CHƯƠNG 1

KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

Tuần 2 - CÁCH TÍNH ĐỘ PHỨC TẠP

Cách tính độ phức tạp

Cho 2 đoạn chương trình:

- P1 có thời gian thực hiện T1(n)=O(f1(n)).
- P2 có thời gian thực hiện T2(n)=O(f2(n)).

- **Quy tắc cộng**:

  Thời gian thực hiện P1 và P2 **nối tiếp** nhau sẽ là:

  T(n) = T1(n) + T2(n) = O(max(f1(n), f2(n))

- **Quy tắc nhân**:

  Thời gian thực hiện P1 và P2 **lồng nhau** (chẳng hạn vòng lặp lồng nhau) sẽ là:

  T(n) = T1(n) × T2(n) = O(f1(n) × f2(n))

CANTHO UNIVERSITY

Ví dụ về quy tắc cộng và nhân

VD : P_1 có $T_1(n) = \mathbf{O(n^2)}$; P_2 có $T_2(n) = \mathbf{O(n^3)}$

(+) P_1 và P_2 **nối tiếp nhau** : Quy tắc cộng

(+) P_1;
P_2;

$$\mathbf{T(n) = T1(n) + T2(n) = O(max(n^2, n^3)) = O(n^3)}$$

(×) P_1 và P_2 **lồng nhau** : Quy tắc nhân

(×) P_1
P_2 ;

$$\mathbf{T(n) = T1(n) \times T2(n) = O(n^2 \times n^3) = O\ (n^5\ )}$$

Quy tắc tổng quát tính độ phức tạp

- Lệnh đọc (**read, scanf**), lệnh ghi (**write, prinft**), lệnh **gán,** lệnh **return, định trị biểu thức, so sánh**: Thời gian = hằng số hay **O(1)**
- Lệnh **if** :

```
if (điều kiện)
    lệnh 1
else
    lệnh 2;
```

  - Thời gian = **max (lệnh 1, lệnh 2) + điều kiện** = max (lệnh 1, lệnh 2, điều kiện)

. **Vòng lặp**:

  - Thời gian = **Tổng thời gian thực hiện thân vòng lặp**
  - Vòng lặp có số lần lặp xác định (**for**): tính số lần lặp.
  - Vòng lặp có số lần lặp không xác định (**while**): tính số lần lặp trong *trường hợp xấu nhất* (lặp nhiều nhất).

Phương pháp tính độ phức tạp

- Xét phương pháp tính độ phức tạp trong 3 trường hợp:

(1) Chương trình **không gọi** chương trình con.

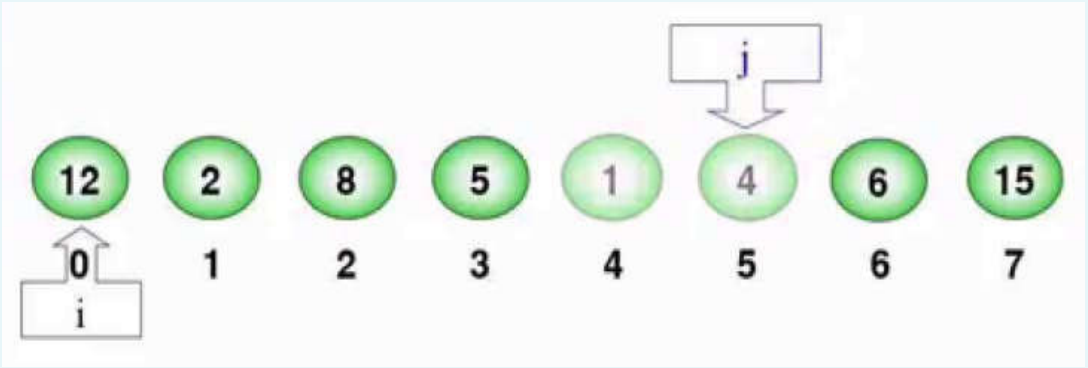
(2) Chương trình **có gọi** chương trình con *không đệ quy.*

(3) Chương trình **đệ quy.**

Ví dụ 1.
Thủ tục sắp xếp "nổi bọt"

(1) Chương trình **không gọi** chương trình con.

```
void BubbleSort(int a[], int 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-1] > a[j]) {
4.              temp=a[j-1];
5.              a[j-1] = a[j];
6.              a[j]    = temp;
              }
 }
```



