $f(n) = n^k$ là một hàm nhân, vì $f(m.n) = (m.n)^k = m^k.n^k = f(m).f(n)$.
  - Hàm f(n) = logn không phải là một hàm nhân, vì $f(n.m) = \log(n.m) = \log n+\log m \neq \log n.\log m = f(n).f(m)$

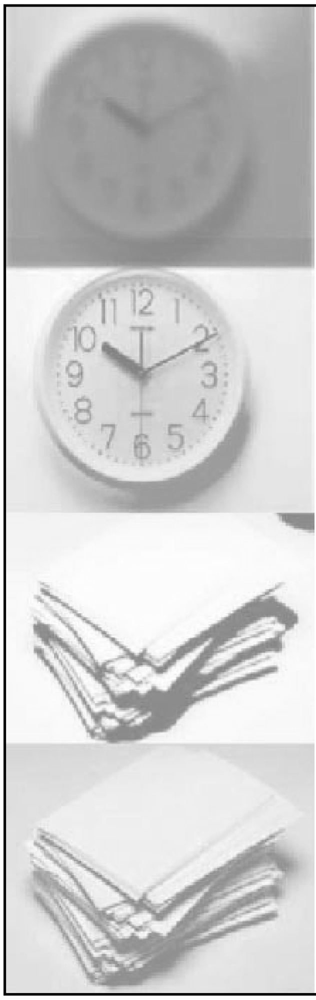


Tính nghiệm riêng khi d(n) là hàm nhân

- Khi d(n) là hàm nhân, ta có:
- $d(b^{k-j}) = d(b.b.b\ldots..b) = d(b).d(b)\ldots d(b) = [d(b)]^{k-j}$

$$NR = \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right) = \sum_{j=0}^{k-1} a^j [d(b)]^{k-j} = [d(b)]^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right]^j = [d(b)]^k \frac{\left[\frac{a}{d(b)}\right]^k - 1}{\frac{a}{d(b)} - 1}$$

$$\text{Hay } NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$



Ba trường hợp

$$NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

- **Trường hợp 1**: a > d(b)

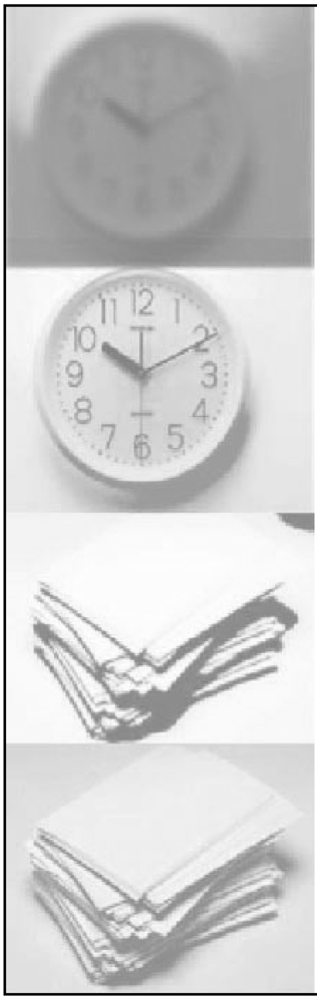
  Trong công thức trên ta có $a^k > [d(b)]^k$, theo quy tắc lấy độ phức tạp ta có NR là $O(a^k) = O(n^{\log_b a})$ = NTN.

  Do đó **T(n) là $O(n^{\log_b a})$**.

- **Trường hợp 2**: a < d(b)

  Trong công thức trên ta có $[d(b)]^k > a^k$, theo quy tắc lấy độ phức tạp ta có NR là $O([d(b)]^k) = O(n^{\log_b d(b)})$ > NTN.

  Do đó **T(n) là $O(n^{\log_b d(b)})$**.



Ba trường hợp (tt)

$$NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

Trường hợp 3: a = d(b)

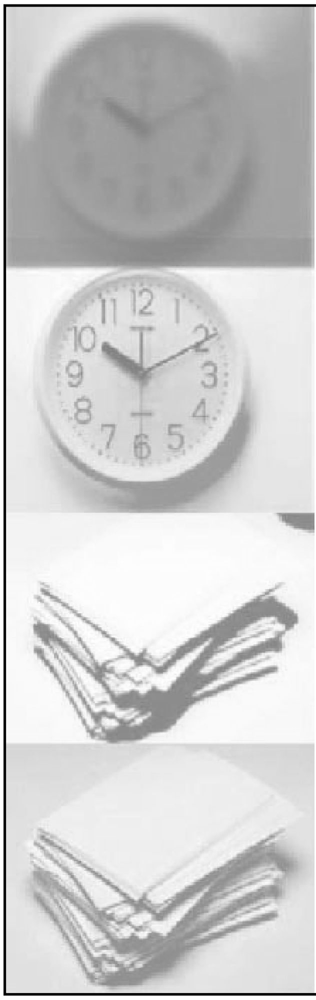
Công thức trên không xác đinh nên ta phải tính trực tiếp nghiệm riêng:

$$NR = [d(b)]^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right]^j = a^k \sum_{j=0}^{k-1} 1 = a^k k \qquad (\text{do } a = d(b))$$

