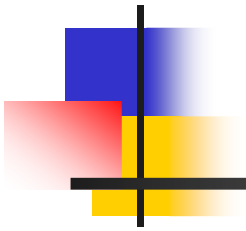
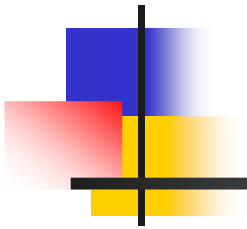


Phần 2

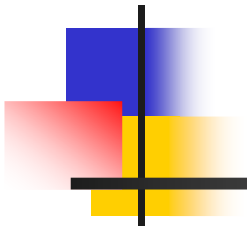
Các thuật toán sắp xếp



Nội dung

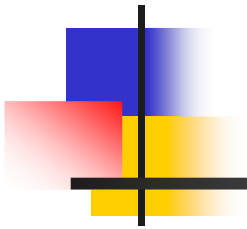
- Giới thiệu về thuật toán sắp xếp
- Thuật toán sắp xếp chọn
- Thuật toán sắp xếp chèn
- Thuật toán sắp xếp nhanh

- Một số thuật toán sắp xếp cơ bản, đánh giá, ứng dụng



Giới thiệu chung

- Các thuật toán sắp xếp được xét đến trong phần này đều dựa trên hai phép toán cơ bản, đó là so sánh hai phần tử với nhau và sắp đặt lại các phần tử trong danh sách.
- Độ phức tạp của các thuật toán sắp xếp dựa trên phép so sánh hai phần tử có thể được xem xét thông qua số phép so sánh (trong một số trường hợp thì số lần đổi chỗ – sắp đặt lại các phần tử cũng được quan tâm).
- Số tối đa các phép so sánh cần dùng để sắp xếp một danh sách có thể coi là trường hợp xấu nhất mà thuật toán gặp phải.



Giới thiệu chung

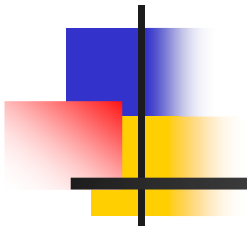
- Nếu ta biểu diễn thuật toán sắp xếp dựa trên phép so sánh bởi một cây quyết định nhị phân thì số phép toán cần thực hiện chính là độ dài của đường đi từ gốc tới lá.
- Số lá của cây quyết định nhị phân này là n!.

- Chiều cao của cây quyết định nhị phân là

$$h \geq \log_2(n!).$$

- Như vậy số phép so sánh tối thiểu phải thực hiện là

$$[\log_2(n!)].$$

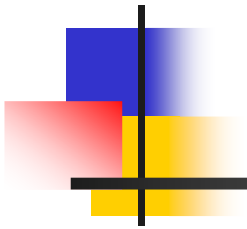


Giới thiệu chung

Nhận xét:

- Dễ dàng thấy được là $n! < n^n$ với mọi $n \geq 1$

  cho nên $[\log_2(n!)] = O(n \log n)$.
- Mặt khác ta cũng thấy là với $n>4$ thì $n! > n^{n/4}$.
- Suy ra: không có thuật toán sắp xếp nào dựa trên phép so sánh lại có đánh giá tốt hơn $O(n \log n)$.



Giới thiệu chung

Chứng minh $n! > n^{n/4}$.

Thật vậy: với n=4 bất đẳng thức là hiển nhiên.

Giả thiết là có $n! > n^{n/4}$ Ta sẽ chứng minh $(n+1)! > (n+1)^{(n+1)/4}$

Xét bất đẳng thức:

$(n+1)n^{n/4} > (n+1)^{(n+1)/4}$,

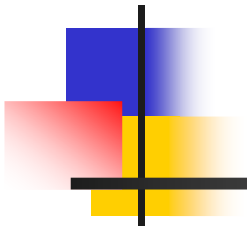
$\Leftrightarrow n^{n/4} > (n+1)^{(n-3)/4}$

$\Leftrightarrow n^n > (n+1)^{n-3}$

$\Leftrightarrow (n+1)^3 > ((n+1)/n)^n$

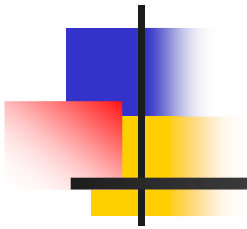
Điều này là hiển nhiên vì

$(n+1)^3 > e > ((n+1)/n)^n$, với mọi n>4.



Selection sort

- **Bài toán**:
- Cho mảng *n* số nguyên X(1), X(2), ..., X(n). Hãy sắp xếp lại mảng này theo thứ tự không giảm.
- Mục này nhằm trình bày thuật toán sắp xếp đơn giản và thông dụng nhất để giải bài toán trên. Thuật toán này được gọi là thuật toán chọn hay sắp xếp tuần tự.

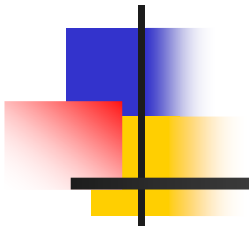


Selection sort

- **Mô tả thuật toán:**
- *Input*: Dãy X(1), X(2), ..., X(*n*) các số nguyên và số các phần tử *n*.
- *Output*: Dãy X(1), X(2), ..., X(*n*) không giảm;

Ý tưởng:

- Chọn phần tử nhỏ nhất Xmin trong các phần tử X(1), X(2), ..., X(n) và hoán vị nó (tức là Xmin) với phần tử đầu tiên X(1);
- Chọn phần tử nhỏ nhất Xmin trong phần còn lại của dãy X(2), X(3), ..., X(n) và hoán vị nó (tức là Xmin) với phần tử thứ hai X(2);
- Tiếp tục thủ tục trên để chọn phần tử X(3), X(4), ..., X(n-1) thích hợp cho dãy.



