



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Data Structures & Algorithms

SẮP XẾP (TT)



Nội dung

1. QuickSort
2. Merge Sort
3. Heap Sort

Các thuật toán sắp xếp đơn giản

- Bubble Sort
- Selection Sort
- Insertion Sort
- Interchange Sort
- Shake Sort

$O(N^2)$

Các thuật toán sắp xếp phức tạp hơn

- Quick Sort
- Shell Sort
- Merge Sort
- Heap Sort

$O(N \log N)$

Quick Sort

Bài toán: Cho mảng một chiều các số nguyên và giá trị x. Hãy tách mảng a ban đầu thành 2 mảng b và c sao cho mảng b chỉ chứa các giá trị nhỏ hơn x, mảng c chứa các giá trị lớn hơn x.

Quick Sort

```

11. void Split(int a[],int n,
               int x,
               int b[],int &k,
               int c[],int &l)
12. {
13.     k = l = 0;
14.     for(int i=0;i<n;i++)
15.         if(a[i]<x)
16.             b[k++] = a[i];
17.         else
18.             if(a[i]>x)
19.                 c[l++]=a[i];
20. }
```

Quick Sort

Thuật toán quick sort chia không gian cần sắp xếp thành 2 không gian con là không gian con 1 và không gian con 2. Không gian con 1 là không gian mà tất cả các phần tử thuộc không gian này đều nhỏ hơn tất cả các phần tử thuộc không gian con 2.

+ Nếu không gian con thứ nhất có nhiều hơn một phần tử thì sắp xếp không gian con này bằng thuật toán Quick Sort.

+ Nếu không gian con thứ hai có nhiều hơn một phần tử thì sắp xếp không gian con này bằng thuật toán Quick Sort.

Quick Sort – Ý tưởng

Giải thuật QuickSort sắp xếp dãy a_1, a_2, \dots, a_N dựa trên việc phân hoạch dãy ban đầu thành 3 phần :

- Phần 1: Gồm các phần tử có giá trị bé hơn x
 - Phần 2: Gồm các phần tử có giá trị bằng x
 - Phần 3: Gồm các phần tử có giá trị lớn hơn x
- với x là giá trị của một phần tử tùy ý trong dãy ban đầu.

Quick Sort – Ý tưởng

Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 đoạn:

- 1. $a_k \leq x$, với $k = 1 \dots j$
- 2. $a_k = x$, với $k = j+1 \dots i-1$
- 3. $a_k \geq x$, với $k = i \dots N$

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

Quick Sort – Ý tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn 1, 3 được sắp.

Quick Sort – Ý tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn 1, 3 được sắp.
- Để sắp xếp các đoạn 1 và 3, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày ...

Quick Sort – Ý tưởng

- **Bước 1:** Nếu $left \geq right$ //dãy có ít hơn 2 phần tử
Kết thúc; //dãy đã được sắp xếp
- **Bước 2:** Phân hoạch dãy $a_{left} \dots a_{right}$ thành các đoạn: $a_{left} \dots a_j, a_{j+1} \dots a_{i-1}, a_i \dots a_{right}$
 - Đoạn 1 $\leq x$*
 - Đoạn 2: $a_{j+1} \dots a_{i-1} = x$*
 - Đoạn 3: $a_i \dots a_{right} \geq x$*
- **Bước 3:** **Sắp xếp đoạn 1:** $a_{left} \dots a_j$
- **Bước 4:** **Sắp xếp đoạn 3:** $a_i \dots a_{right}$

Quick Sort – Ý tưởng

- **Bước 1:** Chọn tùy ý một phần tử $a[k]$ trong dãy là giá trị mốc ($l \leq k \leq r$):
 $x = a[k]; i = l; j = r;$
- **Bước 2:** Phát hiện và hiệu chỉnh cặp phần tử $a[i], a[j]$ nằm sai chỗ :

 - **Bước 2a:** Trong khi ($a[i] < x$) $i++$;
 - **Bước 2b:** Trong khi ($a[j] > x$) $j--$;
 - **Bước 2c:** Nếu $i < j$ $Swap(a[i], a[j]);$

- **Bước 3:** Nếu $i < j$: Lặp lại Bước 2.
Ngược lại: Dừng

Quick Sort – Ý tưởng

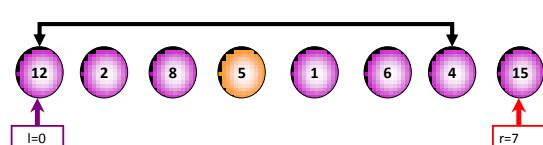
- Chọn 1 phần tử bất kỳ trong mảng làm **nút trục**, xác định vị trí hợp lệ của nút này trong mảng (vị trí **pivot**).
- Phân hoạch các phần tử còn lại sao cho từ vị trí 0 đến pivot-1 đều có giá trị nhỏ hơn hoặc bằng nút trục, từ vị trí pivot+1 đến n-1 lớn hơn nút trục.
- Quá trình tiếp tục như thế với 2 mảng con này.