Do $n = b^k$ nên $k = \log_b n$ và $a^k = n^{\log_b a}$.

Vậy NR là $n^{\log_b a}\log_b n$ > NTN.

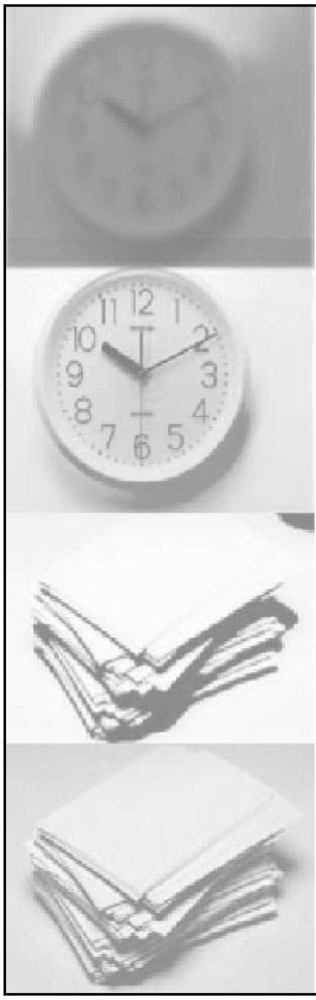
Do đó **T(n) là $O(n^{\log_b a}\log_b n)$**.



Ví dụ: GPT với T(1) = 1 và

$$1/\quad T(n) = 4T\left(\frac{n}{2}\right) + n$$

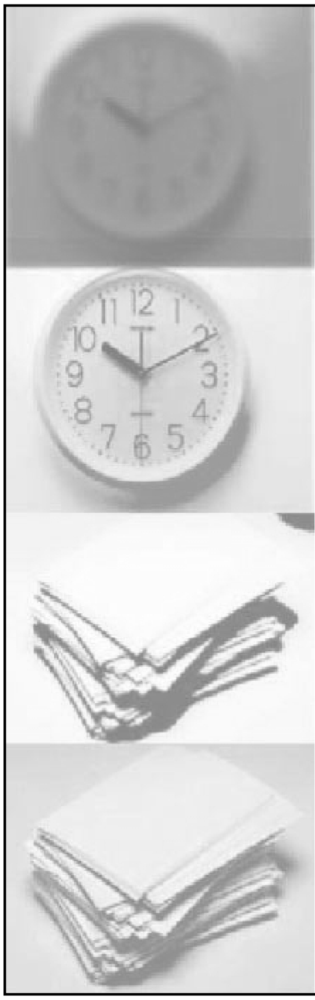
- Phương trình đã cho có dạng phương trình tổng quát.
- d(n)=n là hàm nhân.
- a = 4 và b = 2.
- d(b) = b = 2 < a.
- T(n) = $O(n^{\log_b a}) = O(n^{\log 4}) = O(n^2)$.



Ví dụ: GPT với T(1) = 1 và

$$2/\ \ T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

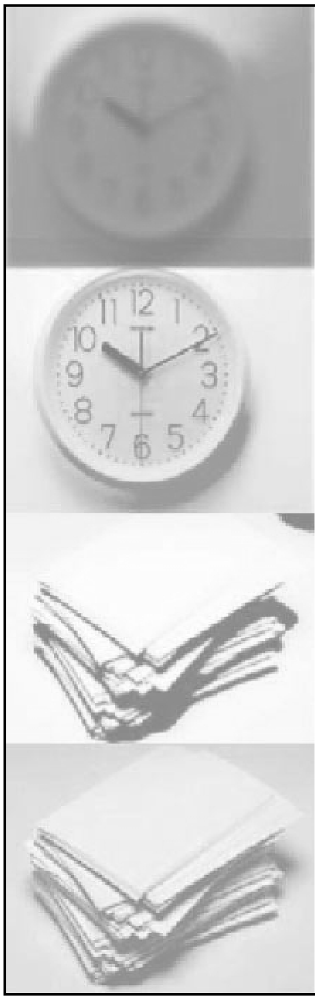
- Phương trình đã cho có dạng phương trình tổng quát.
- $d(n)=n^2$ là hàm nhân.
- $a = 4$ và $b = 2$.
- $d(b) = b^2 = 4 = a$.
- $T(n) = O(n^{\log_b a}\log_b n)$
  $= O(n^{\log 4}\log n) = O(n^2\log n)$.