Selection sort

Chi tiết:

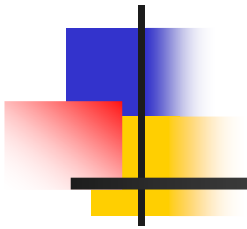
Với $i = 1$ đến $n-1$

Tìm X(k) = min { X(i), X(i+1), X(i+2),..., X(n)};

If k≠i Then Đổi chỗ(X(k), X(i));

Dãy X(1), X(2),..., X(n) đã được sắp không giảm.

```
For i:=1 to n-1 do
    k:=i;
    for j:=i+1 to n do
            If X(j) <   X(k) Then       k:=j;
    End For
    If   k≠i  Then Đổi_chỗ(X(i),X(k));
End For
```



Selection sort

- **Đánh giá:**

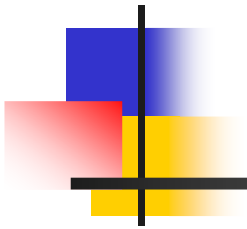
  Dễ dàng thấy được trong bước lặp thứ *i* có n-i phép so sánh và một phép hoán vị;

  Vậy suy ra thuật toán có n(n-1)/2 phép so sánh và n-1 phép hoán vị.

  Thuật toán có độ phức tạp $O(n^2)$.

- **Nhận xét:**

  Trong trường hợp xấu nhất thuật toán này có số phép toán ít hơn so với thuật toán chèn, đặc biệt là số phép hoán vị ít hơn nhiều so với thuật toán sắp xếp chèn. Điều này rất có lợi khi dữ liệu lớn.



Insertion sort

- **Bài toán:** Sắp xếp mảng số nguyên X(1),...X(n) theo thứ tự không giảm.
- **Ý tưởng**

  Xuất phát từ mảng B(1) là mảng có một phần tử

  Giả sử có phần đầu của mảng B(i-1) = <X(1),...,X(i-1)> không giảm (B(i-1) là mảng đã được sắp xếp);

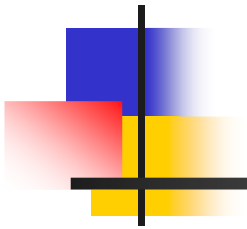
  Bước 1:

  Kiểm tra tới phần tử X(i); Tìm vị trí "thích hợp" của X(i) trong dãy B(i-1) và chèn nó vào đó.

  Sau bước này, dãy mới B(i) = <X'(1),...,X'(i-1),X'(i)> cũng không giảm;

  Bước 2:

  Lặp lại thủ tục trên với X(i+1), ... cho đến khi nào i = n;



Insertion sort

Ví dụ xét dãy:

X = 1, 3, 5, 7, 8, 9, 12, 4, 5, 6, 2

B(7) = <1, 3, 5, 7, 8, 9, 12> ,

i = 8,

j = i-1 = 7

Đổi dần tg = 4 lùi về phía trước; sau các bước

X = 1, 3, 5, 7, 8, 9, -, 12, 5, 6, 2, j=6

X = 1, 3, 5, 7, 8, -, 9, 12, 5, 6, 2, j=5

X = 1, 3, 5, 7, -, 8, 9, 12, 5, 6, 2, j=4

X = 1, 3, 5, -, 7, 8, 9, 12, 5, 6, 2, j=3

X = 1, 3, -, 5, 7, 8, 9, 12, 5, 6, 2, j=2

Tại bước này j=2. Gán tg =4 vào X(j+1) là X(3)

X = 1, 3, **4**, 5, 7, 8, 9, 12, 5, 6, 2

Insertion sort

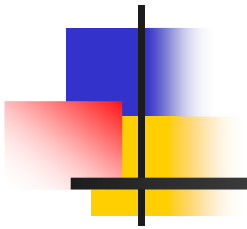
- *Input*: Dãy X(1), X(2), ..., X(*n*)
- *Output*: Dãy X(1), X(2), ..., X(*n*) không giảm;
- Chi tiết:

Bắt đầu với B(1) = < X(1) >;

For i:=2 to n do

1) tg := X(i);
2) j := i - 1;
3) While (j>0) and (tg < X(j)) do
   X(j+1):= X(j);
   j := j-1;
   End While;
4) X(j+1):=tg;

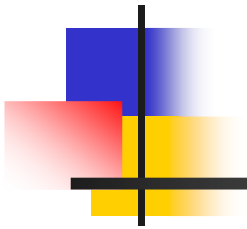
End For;



Insertion sort

Đánh giá thuật toán:

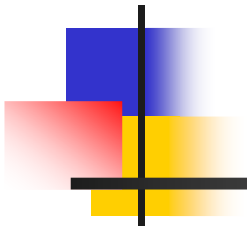
- Ta dễ dàng nhận thấy rằng nếu mảng X đã được sắp xếp thứ tự không giảm thì thuật toán sắp xếp chọn vẫn cần n-1 phép so sánh.
- Tuy nhiên ta dễ dàng nhận thấy rằng nếu dãy hầu như được sắp thì sắp xếp chèn sẽ gần như là tuyến tính.
- Trong trường hợp xấu nhất cần tới n(n-1)/2 phép so sánh và n(n+1)/2 phép hoán vị (thông qua phép gán).
- Vậy độ phức tạp của thuật toán $O(n^2)$
- Thử hình dung rằng mảng X(1),..., X(n) chỉ là khoá của các bản ghi nào đó. Hiển nhiên thời gian chi phí cho các hoán vị này là không nhỏ.