Quy tắc tổng quát tính độ phức tạp

- Lệnh đọc (**read, scanf**), lệnh ghi (**write, prinft**), lệnh **gán,** lệnh **return, định trị biểu thức, so sánh**: Thời gian = hằng số hay **O(1)**
- Lệnh **if** :

```
if (điều kiện)
    lệnh 1
else
    lệnh 2;
```

  - Thời gian = **max (lệnh 1, lệnh 2) + điều kiện** = max (lệnh 1, lệnh 2, điều kiện)

. **Vòng lặp**:

  - Thời gian = **Tổng thời gian thực hiện thân vòng lặp**
  - Vòng lặp có số lần lặp xác định (**for**): tính số lần lặp.
  - Vòng lặp có số lần lặp không xác định (**while**): tính số lần lặp trong *trường hợp xấu nhất* (lặp nhiều nhất).

Ví dụ 1.
Thủ tục sắp xếp "nổi bọt"

(1) Chương trình **không gọi** chương trình con.

```
void BubbleSort(int a[], int 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-1] > a[j]) {
4.            temp = a[j-1];
5.            a[j-1]= a[j];
6.            a[j]   = temp;
            }
 }
```

(×) (×) (+)

1. $O((n-2)-0+1)) \rightarrow O(n \times n) = O(n^2)$
2. $O((n-1)-(i+1)+1)) \rightarrow O(n \times 1) = O(n)$
3. $O(1)$
4. $O(1)$
5. $O(1)$
6. $O(1)$

$\rightarrow O(\max(1, 1)) = O(1)$

$\rightarrow O(\max(1, 1, 1)) = O(1)$

$\rightarrow$ **$T(n) = O(n^2)$**

Ví dụ 1.
Thủ tục sắp xếp "nổi bọt"

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

Số số hạng = (Số cuối – số đầu) + 1

Tính độ phức tạp của thủ tục sắp xếp "nổi bọt"

- 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}.
- 3 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-1) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n-i-1)×1) = O(n-i-1).
- Vòng lặp {1} có i chạy từ 0 đến n-2 nên thời gian thực hiện của vòng lặp {1} cũng là độ phức tạp của thuật toán:

$$T(n)=\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}=O(n^2)$$

Ví dụ 2.
Thủ tục tìm kiếm tuần tự

(1) Chương trình **không gọi** chương trình con.

```
1.    int tim_kiem (int x, int a[ ], int n)) {
2.        int found, i;
3.        found = 0;
4.        i = 0;
5.            while (i < n  &&  ! found)
6.              if (a[i] == x)
7.                  found = 1;
8.              else
9.                  i = i + 1;
10.       return i;
11.   }
```

Ví dụ 2.
Thủ tục tìm kiếm tuần tự

(1) Chương trình **không gọi** chương trình con.

```
1.   int tim_kiem (int x, int a[ ], int n)) {
2.       int found, i;
3.       found = 0;
4.       i = 0;
5.       while (i < n  &&  ! found)
6.           if (a[i] == x)
7.               found = 1;
8.           else
9.               i = i + 1;
10.      return i;
11.  }
```

→ **T(n) = O(n)**

3. O(1)
4. O(1)
5. O(n) → O(n × 1) = O(n)
6. O(1) → O(max(1, 1, 1)) = O(1)
7. O(1)
9. O(1)
10. O(1)→ O(max(1, 1, n, 1))=**O(n)**

(+) (×)

Tính độ phức tạp của hàm tìm kiếm tuần tự

- Các lệnh {3}, {4}, {5} và {10} nối tiếp nhau, do đó độ phức tạp của hàm Search là độ phức tạp lớn nhất trong 4 lệnh này.
- 3 lệnh {3}, {4} và {10} đều có độ phức tạp O(1), do đó độ phức tạp lớn nhất trong 4 lệnh này là độ phức tạp của lệnh {5}. Lồng trong lệnh {5} là lệnh {6}, lệnh {6} có độ phức tạp O(1).
- Lệnh {5} là một vòng lặp không xác định, do đó cần tính số lần lặp trong trường hợp xấu nhất (khi tất cả các phần tử của mảng a đều khác x): xét tất cả các a[i], i có các giá trị từ 0 đến (n – 1) hay vòng lặp {3} thực hiện n lần, tốn O(n) thời gian.

