Giới thiệu

Các thuật toán sắp xếp

Nội dung trình bày

- *Tiếp cận sắp xếp đơn giản*
  - *Sắp xếp chọn*
  - *Sắp xếp chèn*
  - *Sắp xếp nổi bọt*
- Tiếp cận sắp xếp độ phức tạp O(nlog(n))
  - Sắp xếp theo phân đoạn (Quick sort)
  - Sắp xếp hòa nhập
  - Sắp xếp vung đống
- *Một số tiếp cận khác*
  - *Sắp xếp theo cơ số*
  - *Sắp xếp hòa nhập hai file lớn*

Sắp xếp phân đoạn - quicksort

- Ý tưởng
  - Cho một dãy, chọn một phần tử ở giữa, chia đoạn thành 2 phần
  - Chuyển các phần tử nhỏ, hoặc bằng đến trước, các phần tử lớn hơn về sau
  - Sẽ được nửa đầu bé hơn nửa sau
  - Lặp lại việc chuyển đổi cho các phần tử nửa đầu, và nửa sau đến lúc số phần tử là 1

Sắp xếp phân đoạn – quicksort (t)

- Thuật toán ban đầu là chia: cố gắng chia thành hai đoạn khác nhau
- Trị: thực hiện các thuật toán sắp xếp trên các đoạn con
- Thực hiện kết hợp: thuật toán tự kết hợp kết quả

Sắp xếp phân đoạn – quicksort (t)

- Phân đoạn
  - Chọn một phần tử chốt x (đầu tiên)
  - Duyệt từ vị trí tiếp theo sang phải tìm vị trí phần tử đầu tiên >= x, i
  - Duyệt từ phải sang trái, tìm vị trí phần tử đầu tiên <x, j
  - Nếu i<j thì hoán đổi vị trí
  - Tiếp tục đến lúc j<i

Sắp xếp phân đoạn – quicksort (t)

- Thuật toán: partition
- Input: A[l..r], l,r: đoạn cần phân chia
- Ouput: A[l..r], i chi số phân chia

1. X=a[l]
2. i=l+1;
3. J=r;
4. While (i<j)
   a. While (i<j && a[i]<x) i++
   b. While (j>=i && a[j]>=x) j—
   c. If(i<j) swap(a[i],a[j])
5. Swap(a[l],a[j])
6. Return j;

Sắp xếp phân đoạn – quicksort (t)

- Partition

j	j	0	1	2	3	4	5	6
2	4	3	1	7	8	2	6	9
3	2	3	1	2	8	7	6	9

Sắp xếp phân đoạn – quicksort (t)

- Thuật toán: quicksort
- Input: A[l..r]: đoạn cần sắp xếp
- Ouput: A[l..r] đã sắp xếp

1. If(l>=r) return;
2. i=partition(A,l,r)
3. quicksort(A,l,i-1)
4. quicksort(A,i+1,r)

Sắp xếp phân đoạn – quicksort (t)

A	0	1	2	3	4	5	6
	3	1	7	8	2	6	9
Part	3	1	2	8	7	6	9
	2	1	3	8	7	6	9
Part	2	1		8	7	6	9
	1	2		6	7	8	9
Part	1			6	7		9
				6	7		
					7		
	1	2	3	6	7	8	9

Sắp xếp phân đoạn – quicksort (t)

- Đánh giá độ phức tạp
  - Số phép toán gán giá trị: 3 * n/2 * h
  - Số phép toán so sánh: n*h
  - Số phép toán gán chỉ số: n*h
- Trường hợp xấu nhất: h=n
- Trường hợp trung bình: h = log(n)
- Độ phức tạp trường hợp xấu nhất: $O(n^2)$
- Độ phức tạp trường hợp trung bình: O(nlog(n))

Sắp xếp trộn – mergesort

- Ý tưởng sắp xếp trộn
  - Nếu có hai dãy a và b đã được sắp xếp, tiến hành trộn hai dãy này thành dãy c đã được sắp xếp.
  - Nếu chia nhỏ mảng cần sắp xếp thành các đoạn 1 phần tử thì nó là đoạn được sắp xếp
  - Tiến hành ghép các đoạn nhỏ thành các đoạn lớn đã được sắp xếp