Quick Sort – Ví dụ

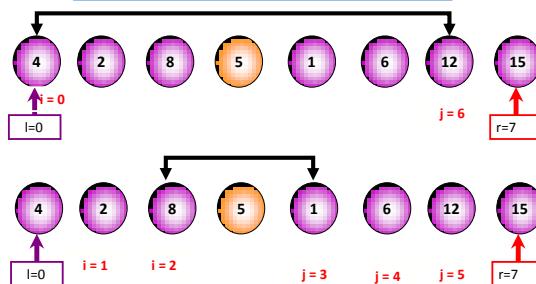
- Cho dãy số a:

12 2 8 5 1 6 4 15

Phân hoạch đoạn $l=0, r=7$:



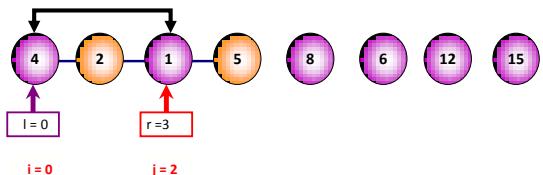
Quick Sort – Ví dụ



Quick Sort – Ví dụ

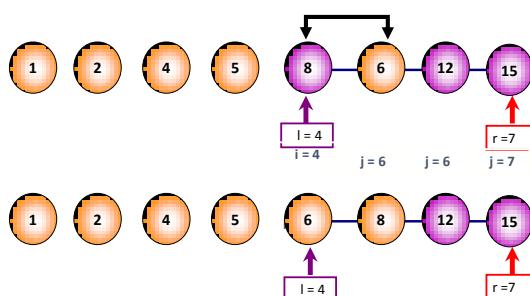
Quick sort – Ví dụ

- Phân hoạch đoạn $l=0, r=2$:



Quick Sort – Ví dụ

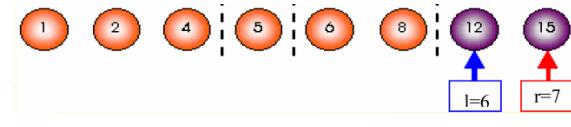
- Phân hoạch đoạn $l=4, r=7$:



Quick Sort – Ví dụ

Quick sort – Ví dụ

- Phân hoạch đoạn $l=6, r=7$:



Quick Sort – Giải thuật

Input: - mảng a: $a[0], a[1], \dots, a[n-1]$ chưa sắp xếp giữa 2 vị trí left và right
- 2 giá trị left, right

Output: mảng a đã sắp xếp giữa 2 vị trí left và right

Các bước thực hiện:

- | | | |
|--|--|----------------------------------|
| 1. if (left >= right) | Kết thúc giải thuật | // điều kiện dừng, đây có thể là |
| | | 2 phần tử |
| 2. if (left < right) | | // bước đệ quy |
| 2.1. Phân hoạch dãy $a_{left} \dots a_{right}$, thành các dãy con | | |
| / phân hoạch dãy làm 3 phần: | dãy con 1: $a[i] \rightarrow a[pivot]$, $i < pivot$ | |
| | nút làm trục: $a[pivot]$ | |
| | dãy con 2: $a[i] \rightarrow a[pivot], i > pivot$ * | |
| 2.2. Gọi đệ quy: QuickSort ($a[], left, pivot - 1$) | | |
| 2.3. Gọi đệ quy: QuickSort ($a[], pivot+1, right$) | | |