Vậy T(n) = O(n)

Bài tập

Bài 1: Tính thời gian thực hiện của các đoạn chương trình sau:

a) Tính tổng của các số

```
{1}   Sum := 0;
{2}   for i:=1 to n do begin
{3}         readln(x);
{4}         Sum := Sum + x;
      end;
```

b) Tính tích hai ma trận vuông cấp n C = A*B:

```
{1}   for i := 1 to n do
{2}     for j := 1 to n do begin
{3}       c[i,j] := 0;
{4}       for k := 1 to n do
{5}             c[i,j] := c[i,j] + a[i,k] * b[k,j];
        end;
```

Bài giải – Bài tập 1a).

1. a) Tính tổng của các số

```
1.      sum := 0;                          → 1. O(1)
2.      for i:=1 to n do begin             → 2. O(n)
3.          readln (x);                    → 3. O(1)
4.          sum := sum + x;                → 4. O(1)
        end;
```

(+) (x) (+)

→ O(max(1, n) = **O(n)**

→ O(n × 1) = O(n)

→ O(max(1, 1) = O(1)

→ **T(n) = O(n)**

Bài giải – Bài tập 1b).

1. b) Tính tích 2 ma trận vuông cấp n

```
1. (×) for i:=1 to n do                          → 1. O(n)
2.     for j:=1 to n do begin                    → 2. O(n)
3. (×)     c[i, j] := 0;                         → 3. O(1)
(+)        for k:=1 to n do                      → 4. O(n)
4.
5. (×)     c[i, j]:=c[i,j]+a[i,k]*b[k,j];        → 5. O(1)
       end;
```

→ 4. O(n) + → 5. O(1) → O(n)

→ 2. O(n) + → O(n) → $O(n^2)$

→ 1. O(n) + → $O(n^2)$ → $O(n^3)$

→ **T(n) = $O(n^3)$**

Phương pháp tính độ phức tạp

- Xét phương pháp tính độ phức tạp trong 3 trường hợp:

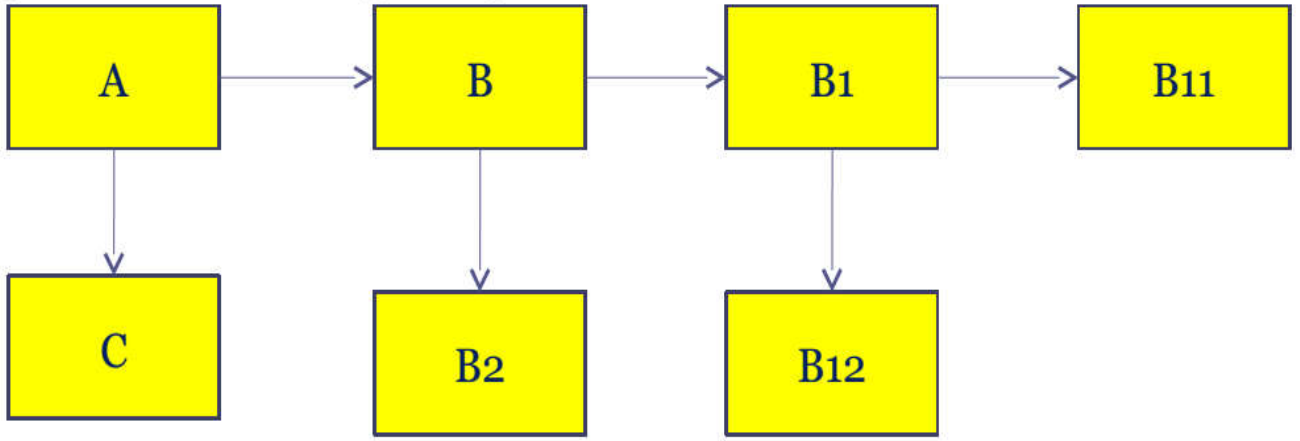
(1) Chương trình **không gọi** chương trình con.

(2) Chương trình **có gọi** chương trình con *không đệ quy.*

(3) Chương trình **đệ quy.**

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

- Quy tắc: tính từ trong ra ngoài



- C, B2, B12, B11
- B1
- B
- A

Ví dụ 3.