Bubble sort

- **Bài toán**:
- Cho mảng *n* số nguyên X(1), X(2), ..., X(n). Hãy sắp xếp lại mảng này theo thứ tự không giảm.
- **Ý tưởng**

  Về cơ bản thuật toán sắp xếp nổi bọt gần giống với thuật toán sắp xếp tuần tự. Điểm khác biệt là ở chỗ trong mỗi bước lặp để tìm giá trị min trong dãy X(i), X(i+1), X(i+2),..., X(n) là xuất phát từ "dưới lên" so sánh từng cặp một, nếu phần tử đứng dưới X(j+1) mà nhỏ hơn so với phần tử đứng trên X(j) thì đổi chỗ chúng cho nhau.



Bubble sort

- **Thuật toán**

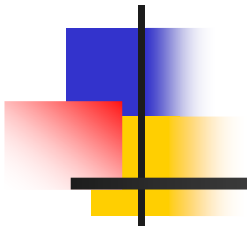
  For i=1 to *n* -1

  for j=*n* to *i*+1

  if X(j) < X(j-1) then Đổi chỗ (X(j), X(j-1))

- Dãy X(1),..., X(n) đã được sắp xếp không giảm.

- **Đánh giá:** Thuật toán sắp xếp nổi bọt cũng cần n(n-1)/2 phép so sánh và n(n-1)/2 phép hoán vị trong trường hợp xấu nhất. Độ phức tạp của thuật toán nổi bọt cũng là $O(n^2)$.



Quick sort

Bài toán:

Cho mảng *n* số nguyên X(1), X(2), ..., X(n). Hãy sắp xếp lại mảng này theo thứ tự không giảm.

Thuật toán

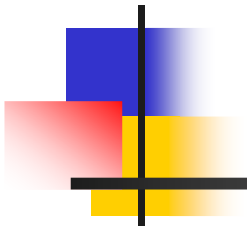
Chọn một phần tử trong dãy X(0,i) nào đó làm khoá;

Xếp:

- các phần tử nhỏ hơn khoá ở phía trước khoá, được đoạn B(1);
- các phần tử lớn hơn khoá ở phía sau khoá, được đoạn B(2);

bằng cách so sánh các phần tử này với khóa và đổi chỗ của chúng cho nhau hoặc với khoá nếu cần thiết.

Kết quả được dãy < B(1), X(0,i), B(2) >



Quick sort

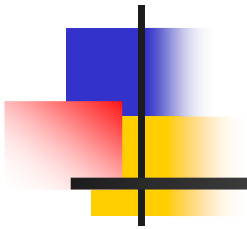
Tiếp tục thực hiện thủ tục trên với hai đoạn B(1) và B(2) được hai dãy con là:

< B(1,1), X(1,i), B(1,2) > và < B(2,1), X(2,i), B(2,2) >

Dãy X được xếp lại là:

< B(1,1), X(1,i), B(1,2), X(0,i), B(2,1), X(2,i), B(2,2) >

Lặp lại thủ tục trên với các dãy con B(1,1) , B(1,2), B(2,1) và B(2,2),…. Cho tới khi dãy con là dãy có 1 phần tử (khi đó dãy con đã được sắp xếp)

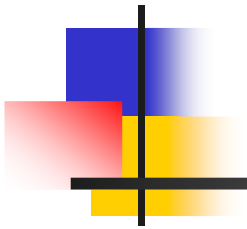


Quick sort

Chi tiết:

```
Procedure SXN(L,R:integer);
If L<R then
      Begin
            key:=x[L];
            i  := L+1;
            j  := R;
            While i<=j Do
                        while (i<=n) and (x[i]<=key) do i:=i+1;
                        while (j>1) and(x[j]>key) Do j:=j-1;
                        If i<j then Doi_cho(x[i],x[j]);

            Doi_cho(x[L],x[j]);
            SXN(L,j-1);
            SXN(j+1,R);
      End;
```



Quick sort

Đánh giá thuật toán:

Kí hiệu

T(k) là số phép toán cần thực hiện để sắp xếp dãy có độ dài k: X(l..r), r=l+k-1;

P(k) là số phép toán cần thiết để phân đoạn dãy có độ dài k thành 2 đoạn con;

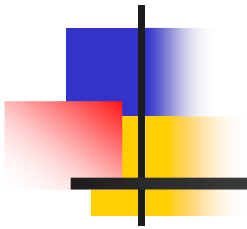
Ta có: T(k) = P(k) + T(j-l)+T(r-j);

Để ý rằng P(k) = c.k với c là hằng số dương nào đó.