Partition – Phân hoạch

- ✓ Input: dãy các nút từ vị trí left đến right
 - ✓ Output: đổi chỗ các nút sao cho các nút nhỏ hơn hoặc bằng nút trục đưa về trước nút trục, các nút lớn hơn đưa về sau nút trục
 - ✓ Các bước thực hiện:
 - Chọn tùy ý một phần tử $a[k]$ trong dãy là giá trị mốc ($left \leq k \leq right$):
 $x = a[k]; i = left; j = right;$
 - Phát hiện và hiệu chỉnh cặp phần tử $a[i], a[j]$ nằm sai chỗ:
// xử lý 2 hướng quét để đổi chỗ các cặp sai vị trí

Partition – Phân hoạch

- 2.1. Quét hướng từ left: i xuất phát từ left và tăng dần, dừng lại khi gặp nút lớn hơn hay bằng nút trục, ghi nhận vị trí i lúc này
 - 2.2. Quét hướng từ right: j xuất phát từ right và giảm dần, dừng lại khi gặp nút nhỏ hơn hay bằng nút trục, ghi nhận vị trí j lúc này.
 - 2.3. if ($i \leq j$) Đổi chỗ 2 nút tại 2 vị trí i và j, tăng i và giảm j.

// cứ tiếp tục quét theo 2 hướng và đổi chỗ các cặp nút sai vị trí.

 3. Nếu $i \leq j$: Lặp lại Bước 2.

Nhược điểm: Dùng

Quick Sort – Cài đặt

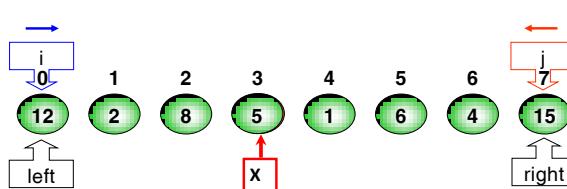
```

void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2];           // chọn phần tử chính giữa làm trục
    i = left; j = right;
    while(i < j)                   // phân hoạch
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Doichu(a[i],a[j]);
            i++; j--;
        }
    }
    if(left < j) QuickSort(a, left, j); // gọi đệ quy
    if(i < right) QuickSort(a, i, right);
}

```

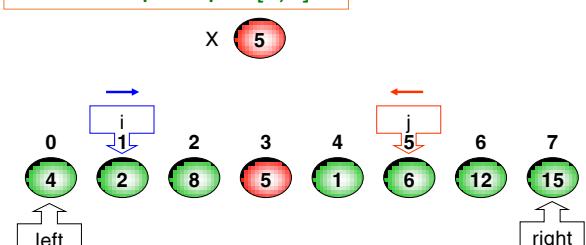
Quick Sort – Ví dụ

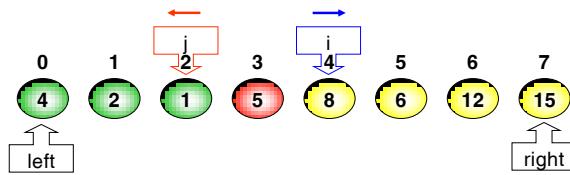
➤ Phân hoạch đoạn [0,7]



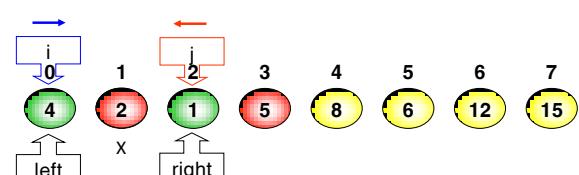
Quick Sort – Ví dụ

✓ Phân hoạch đoạn [0,7]