Sắp xếp "nổi bọt" gọi chương trình con

(2) Chương trình **có gọi** chương trình con không đệ quy

```
1.      void Swap (int &a,int &b) {
2.        int temp;
3.        temp = x;
4.        x = y;
5.        y = temp; }

6.   void BubbleSort(int a[],int n) {
7.      for(i= 0; i<=n-2; i++)
8.        for(j=n - 1; j>=i+1;j--)
9.          if (a[j-1] > a[j]) {
10.           Swap(a[j-1],a[j]);  }
```

(+) x = y; y = temp;

(×) for(j=n - 1; j>=i+1;j--) if (a[j-1] > a[j]) { Swap(a[j-1],a[j]); }

(×) if (a[j-1] > a[j]) { Swap(a[j-1],a[j]); }

Swap: O(1)

for(i= 0; i<=n-2; i++): O(n)

for(j=n - 1; j>=i+1;j--): O(n)

if (a[j-1] > a[j]) { Swap(a[j-1],a[j]); }: O(1) + O(1) → O(1)

O(n) × O(n) → **$O(n^2)$**

Tính độ phức tạp của sắp xếp "nổi bọt" có gọi chương trình con

- Chương trình BubbleSort gọi chương trình con Swap, do đó để tính độ phức tạp của BubbleSort trước tiên cần tính độ phức tạp của Swap.
- Swap bao gồm 3 lệnh gán 3 lệnh gán {3}, {4} và {5} đều tốn O(1), do đó độ phức tạp của Swap là O(1).
- Trong BubbleSort: lệnh {10} gọi Swap nên tốn O(1); lệnh {9} có điều kiện so sánh a[j] < a[j-1] cũng tốn O(1); vòng lặp {8} thực hiện (n – i - 1) lần, mỗi lần tốn O(1) nên vòng lặp {8} tốn O((n-i- 1) ×1) = O(n-i-1); Vòng lặp {7} có i chạy từ 0 đến n-1 nên thời gian thực hiện của vòng lặp {7} cũng là độ phức tạp của thuật toán:

$$T(n)=\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}=O(n^2)$$

Phương pháp tính độ phức tạp

- Xét phương pháp tính độ phức tạp trong 3 trường hợp:

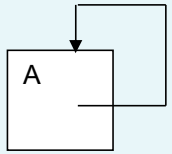
(1) Chương trình **không gọi** chương trình con.

(2) Chương trình **có gọi** chương trình con *không đệ quy.*

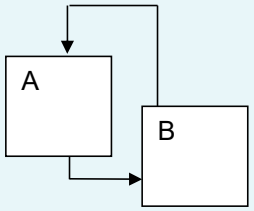
(3) Chương trình **đệ quy.**

Phân tích các chương trình đệ qui

- 2 dạng chương trình đệ quy:



Đệ quy trực tiếp



Đệ quy gián tiếp

- Tính độ phức tạp chương trình đệ quy:

  (1) Thành lập phương trình đệ quy T(n)

  (2) Giải phương trình đệ quy tìm nghiệm

  → Suy ra tỷ suất tăng f(n) hay O(f(n).

Chương trình đệ quy

- Chương trình đệ quy 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* giải bài toán kích thước k (k<n)
- **Ví dụ** : Chương trình đệ quy tính n!

```
1.  int giai_thua(int n) {
2.    if (n == 0)
3.      return 1;
4.    else
5.      return n * giai_thua(n - 1);
6.  }
```

- *Trường hợp dừng* n = 0 và *lời gọi đệ quy* k = n-1.

Thành lập phương trình đệ quy

- **Phương trình đệ quy** là 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, phải căn cứ vào chương trình đệ quy.

  - ***Khi đệ quy dừng***: xem xét khi đó chương trình làm gì và tốn hết bao nhiêu thời gian (thông thường thời gian này là hằng số C(n))

  - ***Khi đệ quy chưa dừng***: xem xét có bao nhiêu lời gọi đệ quy với kích thước k thì sẽ có bấy nhiêu T(k).
- Ngoài ra, còn phải xem xét thời gian phân chia bài toán và tổng hợp các lời giải (chẳng hạn gọi thời gian này là d(n)).

Dạng phương trình đệ quy

Dạng tổng quát của một **phương trình đệ quy** là:

$$T(n) = \begin{cases} C(n) & \leftarrow \textit{Khi đệ quy dừng} \\ F(T(k)) + d(n) & \leftarrow \textit{Khi đệ quy chưa dừng} \end{cases}$$

- **C(n)**: thời gian thực hiện chương trình ứng với trường hợp đệ quy dừng.
- **F(T(k)):** hàm xác định thời gian theo T(k).
- **d(n)**: thời gian phân chia bài toán và tổng hợp các kết quả.