Ví dụ: GPT với T(1) = 1 và

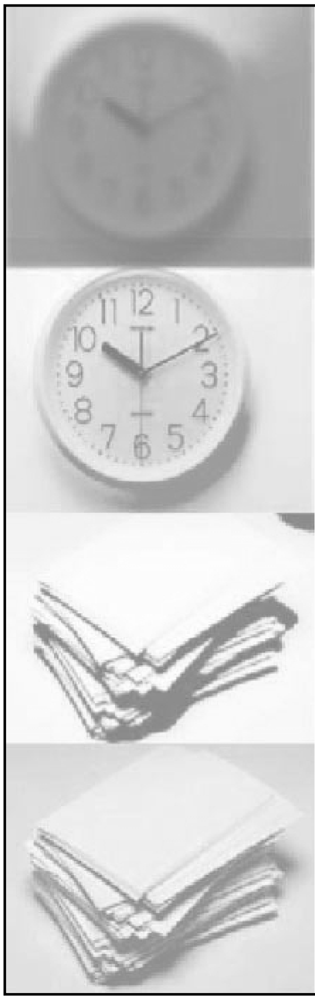
$$3/\ \ T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

- Phương trình đã cho có dạng phương trình tổng quát.
- $d(n)=n^3$ là hàm nhân.
- $a = 4$ và $b = 2$.
- $d(b) = b^3 = 8 > a$.
- $T(n) = O(n^{\log_b d(b)}) = O(n^{\log 8}) = O(n^3)$.



Nghiệm của phương trình đệ quy tổng quát khi d(n) không phải là hàm nhân

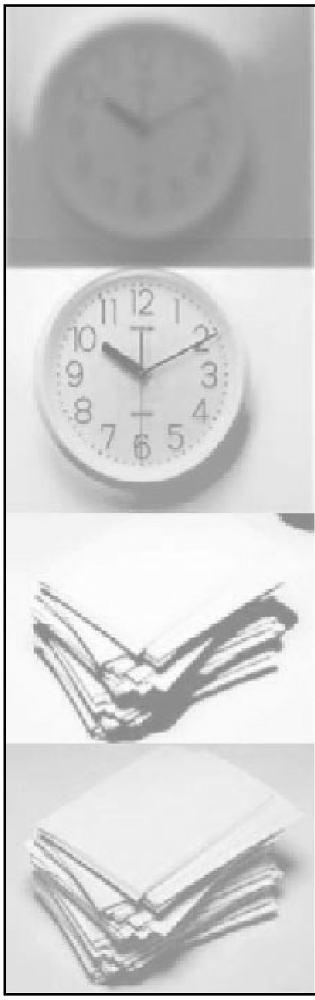
- Trong trường hợp hàm tiến triển không phải là một hàm nhân thì chúng ta không thể áp dụng các công thức ứng với ba trường hợp nói trên mà chúng ta phải tính trực tiếp NR, sau đó lấy MAX(NR,NTN).



Ví dụ: GPT với T(1) = 1 và

$$T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$

- PT thuộc dạng phương trình tổng quát nhưng d(n) = nlogn không phải là một hàm nhân.
- NTN = $n^{\log_b a} = n^{\log 2} = n$
- Do d(n) = nlogn không phải là hàm nhân nên ta phải tính nghiệm riêng bằng cách xét trực tiếp

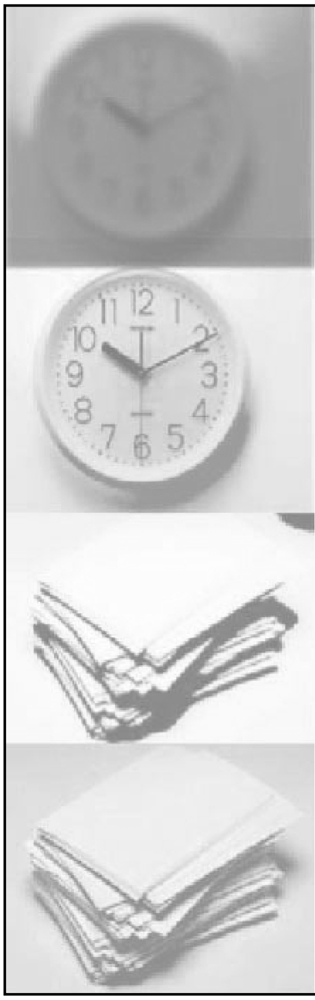


Ví dụ (tt)