Sắp xếp trộn – mergesort

- Ý tưởng của thao tác trộn
  - Duyệt trên dãy a tại vị trí i
  - Duyệt trên dãy b tại vị trí j
  - Nếu a[i]>b[j] thì thêm b[j] và trong dãy c tăng biến j ngược lại thêm a[i] vào dãy và tăng biến i
  - Nếu một trong hai dãy hết trước tiến hành đưa toàn bộ dãy còn lại vào trong dãy c
  - Áp dụng trong trường hợp a, b là hai đoạn của mảng
    - a[l..t], a[t+1..r]
    - c[l..r]
  - Để thuận tiện trong xử lý tiến hành chuyển mảng đã sắp xếp về mảng a

Sắp xếp trộn – mergesort

- Thuật toán trộn – merge
  - Input: a[l..t], a[t+1..r] đã được sắp xếp không giảm
  - Ouput: a[l..r] được sắp xếp không giảm

1. i=l
2. j=t+1
3. p=l;

Sắp xếp trộn – mergesort

- Thuật toán trộn (t)

4. while (i<=t && j<=r)

a. if(a[i]<a[j])

c[p]=a[i]

i++

b. Else

c[p]=a[j];

j++

c. p++

Sắp xếp trộn – mergesort

- Thuật toán trộn (t)

5. while (i<=t)

        c[p]=a[i]

        i++

        p++

6. while (j<=r)

        c[p]=a[j]

        j++

        p++

7. for (i=l; i<=r ;i++)

            a[i]=c[i];

Sắp xếp trộn – mergesort

	p	i	j	0	1	2	3	4	5	6
a		0	4	1	3	7	8	2	6	9
c	0			1						9
a		1	4		3	7	8	2	6	9
c	1			1	2					
a		1	5		3	7	8		6	9
c	2			1	2	3				
a		2	5			7	8		6	9
c	3			1	2	3	6			
a		2	6			7	8			9
c	4			1	2	3	6	7		
a		3	6				8			9
c	5			1	2	3	6	7	8	
a		4	6							9
c	6			1	2	3	6	7	8	9
a				1	2	3	6	7	8	9

Sắp xếp trộn – mergesort

- Thuật toán sắp xếp trộn mergesort
- Input: a[l..r]
- Ouput: a[l..r] đã được sắp xếp

1. if(l>=r) return ;
2. t=(l+r)/2
3. mergesort(l,t);
4. mergesort(t+1,r);
5. merge(a[l..t],a[t+1..r);

Sắp xếp trộn – mergesort

- Thuật toán sắp xếp trộn mergesort

0	1	2	3	4	5	6
3	1	7	8	2	6	9
3	1	7	8	2	6	9
3	1	7	8	2	6	9

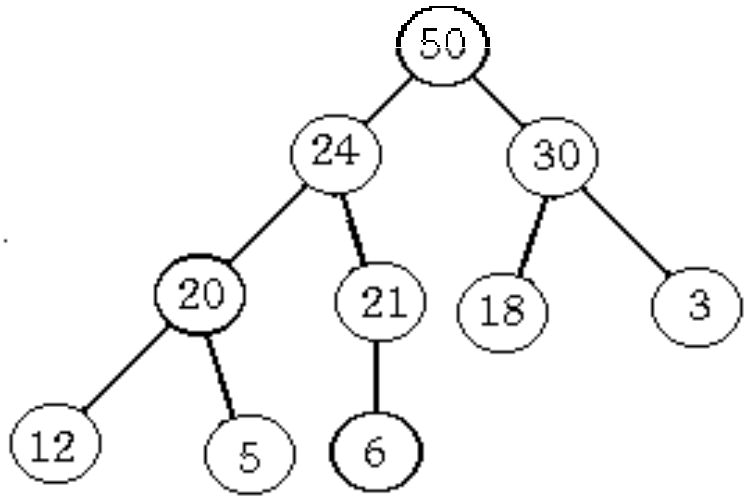
1	3	7	8	2	6	9
1	3	7	8	2	6	9
1	2	3	6	7	8	9

Sắp xếp trộn – mergesort

- Đánh giá độ phức tạp
  - Số phép so sánh: n*log(n)
  - Số phép gáp: 2*n*log(n)
  - Số phép gán chỉ số: 2*n
  - Độ phức tạp phép toán: O(nlog(n))