CANTHO UNIVERSITY

Ví dụ 1. Phương trình đệ quy của n!

n! = n*(n-1)*(n-2)*… *2*1

```
1.  int giai_thua(int n) {
2.    if (n == 0)
3.      return 1;
4.    else
5.      return n * giai_thua(n - 1);
6.  }
```

- Gọi T(n) là thời gian tính n!.
- Thì T(n-1) là thời gian tính (n-1)!.
- Trường hợp n = 0 : thực hiện *return 1*
  tốn O(1) → T(0) = C_1.
- Trường hợp n > 0: gọi đệ quy *giai_thua(n-1)*,
  tốn T(n-1) → T(n) = T(n -1).
- Sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và trả về tích số. Thời gian thực hiện phép nhân và trả kết quả về là một hằng C_2.
- Vậy ta có phương trình đệ quy như sau:

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

Ví dụ 2. Phương trình đệ quy của thuật toán MergeSort

```
FUNCTION MergeSort (L:List; n:Integer):List;
VAR  L1,L2:List;
BEGIN
  IF n=1 THEN RETURN(L)
  ELSE BEGIN
     Chia đôi L thành L1 và L2, với độ dài n/2;
     RETURN(Merge(MergeSort(L1,n/2),MergeSort(L2,n/2)));
   END;
END;
```

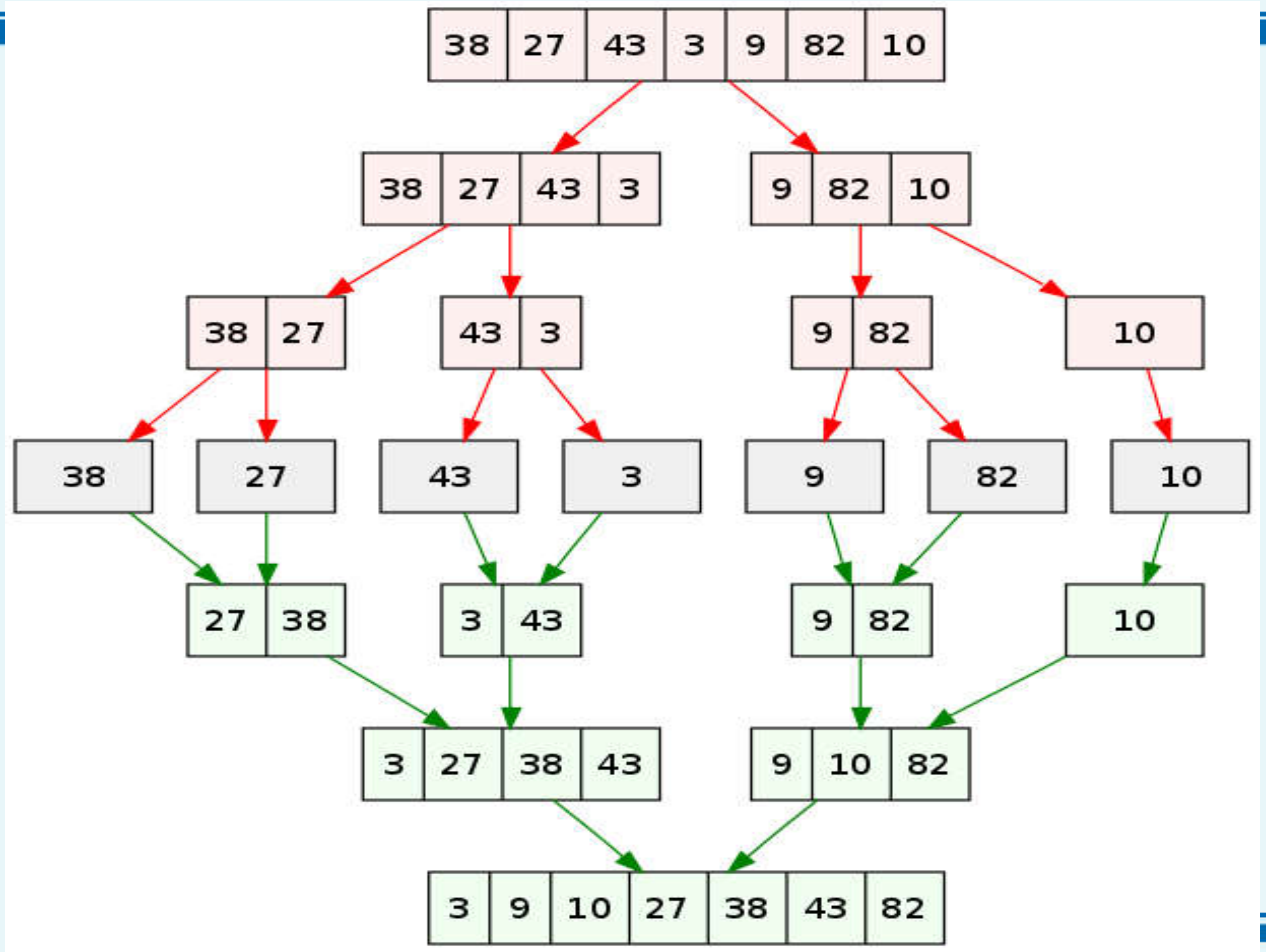
- ***Hàm MergeSort*** nhận danh sách có độ dài n và trả về danh sách được sắp xếp.
- ***Thủ tục Merge*** nhận 2 danh sách đã được sắp thứ tự L1 và L2, mỗi danh sách có độ dài n/2, trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự.