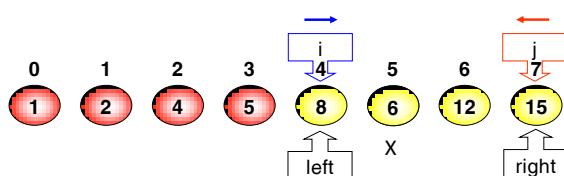


Quick Sort – Ví dụ**➤ Phân hoạch đoạn [0,2]**

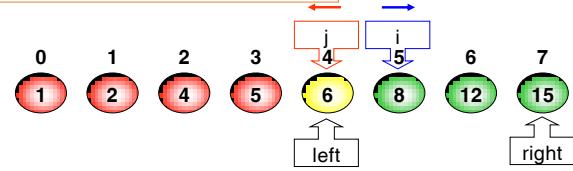
25

Quick Sort – Ví dụ**➤ Phân hoạch đoạn [0,2]**

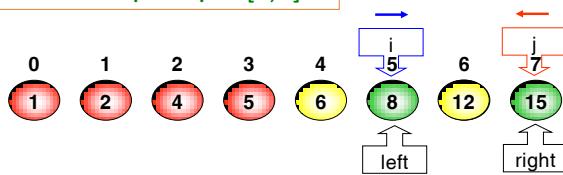
26

Quick Sort – Ví dụ**➤ Phân hoạch đoạn [4,7]**

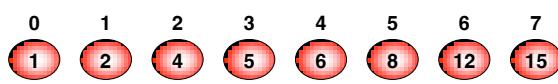
27

Quick Sort – Ví dụ**➤ Phân hoạch đoạn [5,7]**

28

Quick Sort – Ví dụ**➤ Phân hoạch đoạn [5,7]**

29

Quick Sort – Ví dụ

30

Quick Sort – Đánh giá

Hiệu quả thực hiện của giải thuật QuickSort phụ thuộc vào việc chọn giá trị mốc

Trường hợp	Số lần so sánh	Số lần đổi chỗ
Tốt nhất	Nút trục lúc nào cũng ở giữa dãy, 2 dây con lúc nào cũng cân đối nhau	- Cần log ₂ n lần phân hoạch $C(n) = (1*n) + (2*n/2) + (4*n/4) + \dots + (n*n/n) = n\lg n$
Xấu nhất	Nút trục lúc nào cũng ở đầu hay cuối dây, 2 dây con lệch nhau và suy biến thành 1 dây con có số phần tử bắt đì 1	- Cần n lần phân hoạch $C(n) = n + (n-1) + (n-2) + \dots + 1 = n(n+1)/2$

31

Quick Sort – Độ phức tạp

Trường hợp	Độ phức tạp
Tốt nhất	$n * \log(n)$
Trung bình	$n * \log(n)$
Xấu nhất	n^2

32

Quick Sort – Nhận xét

- Phức tạp hơn Bubble Sort nhưng hiệu quả hơn, trong trường hợp tốt nhất độ phức tạp $O(n\log n)$.
- Quick Sort thích hợp cho dãy ban đầu chưa có thứ tự.
- Quick Sort **kém hiệu quả** khi dãy ban đầu gần **có thứ tự**, đặc biệt trường hợp xấu nhất là dãy đã có thứ tự (tăng hoặc giảm)

33

Quick Sort – Bài Tập

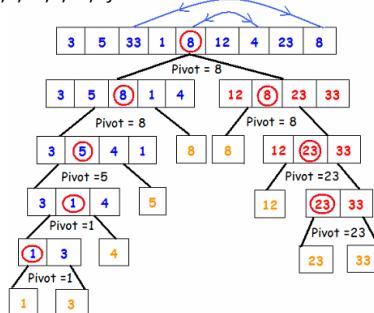
Cho biết kết quả từng bước khi sắp xếp dãy sau tăng dần bằng thuật toán quick Sort

$$M=\{3,5,33,1,8,12,4,23,8\}$$

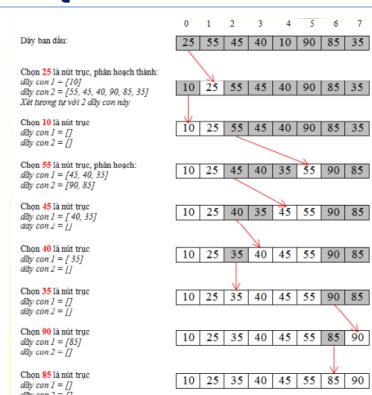
34

Quick Sort – Bài Tập

$$M=\{3,5,33,1,8,12,4,23,8\}$$



Quick Sort – Cách khác



35

Quick Sort – Cách khác

```

10. void QuickSort(int a[],int n
11. {
12.     if (n<=1)
13.         return;
14.     int b[100];int k;
15.     int c[100];int ℓ;
16.     int TrongTai = a[0];
17.     Split(a,n,TrongTai,b,k,c,ℓ);
18.     QuickSort(b,k);
19.     QuickSort(c,ℓ);
20.     for(int i=0;i<k;i++)
21.         a[i] = b[i];
22.     for(int i=0;i<n-k-ℓ;i++)
23.         a[k+i] = TrongTai;
24.     for(int i=0;i<ℓ;i++)
25.         a[k+(n-k-ℓ)+i] = c[i]
26. }