$$NR = \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right) = \sum_{j=0}^{k-1} 2^j 2^{k-j} \log 2^{k-j}$$

$$NR = 2^k \sum_{j=0}^{k-1} (k - j) = 2^k \frac{k(k+1)}{2} = O(2^k k^2)$$

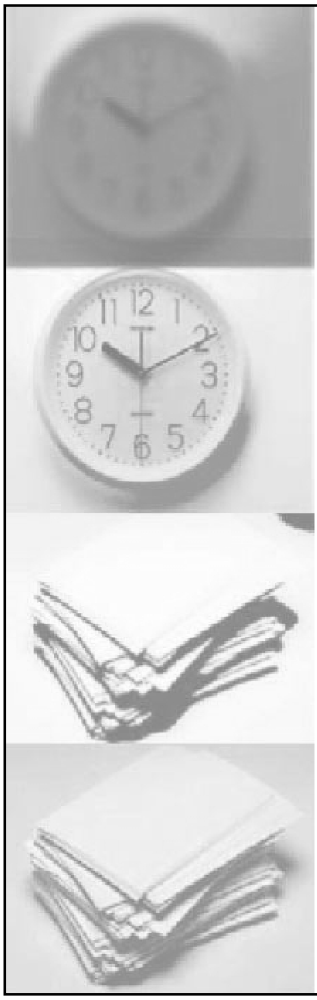
- Theo giải phương trình đệ quy tổng quát thì $n = b^k$ nên $k = \log_b n$, ở đây do $b = 2$ nên $2^k = n$ và $k = \log n$,
- $NR = O(n\log^2 n) > NTN$
- $T(n) = O(n\log^2 n)$.



BT4-1: GPT với T(1) = 1 và

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

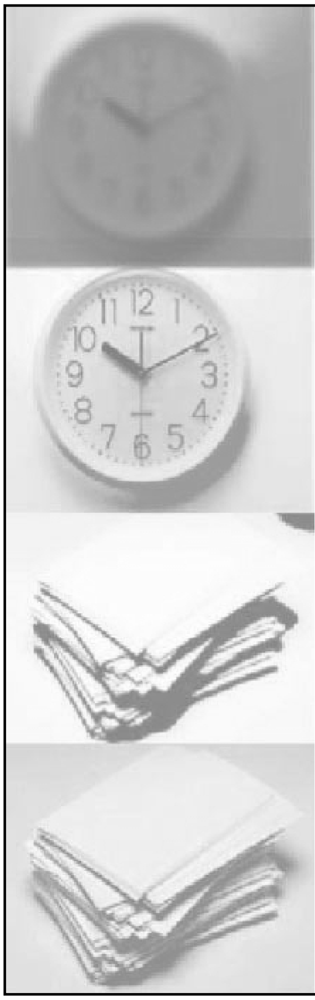
- Phương trình đã cho có dạng phương trình tổng quát.
- d(n)=1là hàm nhân.
- a = 1 và b = 2.
- d(b) = 1 = a.
- T(n) = $O(n^{\log_b a} \log_b n)$ = $O(n^{\log 1} \log n)$ = O(logn).



BT4-2: GPT với T(1) = 1 và

$$T(n) = 2T\left(\frac{n}{2}\right) + \log n$$

- Phương trình đã cho có dạng phương trình tổng quát.
- d(n)=logn không phải là hàm nhân.
- NTN = $O(n^{\log_b a}) = O(n^{\log 2}) = O(n)$.
- Tính trực tiếp nghiệm riêng.



$$\text{BT4 - 2 : GTP voi T(1) = 1 va T(n) = 2T}\left(\frac{\text{n}}{2}\right) + \text{logn}$$

$$NR = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j \log 2^{k-j}$$

$$NR = \sum_{j=0}^{k-1} 2^j (k-j) = \sum_{j=0}^{k-1} k2^j - \sum_{j=0}^{k-1} j2^j$$

$$NR = O(k\sum_{j=0}^{k-1} 2^j) = O(k\frac{2^k - 1}{2-1})$$

$$NR = O(k2^k) = O(n\log n) > n = NTN$$

$$T(n) = O(n\log n)$$