CANTHO UNIVERSITY

Mô hình minh hoạ Mergesort

- Sắp xếp danh sách L gồm 7 phần tử :

38, 27, 43, 3, 9, 82, 10



Ví dụ 2. Phương trình đệ quy của thuật toán MergeSort

- T(n): **mergeSort** (n phần tử)

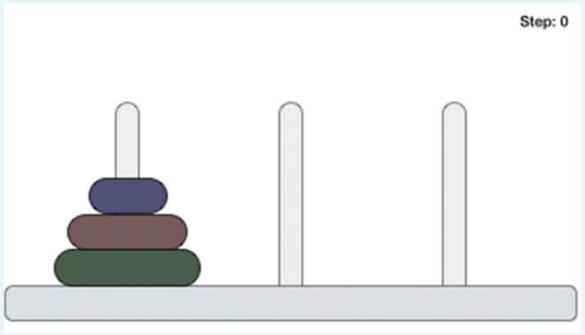
  T(n/2): **mergeSort** (n/2 phần tử).
- Khi n = 1: *return(L)* $\rightarrow$ T(n) = C_1.
- Khi n > 1: gọi đệ quy **mergeSort** 2 lần cho 2 danh sách con với độ dài n/2 $\rightarrow$ T(n) = 2T(n/2).
- Ngoài ra, thời gian chia danh sách và trộn hai danh sách kết quả (hàm **merge**): cần thời gian O(n) = nC_2.
- Vậy ta có phương trình đệ quy như sau:

```
FUNCTION MergeSort (L:List; n:Integer):List;
VAR  L1,L2:List;
BEGIN
  IF n=1 THEN RETURN(L)
  ELSE BEGIN
     Chia đôi L thành L1 và L2, với độ dài n/2;
     RETURN(Merge(MergeSort(L1,n/2),MergeSort(L2,n/2)));
   END;
END;
```

$$T(n) = \begin{cases} C_1 & \text{nếu n=1} \\ 2T(\frac{n}{2}) + nC_2 & \text{nếu n>1} \end{cases}$$

Ví dụ 3. Phương trình đệ quy của Thủ tục tháp Hà nội số tầng n

```
void ThapHN(int n, char A, char B, char C)
{  if(n > 1)
   {
     ThapHN(n-1,A, C, B);
     printf("Chuyen tu %c sang %c\n", A, C);
     ThapHN(n-1,B, A, C);
   }
}
```



Phương trình đệ quy :

$$T(n)=\begin{cases} C1 & n=1 \\ 2T(n-1)+C2 & n>1 \end{cases}$$

CANTHO UNIVERSITY

Ví dụ 4. Phương trình đệ quy của thuật toán tìm kiếm nhị phân dãy n có thứ tự

```
int binarysearch(int x, int *a, int left, int right)
{
   if(left > right)    return -1;
   int mid = (left + right)/2;
   if(x == a[mid])     return mid;
   if (x < a[mid])     return     binarysearch(x,a,left,mid-1)
   else return   binarysearch(x,a,mid+1,right);
 }
```

Phương trình đệ quy : $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2)+1 & n > 1 \end{cases}$

CANTHO UNIVERSITY

Ví dụ 4. Phương trình đệ quy của thuật toán tìm kiếm nhị phân dãy n thứ tự