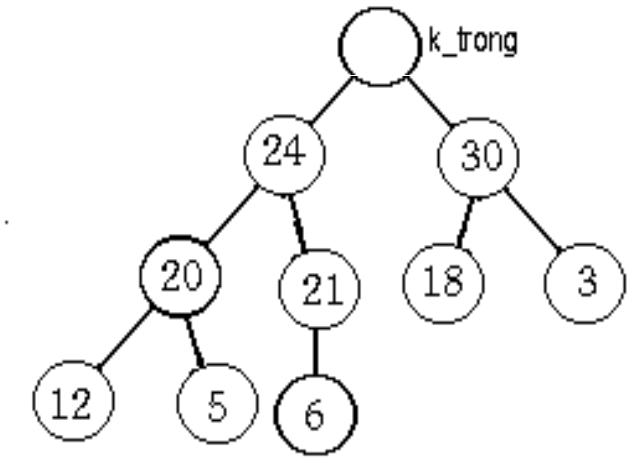
Sắp xếp vun đống – heapsort

- Dựa trên khái niệm cây nhị phân
  - Nếu xây dựng được cây nhị phân cực đại: phần tử trên cha lớn hơn hai con của nó
  - Xây dựng thuật toán duy trì đặc điểm này của cây

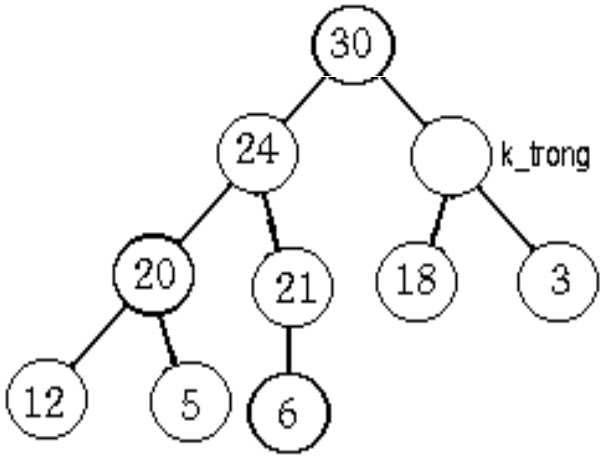


Sắp xếp vun đống – heapsort (t)

- Xây dựng thuật toán
  - Lấy phần tử lớn nhất khỏi cây, tiến hành xây dựng lại cây, với phần tử ở cuối được đưa lên đỉnh



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



Bước 2. Dịch chuyển dần để tạo đống mới

Sắp xếp vun đống – heapsort (t)

- Biểu diễn dữ liệu
  - Mô tả cây có nhiều cấu trúc khác nhau nhưng trong bài toán này xét cấu trúc cây được lưu trên mảng
  - Nếu coi rằng tại vị trí a[0] là đỉnh của cây
  - a[1] sẽ là con trái, a[2] sẽ là con phải
  - A[3] con trái của a[1], a[4] con phải a[1]
  - a[5] con trái của a[2], a[6] con phải của a[2]
  - Con trái của phần tử a[i] là a[i*2+1]
  - Con phải của phần tử a[i] là a[i*2+2]

0	1	2	3	4	5	6	7	8	9
50	24	30	20	21	18	3	12	5	6

Sắp xếp vun đống – heapsort (t)

- Ý tưởng xây dựng đống
  - Một phần tử ban đầu là đống thỏa mãn điều kiện
  - Thêm một phần tử vào đống đã có
    - Thêm phần tử vào lá cuối của đống
    - Nếu phần tử đó lớn hơn cha của nó thì đảo vị trí
    - Lặp lại quá trình cho đến khi không còn cặp cha và con không đúng thứ tự

Sắp xếp vun đống – heapsort (t)

- Thuật toán buildheap
  - Input: a[0..N-1]
  - Ouput: a[0..N-1] tuân thủ qui tắc

1. for(i=1 -> N-1)

    a. x=a[i];

    b. r=i;

    c. while (r>0)

        if(a>a[(r-1)/2])