Thật vậy, số phép toán so sánh là 2k; Số phép toán cộng và trừ là k; Số phép toán hoán vị 3k/2; (đây là số phép toán cực đại);

vậy tổng số phép toán có thể lên tới 9k/2; (hằng số c=9/2);

Suy ra T(k) = c.k + T(j-l) + T(r-j)

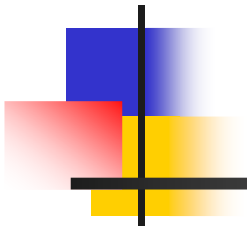


Quick sort

Đánh giá thuật toán:

$T(k) = c.k + T(j-l) + T(r-j)$

- Ta có thể chứng minh được rằng $T(j-l) + T(r-j) \leq T(0) + T(k-1)$.
  Có nghĩa là trường hợp xấu nhất là có một tập trống.
  Khi ấy $T(0)=0$, $T(k-1) \leq c.(k-1) + T(k-2)$.
  Suy ra $T(k) \leq c(k + (k-1) + ... + 1) = c. k(k+1)/2$.
- Nói cách khác thuật toán sắp xếp nhanh có độ phức tạp $O(n^2)$, có nghĩa là không khác gì các thuật toán sắp xếp khác.
- Tuy nhiên, nếu chỉ số j nằm ở chính giữa, nói cách khác đoạn X(l..r) luôn luôn được chia đôi thì khi ấy số phép toán là $O(n\log_2 n)$.



Quick sort

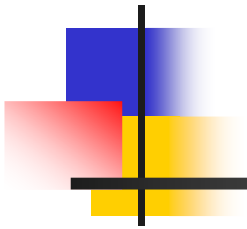
Bài tập:

Xây dựng thuật toán sắp xếp một danh sách nhân sự có họ đệm, tên theo trường khoá là tên. Dữ liệu được nhập vào trên nền tiếng Việt (theo tiêu chuẩn TCVN). Thứ tự được quy định như sau:

thanh: không, huyền, sắc, hỏi, ngã, nặng;

chữ cái: a, â, ă, o, ô, ơ, u, ư, e, ê, d, đ, D, Đ.

Cho dãy X(1),..., X(m) không giảm và dãy Y(1),..., Y(n) không tăng. Viết thuật toán sắp xếp dãy X(1),..., X(m), Y(1),..., Y(n) thành dãy không giảm (không tăng).



Merge sort

Bài toán:

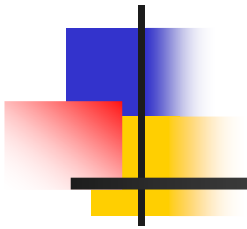
Cho mảng *n* số nguyên X(1), X(2), ..., X(n). Hãy sắp xếp lại mảng này theo thứ tự không giảm.

A. Thủ tục hoà nhập hai mảng đã sắp

Bài toán 1: Cho hai mảng đã được sắp

X = {X(1)≤ ...≤ X(m) } và Y = { Y(1)≤ ...≤Y(n)}

Tạo mảng Z =<Z(1)≤...≤ Z(m+n)> có thứ tự từ 2 mảng X và Y.

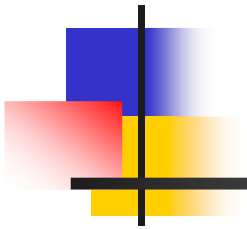


Merge sort

Mô tả thuật toán hoà nhập 1 (HN1):

ý tưởng:

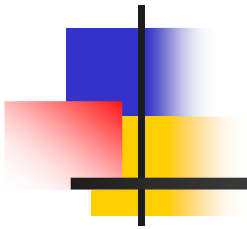
- So sánh hai phần tử nhỏ nhất của hai mảng; Đưa phần tử nhỏ hơn ra mảng đích. Loại phần tử được chọn này ra khỏi mảng đang chứa nó.
- Lặp lại thủ tục này cho đến khi vét hết một trong hai mảng.
- Chuyển toàn bộ phần đuôi của mảng còn lại ra mảng đích.



Merge sort

Chi tiết:

```
i:=1;    j:=1;    k:=1;
While (i ≤ m) & (j ≤ n)
    a)  If X[i] < Y[j]      then  {  Z[k]:=X[i];       inc(i) };
          else      { Z[k]:=Y[j]; inc(j) };
    b)  Inc(k);
If i > m then
          for t:=j to n     do      Z[k+t-j] := Y[t]
else      for t:=i to m     do      Z[k+t-i] := X[t];
```



Merge sort

Bài toán 2:

Cho mảng X gồm hai phần đã được sắp xếp riêng biệt cùng thứ tự (không tăng hoặc không giảm):

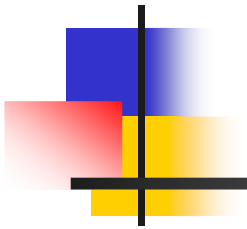
$X(p) \le ... \le X(m)$ và $X(m+1) \le ... \le X(n)$

Sắp xếp mảng X(p,..n).

Mô tả thuật toán hoà nhập 2 (HN2):

ý tưởng:

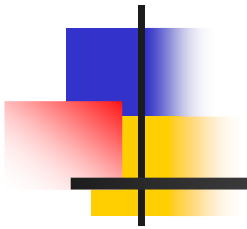
- So sánh hai phần tử nhỏ nhất của hai phần mảng X và đưa phần tử nhỏ hơn ra mảng đích Z. Lướt qua phần tử được chọn này trong phần mảng đang chứa nó.
- Lặp lại thủ tục này cho đến khi vét hết một trong hai phần của mảng.
- Chuyển toàn bộ phần cuối của phần mảng còn lại ra mảng đích.