```

37

Quick Sort – Cách khác

```

11. void Split(int a[],int n,
              int x,
              int b[],int &k,
              int c[],int &ℓ)
12. {
13.     k = ℓ = 0;
14.     for(int i=0;i<n;i++)
15.         if(a[i]<x)
16.             b[k++] = a[i];
17.         else
18.             if(a[i]>x)
19.                 c[ℓ++]=a[i];
20. }

```

38

Merge Sort – Trộn

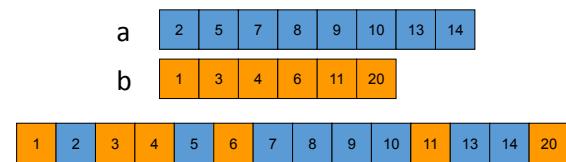
Bài toán: Cho 2 mảng một chiều các số nguyên a và b được sắp thứ tự tăng dần. Hãy trộn mảng a và mảng b lại với nhau để được mảng c sắp thứ tự tăng dần.

39

Merge Sort – Trộn

Tư tưởng trộn:

- Có 2 mảng đã được sắp xếp chiều dài a, b
- Tạo ra 1 mảng chung được sắp xếp



40

Merge Sort – Trộn Ý Tưởng

Bước 1: chọn min của 2 phần tử đầu dãy chép qua mảng kết quả

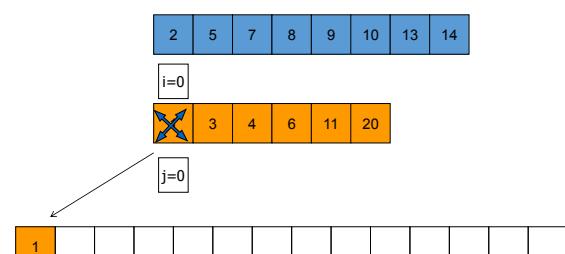
Bước 2: hủy phần tử min

Bước 3: nếu chưa đến cuối mảng trở về bước 1

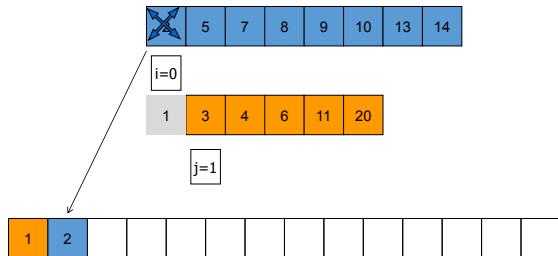
Nếu đến cuối mảng: chép phần còn lại của mảng kia vào mảng kết quả