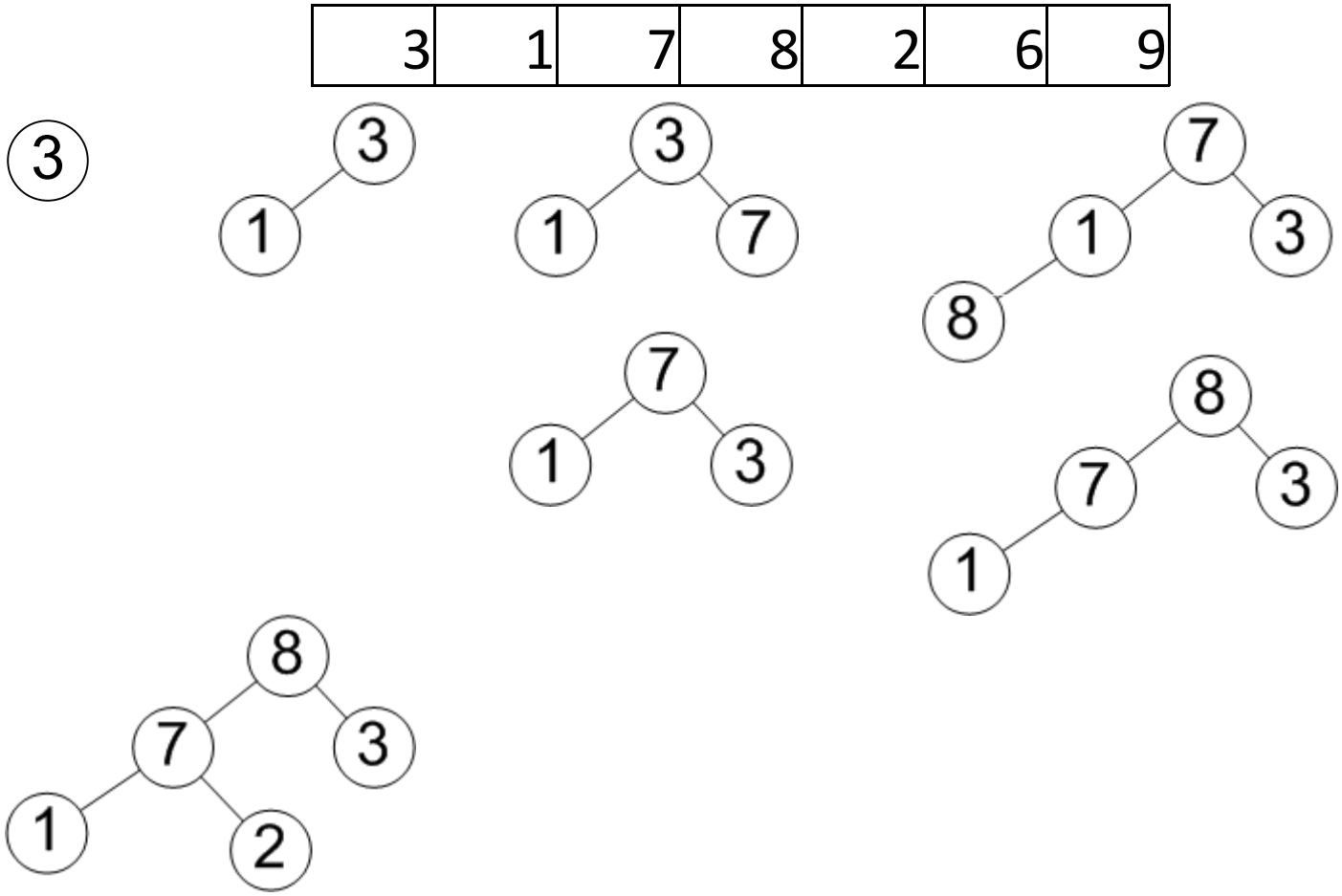
            A[r]=a[(r-1)/2];

            R=(r-1)/2;

    d. a[r]=x;

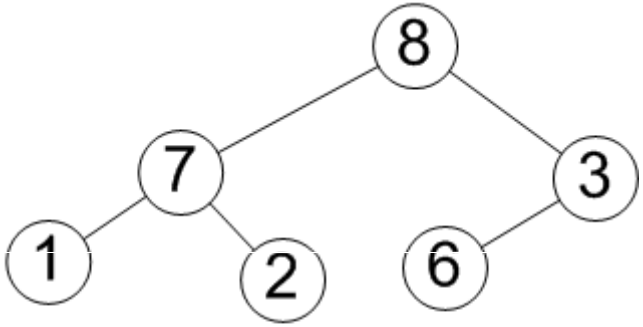
Sắp xếp vun đống – heapsort (t)