Merge sort

Chi tiết HN2P(X:mảng vào;p,m,n:integer):

```
i:=p;    j:=m+1;          k:=p;
While (i ≤ m) & (j ≤ n)
    a)  If X[i] < X[j]       then    { Z[k]:=X[i];     inc(i); };
            else                     { Z[k]:=X[j];     inc(j); };
    b.)  Inc(k);
If          i > m    then
            for t:=j to n   do          Z[t] := X[t]
else        for t:=i to m   do          Z[n-m+t] := X[t];
For i:=  p   to  n  do X(i) := Z(i);
```



Merge sort

Thủ tục sắp xếp mảng dựa vào thủ tục hoà nhập

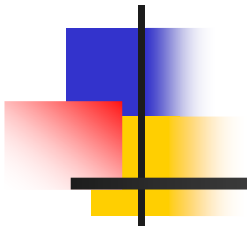
Bài toán: Cho hai mảng X = <X(1), ..., X(n)>

Cho phép sử dụng thêm mảng Y =<Y(1),..., Y(n)> để sắp xếp X.

Mô tả thuật toán

ý tưởng:

- Xét đoạn X(left),....,X(right) tuỳ ý trong đoạn <X(1), ..., X(n)>
- Chia mảng X(left),...,X(right) ra làm 2 phần sao cho lệch nhau ít nhất;
  - Sắp xếp từng phần X(left) ,...X(m) và X(m+1),...,X(right);
  - Sử dụng thủ tục hoà nhập để trộn 2 phần đã được sắp xếp này.
- Khi mảng có 1 phần tử thì mảng đã được sắp xếp



Merge sort

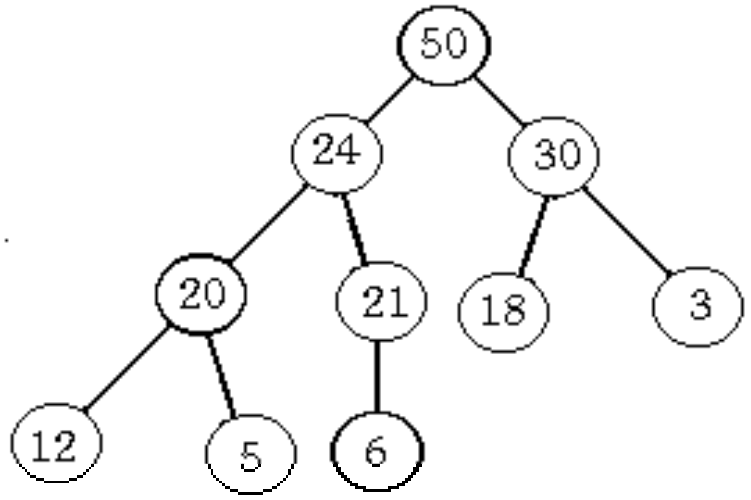
```
Procedure Merge(left, right);
    If     left < right  then
           m:=  (left + right) div 2;
           Merge(left, m);
           Merge(m+1, right);
           HN2(X,left,m,right);
    End if
```

Bài tập: Đánh giá thuật toán. O(nlogn)

Heap sort

1. Tạo đống

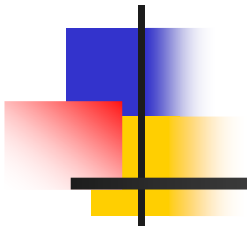
Một cây nhị phân T có chiều cao *d* mà mỗi nút được gán một giá trị khoá được gọi là đống nếu thoả mãn các điều kiện sau đây:



a) Cây T có 2^k nút ở mức k, với $1 \le k \le d-1$. ở mức d-1 các lá đều nằm ở bên trái của các nút trong. Nút trong bên phải nhất mức d-1 có thể có bậc 1 (không có con phải); còn các nút khác có bậc 2.

b) Khoá của mọi nút lớn hơn hoặc bằng khoá ở các nút con của nó (nếu nó có con).

c) Nếu cây T chỉ thoả mãn điều kiện a) ta sẽ tạm gọi là cây T có cấu trúc đống.



Heap sort

Chiến lược sắp xếp vun đống

- Giả thiết mảng X cần sắp xếp có *n* phần tử và mỗi phần tử có một khoá dùng để sắp xếp.
- For i:=1 to n

  Copy khoá từ gốc vào mảng;

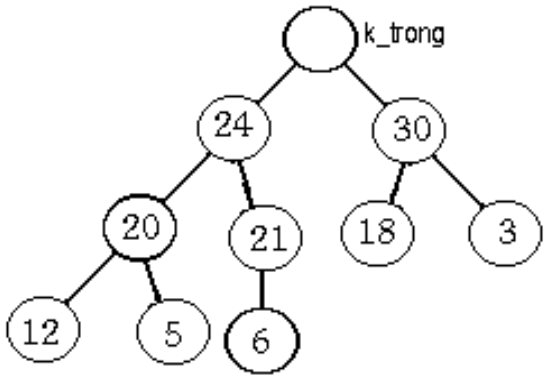
Bài toán 1:

Chuyển một cây "gần đống" về "đống"

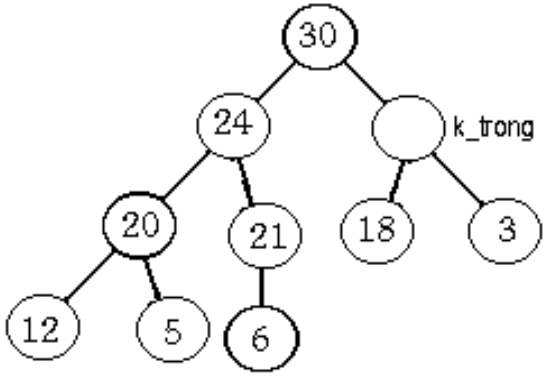
Giá trị ở nút gốc của đống là giá trị lớn nhất trong đống. Sau khi xóa bỏ giá trị ở nút gốc, ta phải sắp xếp lại cây về đống.