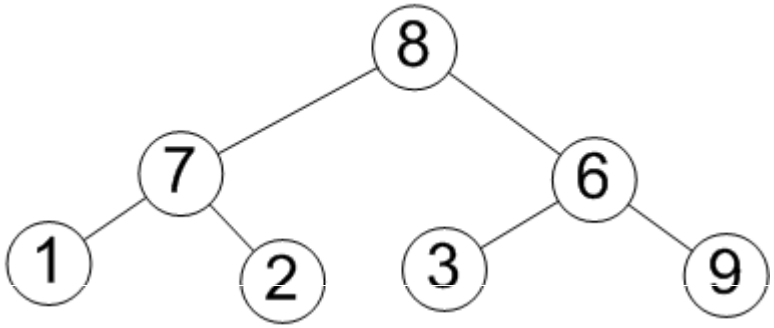
- Thử nghiệm

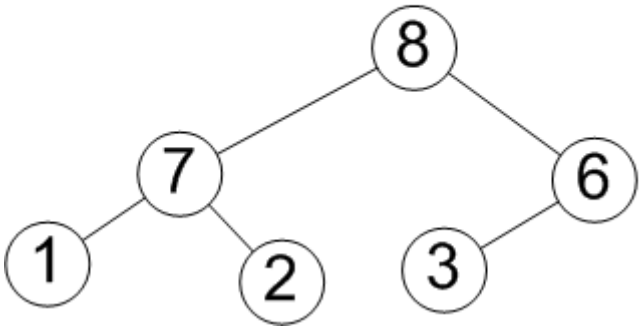


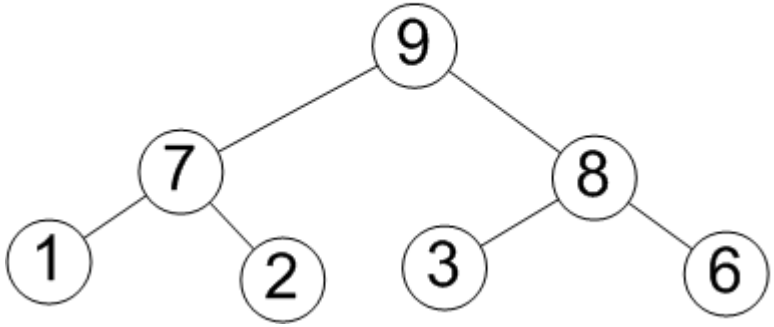
Sắp xếp vun đống – heapsort (t)

- Thử nghiệm









Sắp xếp vun đống – heapsort (t)

- Thử nghiệm

i	r	0	1	2	3	4	5	6
1		3	1	7	8	2	6	9
2	2	3	1	7	8	2	6	9
2		7	1	3	8	2	6	9
3	3	7	1	3	8	2	6	9
3	1	7	8	3	1	2	6	9
3		8	7	3	1	2	6	9
4		8	7	3	1	2	6	9
5	5	8	7	3	1	2	6	9
5		8	7	6	1	2	3	9
6	6	8	7	6	1	2	3	9
6	3	8	7	6	9	2	3	1
6	1	8	9	6	7	2	3	1
6		9	8	6	7	2	3	1

Sắp xếp vun đống – heapsort (t)

- Đánh giá phức tạp xây dựng đống
  - Số phép gán: N + Nlog(N)
  - Số phép so sánh: Nlog(N)
  - Số phép gán chỉ số: Nlog(N)
  - Độ phức tạp thuật toán: Nlog(N)

Sắp xếp vun đống – heapsort (t)

- Ý tưởng sắp xếp
  - Sau khi có đống thỏa mãn điều kiện
  - Lặp từ N-1 đến 1
    - Giá trị phần tử đang xét là x
    - Nhận giá trị từ đỉnh vào phần tử đang xét
    - Tiến hành chuyển xuống
      - Nếu x lớn hơn hai con của đỉnh thì gán x vào đỉnh, kết thúc
      - Nếu x bé hơn phần tử lớn nhất trong hai con của đỉnh thì đỉnh nhận giá trị đó và chuyển đến xét đỉnh con đó

Sắp xếp vun đống – heapsort (t)

- Thuật toán downheap
- Input: A[0..N-1] đống thỏa mãn điều kiện
- Output: A[0..N-1] đã sắp xếp

1. for(i=N-1->1)
   a. x=a[i]
   b. r=0
   c. j=r*2+1
   d. a[i]=a[0];