Heap sort

- Ta xét một bước của vòng lặp sắp xếp (làm lại đống). Giá trị của khoá tạo gốc (root) của cây ở bước đầu tiên bằng 50. Giá trị này được kí hiệu là max. Sau khi loại bỏ giá trị này, cây được sắp xếp lại (rearrange) để thành đống. Quá trình này được thực hiện dần dần bằng cách tạo ra các "khoảng trống" di chuyển dần theo các mức xuống thấp, cho đến khi chạm đáy (mức d-1).



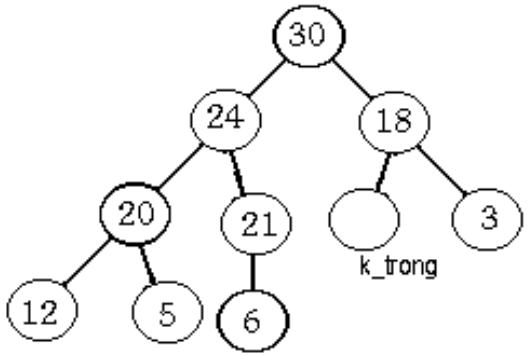
Bước 1. Chọn max =50 là khoá ở gốc, k_trong ở gốc, key = 6



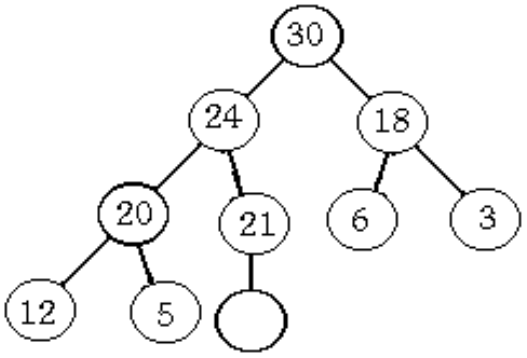
Bước 2. Dịch chuyển dần để tạo đống mới

Heap sort

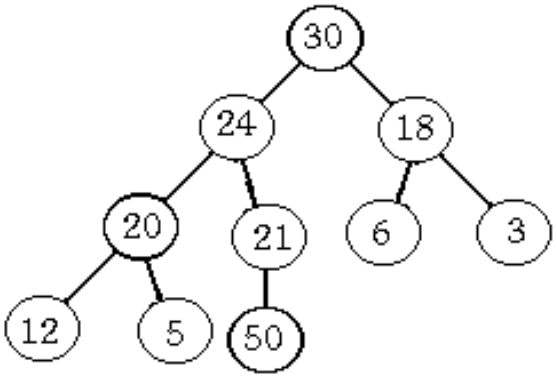
- ở bước 4 ghi key = 6 vào ô có khoảng trống được tạo ra sau cùng.
- Trong bước 5 giá trị max = 50 được lưu lại trong bước trước sẽ ghi vào nút trước đó ghi key. Nút này sẽ không tham gia vào đống trong các bước tiếp theo sau.



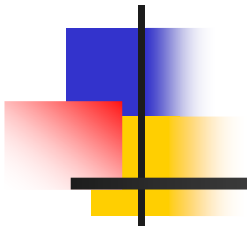
Bước 3. Dịch chuyển dần để tạo đống mới
Gặp lá. Dừng không tìm tiếp



Bước 4. Ghi key = 6 vào ô k_trong cuối cùng



Bước 5. Ghi max = 50 vào ô trước đó đã chọn được key



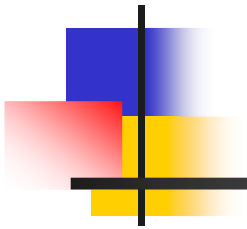
Heap sort

Xét thuật toán FixHeap

Input: **Chỉ số root và giá trị key**

Output: **Đống với các khoá được sắp xếp lại.**

```
Procedure FixHeap(g : Chỉ số gốc;   Key : Giá trị chờ;  m : Số lượng nút của đống mới);
    Begin
          Ô k_trống  có chỉ số := Chỉ số root;
          While  Ô k_trống  chưa phải lá  do
          Begin
                  j := Chỉ số của con có khoá lớn nhất
                  If  Key < X(j)  then   Xác định Ô k_trống mới
                  else Kết thúc vòng lặp
          End;
         Khoá của ô k_trống := Key;
     End;
```



Heap sort

Nhận xét:

Nếu đống có chiều cao *m* thì thuật toán FixHeap có 2*m* phép so sánh.

Heap sort

Bài toán 2: Tạo đống

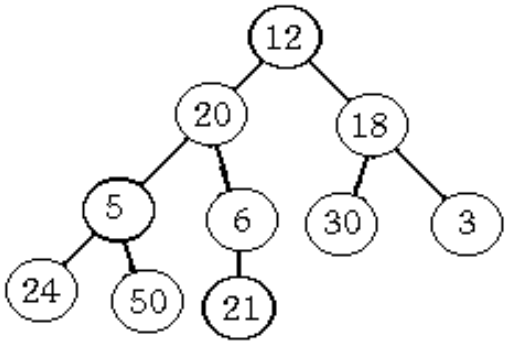
Chuyển một cây có cấu trúc đống về đống

Ý tưởng:

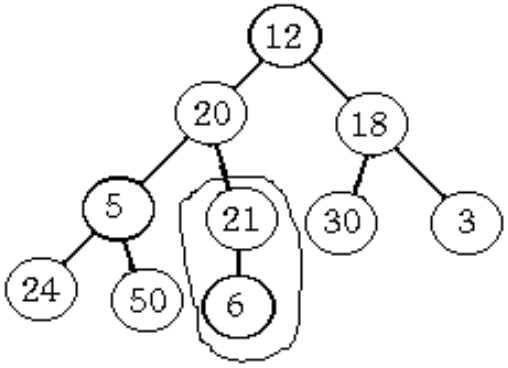
Xét và sắp xếp thành đống từ dưới lên.