Sắp xếp vun đống – heapsort (t)

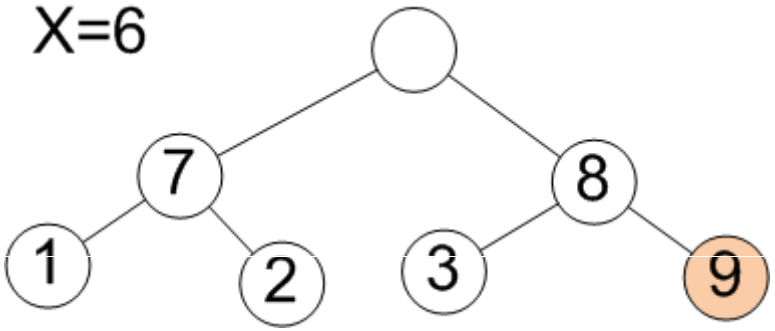
- Thuật toán downheap(t)
- Input: A[0..N-1] đống thỏa mãn điều kiện
- Output: A[0..N-1] đã sắp xếp

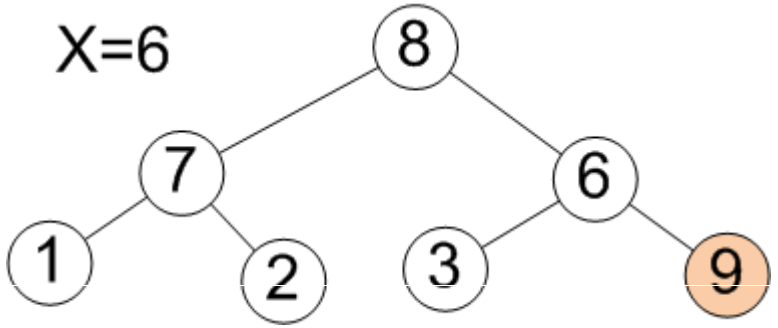
e. while (j<i)

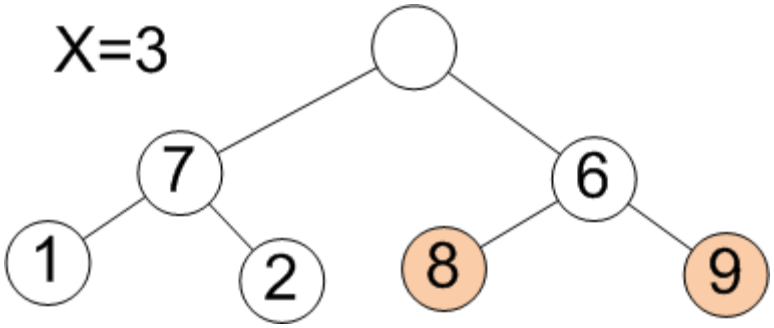
```
    if(j+1<i && a[j+1]>a[j]) j++;
    if(x<a[j])
            a[r]=a[j]
            r=j;
            j=r*2+1
    else
    break;
```

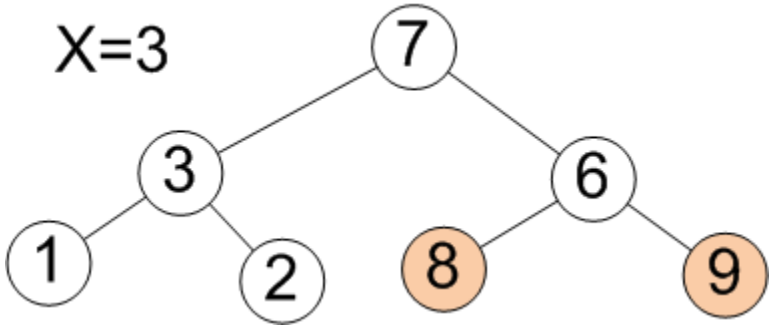
f. a[r]=x;

Sắp xếp vun đống – heapsort (t)

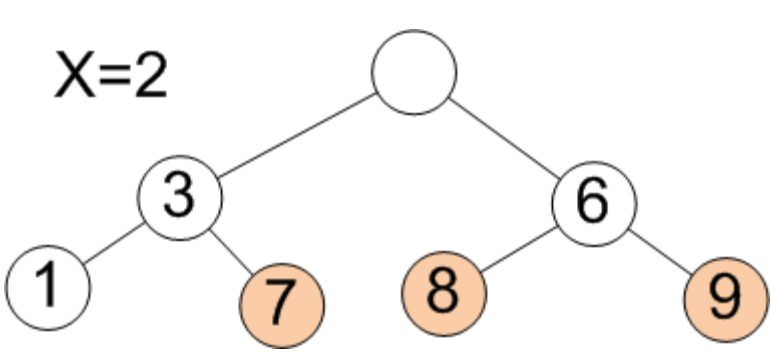


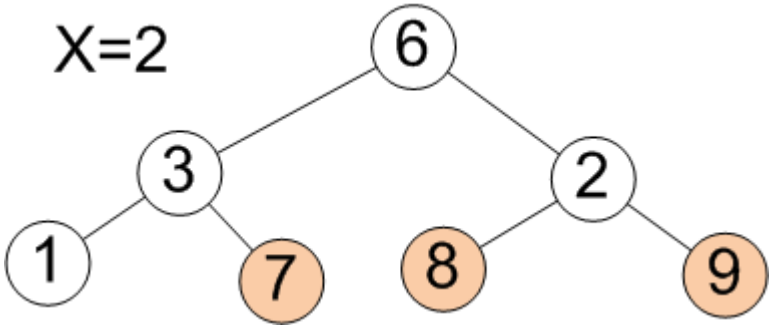


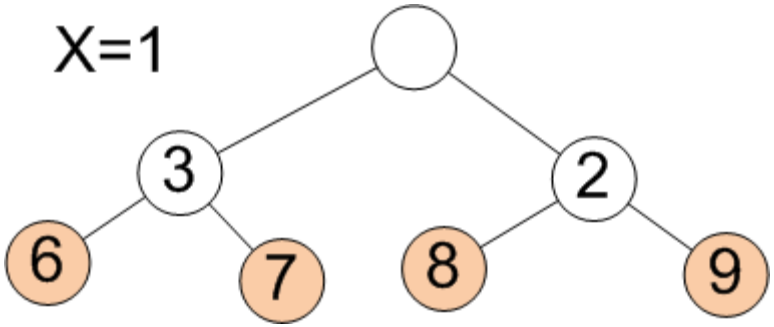


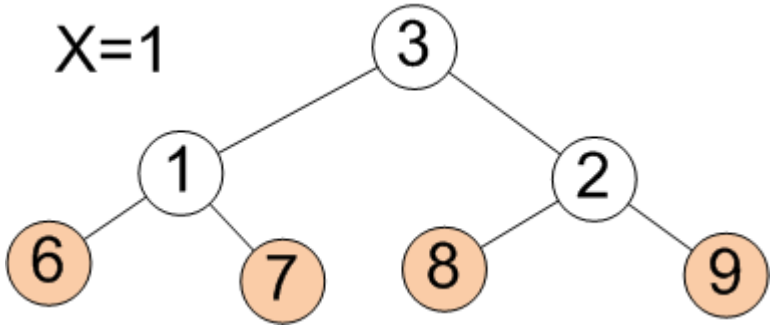


Sắp xếp vun đống – heapsort (t)