Khởi đầu chỉ xét cây con mà gốc là X(n/2) và các con của nó. Tiếp theo là cây có gốc là các phần tử tương ứng X(n/2 – 1), X(n/2 - 2),..., X(1).

Ví dụ

- 

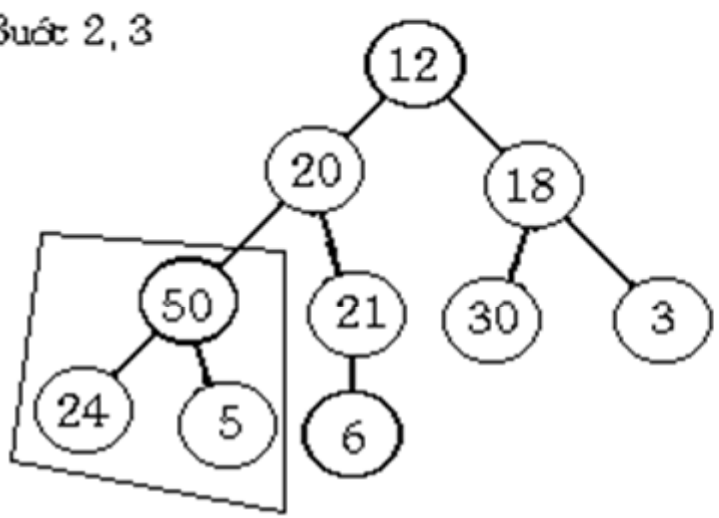
Cây nhị phân hình thành từ mảng X ban đầu



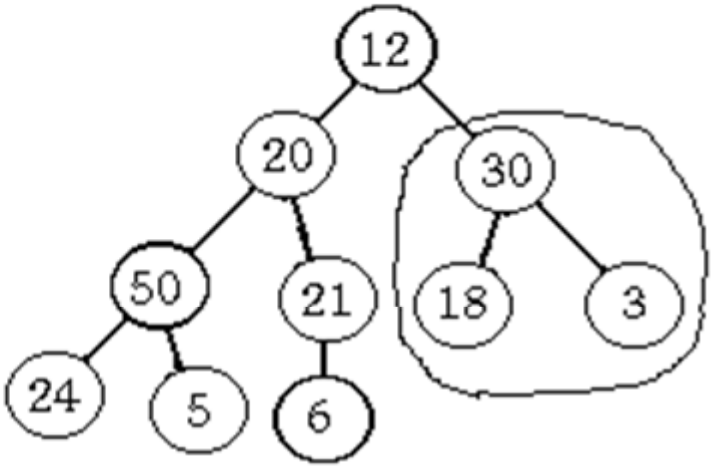
Sắp đống đầu tiên gồm có nút 5 và 10

Heap sort

Bước 2, 3

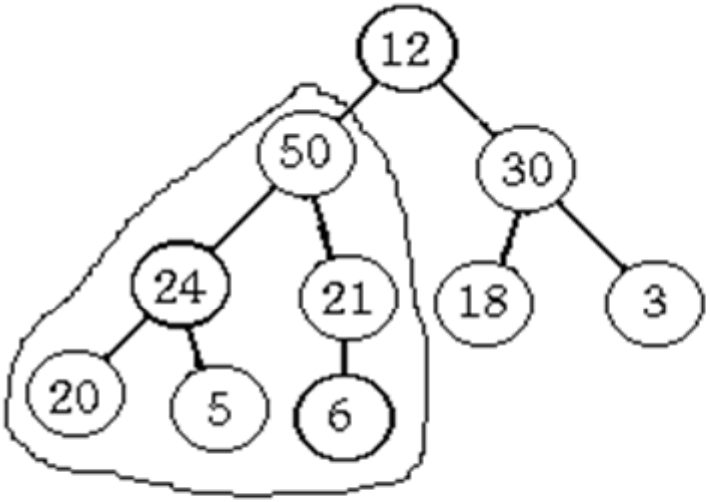


Sắp đống gồm có nút 4 và 8,9

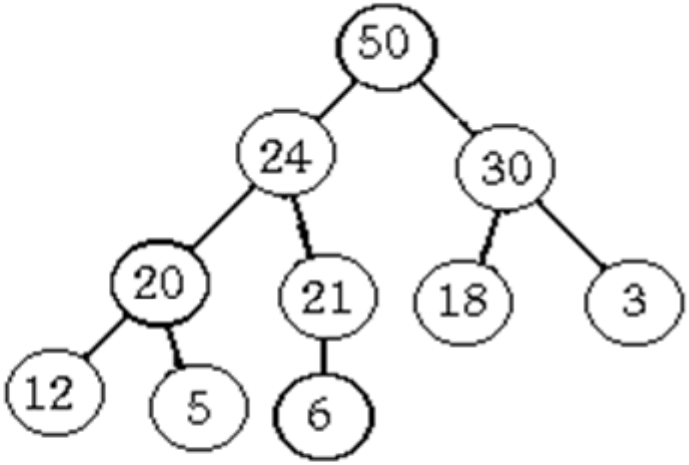


Sắp đống gồm có nút 3 và 6,7

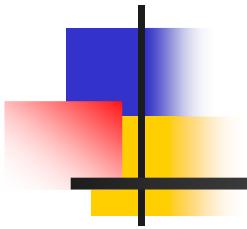
Heap sort



Sấp đống gồm có nút 2 và
4,5,8,9,10



Sấp đống gồm có nút 1 và
2,3,4,5,6,7,8,9,10



Heap sort

Áp dụng thuật toán FixHeap trên mảng.