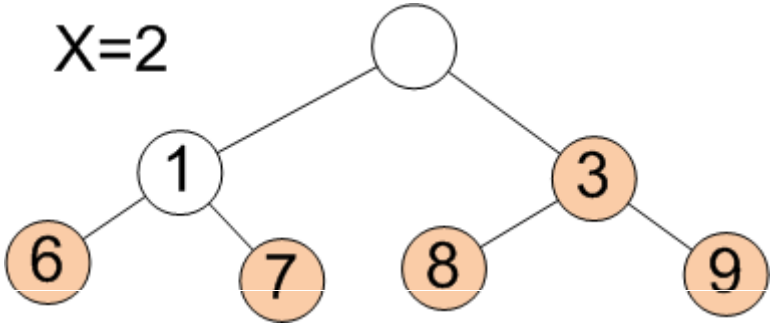


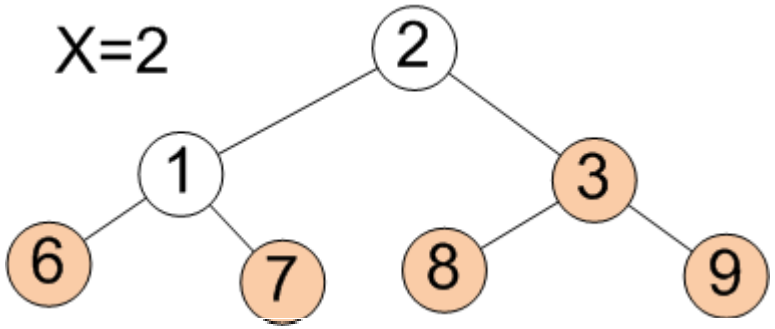


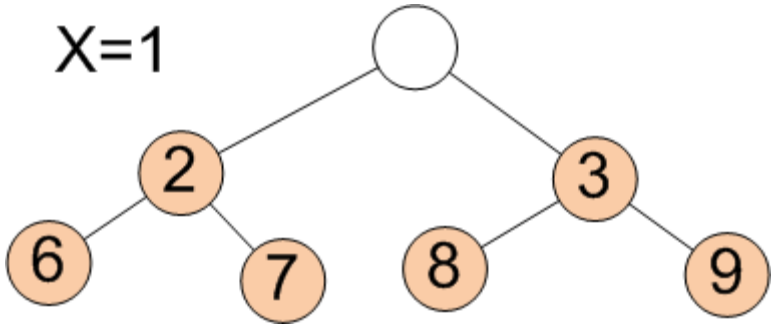


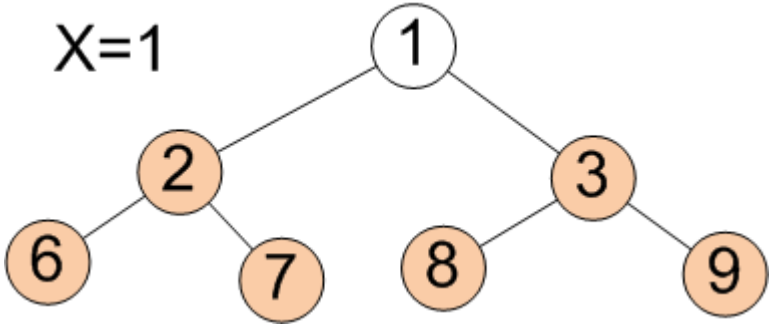


Sắp xếp vun đống – heapsort (t)









1	2	3	6	7	8	9

Sắp xếp vun đống – heapsort (t)

x	i	r	j	0	1	2	3	4	5	6
1	6	0	1		8	6	7	2	3	9
1	6	1	3	8		6	7	2	3	9
1	6	3		8	7	6	1	2	3	9
3	5	0	1		7	6	1	2	8	9
3	5	1		7	3	6	1	2	8	9
2	4	0	2		3	6	1	7	8	9
2	4	2		6	3	2	1	7	8	9
1	3	0	1		3	2	6	7	8	9
1	3	1		3	1	2	6	7	8	9
2	2	0		2	1	3	6	7	8	9
1	1	0		1	2	3	6	7	8	9

Sắp xếp vun đống – heapsort (t)

- Độ phức tạp thuật toán
- Số phép gán: nlog(n)
- Số phép so sánh: 2*nlog(n)
- Số phép toán chỉ số: 2*nlog(n)
- Độ phức tạp thuật toán: nlog(n)

Sắp xếp vun đống – heapsort (t)

- Thuật toán sắp xếp heapsort
- Input:a[0..N-1]
- Output: a[0..N-1] đã được sắp xếp

1. buildheap(a[0..N-1])
2. downheap(a[0..N-1])

Sắp xếp vun đống – heapsort (t)

- Đánh giá độ phức tạp
- Tổng cộng của hai thuật toán trên

Sắp xếp vun đống – heapsort (t)

- Sắp xếp phân đoạn: quicksort
- Sắp xếp trộn: mergesort
- Sắp xếp vun đống: heapsort

Bài tập

- Cài đặt thuật toán trên ngôn ngữ lập trình và chạy thử
- Thử nghiệm các thuật toán sắp xếp để đạt được dãy không tăng với các bộ dữ liệu sau
- 1 2 3 4 5 6 7 8
- 8 7 6 5 4 3 2 1
- 5 2 6 4 8 3 7 1
- Nhận xét trong trường hợp nào thuật toán nào thực hiện nhiều thao tác nhất (đâu là trường hợp tốt nhất, xấu nhất) và số thao tác trong trường hợp đó