Cho mảng x[1,..n] chứa giá trị khóa cần sắp xếp

Nếu coi X[i] là giá trị nút cha thì sẽ có 2 con tương ứng có giá trị là (nếu x[i] không phải là nút lá):

con bên trái x[2*i]

con bên phải x[2*i + 1]

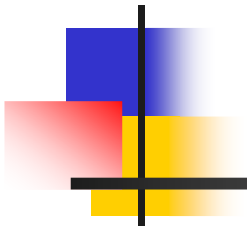
Như vậy:

Các nút cha sẽ là: x[1], x[2], … x[n div 2]

Xét thủ tục:

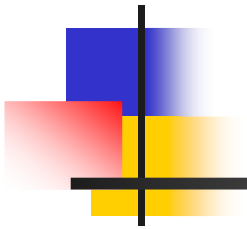
Procedure FixHeap (i:byte; Key:Integer; m : byte);

Thực hiện trên mảng X, với i là giá trị nút gốc của cây, m là chỉ số nút lá cuối cùng của cây



Heap sort

```
Procedure FixHeap (i:byte; Key:Integer; m : byte);
    Var    k,j:Integer; Da_xong : Boolean;
 Begin
    k := i; Da_xong := False;
    While (2*k <= m) and (not Da_xong) do
          Begin
                j := 2*k;
                If (j<m) and (X[j]<X[j+1]) then j:=j+1;
                If Key < X[j] then
                      X[k]:= X[j];
                      k := j;
                Else Da_xong :=true;
          End;
    X[k] := Key;
End;
```



Heap sort

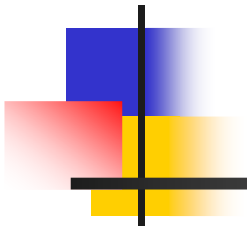
Input :
Mảng X[1..n] of integer chưa được sắp xếp, n>0;
Output :
Mảng X[1..n] đã được sắp xếp không giảm.

```
Procedure Heap_sort(n:byte);
 Begin
```

1. Tạo đống ngay trên mảng X

Vun đống từ dưới lên :

```
For   i:=(n div 2)   downto  1  do    Fix_Heap(i , X[i] , n);
```



Heap sort

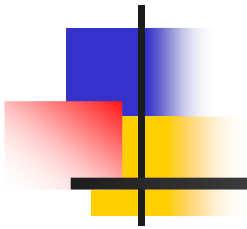
2.Sắp xếp trên đống-mảng X

For heapsize:= n downto 2 do
begin

1. Lưu phần tử lớn nhất trong mảng chưa được sắp xếp. Đây chính là phần tử đầu tiên của đống và cũng là phần tử đầu tiên trên mảng: max := X[1];
2. Vun phần còn lại trên mảng sau khi đã "loại bỏ" một cách hình thức X[1] thành đống. Cũng cần chú ý là các phần tử của mảng-đống khi này đã giảm bớt (sau mỗi vòng lặp):
   Fix_Heap(1, X[heapsize] , heapsize-1);
3. Gán giá trị max vào phần tử cuối cùng của đống (nơi đã lấy khoá làm key khi trước). Sau phép gán này thì phần tử này sẽ không được tính là phần tử của đống. X[heapsize] := max;

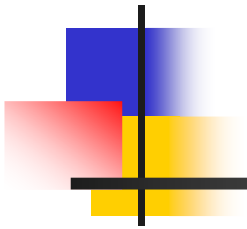
end;

End;



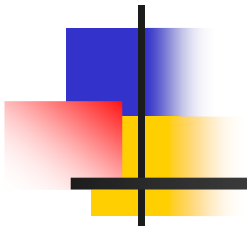
Heap sort

```
Procedure Heap_sort(n:byte);
Begin
        For i:=(n div 2) downto 1 do  Fix_Heap(i,X[i],n);
        For i:= n downto 2 do
                max := X[1];
                Fix_Heap (1, X[i], i-1);
                X[i] := max;
End
```



Heap sort

Đánh giá độ phức tạp của thuật toán: ?



Heap sort

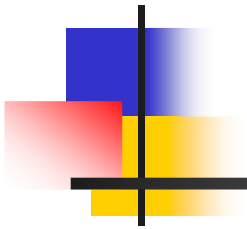
Bài tập:
Cho dãy số nguyên:

4, 9, 7, 2, 6, 8, 9, 1, 3, 5

Sử dụng một trong các thuật toán sắp xếp đã học, sắp xếp dãy số.

1. Mô tả sự dịch chuyển của dãy số
2. Tính số phép so sánh cần phải thực hiện

Viết các thuật toán sắp xếp thể hiện bằng các giải thuật đệ quy



Nội dung

- Giới thiệu về thuật toán sắp xếp
- Thuật toán sắp xếp chọn
- Thuật toán sắp xếp chèn
- Thuật toán sắp xếp nhanh

- Một số thuật toán sắp xếp cơ bản, đánh giá, ứng dụng