41

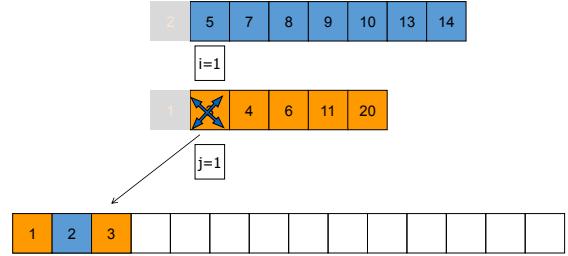
Merge Sort – Trộn



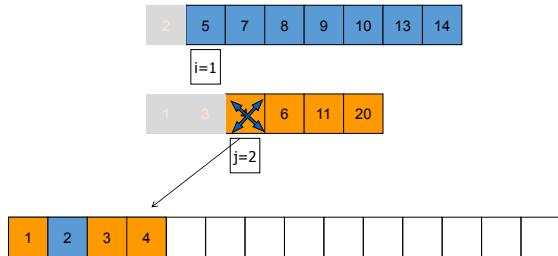
42

Merge Sort – Trộn

43

Merge Sort – Trộn

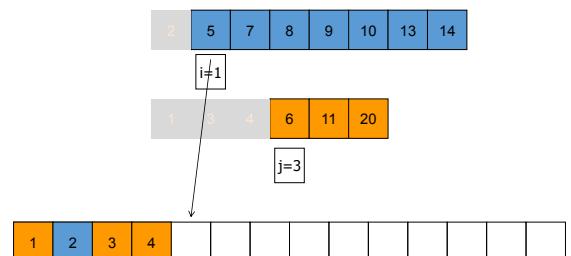
44

Merge Sort – Trộn

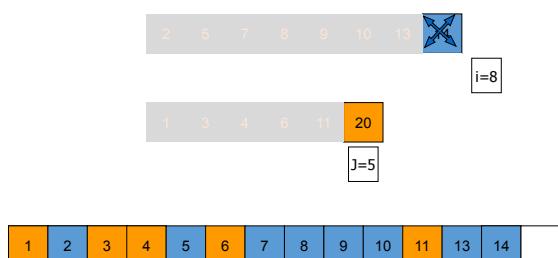
45

Merge Sort – Trộn

Thực hiện lần lượt



46

Merge Sort – Trộn

47

Merge Sort – Trộn

Chép phần còn lại của mảng 2



48

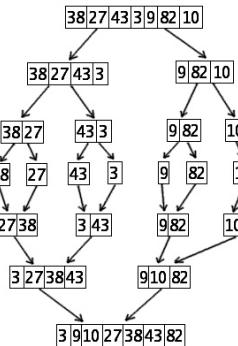
Merge Sort – Trộn

Ví dụ khác ta có a=(1,3,7,9), b=(2,6), quá trình hòa nhập diễn ra như sau:

Danh sách a	Danh sách b	So sánh	Danh sách c
1,3,7,9	2,6	1<2	1
3,7,9	2,6	2<3	1,2
3,7,9	6	3<6	1,2,3
7,9	6	6<7	1,2,3,6
7,9			1,2,3,6,7,9

49

Merge Sort – Trộn



50

Merge Sort – Viết hàm

```

11. void Tron(int a[],int n,
             int b[],int m,
             int c[],int &p)
12. {
13.   ...
14. }
15. ...
16. ...
17. ...
18. ...
19. ...
20. ...
21. ...
22. ...
23. }
```

51

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
             int b[],int m,
             int c[],int &p)
12. {
13.   int i = 0;
14.   int j = 0;
15.   p = 0;
16.   ...
17.   ...
18.   ...
19.   ...
20.   ...
21.   ...
22.   ...
23. }
```

52

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
             int b[],int m,
             int c[],int &p)
12. {
13.   int i = 0;
14.   int j = 0;
15.   p = 0;
16.   while(      )
17.   {
18.     if(          a[i]<b[j])
19.       c[p] = a[i];
20.     ...
21.   }
22. }
```

53

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
             int b[],int m,
             int c[],int &p)
12. {
13.   int i = 0;
14.   int j = 0;
15.   p = 0;
16.   while(      )
17.   {
18.     if(          a[i]<b[j])
19.       c[p++] = a[i];
20.     ...
21.   }
22. }
```

54

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
           int b[],int m,
           int c[],int &p)
12. {
13.     int i = 0;
14.     int j = 0;
15.     p = 0;
16.     while( )
17.     {
18.         if(          a[i]<b[j])
19.             c[p++] = a[i++];
20.         else
21.             c[p++] = b[j++];
22.     }
23. }

```

55

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
           int b[],int m,
           int c[],int &p)
12. {
13.     int i = 0;
14.     int j = 0;
15.     p = 0;
16.     while( )
17.     {
18.         if(          a[i]<b[j])
19.             c[p++] = a[i++];
20.         else
21.             c[p++] = b[j++];
22.     }
23. }

```

56

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
           int b[],int m,
           int c[],int &p)
12. {
13.     int i = 0;
14.     int j = 0;
15.     p = 0;
16.     while(! (i>=n && j>=m))
17.     {
18.         if(          a[i]<b[j])
19.             c[p++] = a[i++];
20.         else
21.             c[p++] = b[j++];
22.     }
23. }

```

57

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
           int b[],int m,
           int c[],int &p)
12. {
13.     int i = 0;
14.     int j = 0;
15.     p = 0;
16.     while(! (i>=n && j>=m))
17.     {
18.         if((i<n&&j<m&&a[i]<b[j])
19.            ||
20.            )
21.             c[p++] = a[i++];
22.         else
23.             c[p++] = b[j++];
24.     }
25. }

```

58

Merge Sort – Trộn

```

11. void Tron(int a[],int n,
           int b[],int m,
           int c[],int &p)
12. {
13.     int i = 0;
14.     int j = 0;
15.     p = 0;
16.     while(! (i>=n && j>=m))
17.     {
18.         if((i<n&&j<m&&a[i]<b[j])
19.            ||
20.            )
21.             c[p++] = a[i++];
22.         else
23.             c[p++] = b[j++];
24.     }
25. }

```

59

Trộn 2 dãy đã có thứ tự thành 1 dãy có thứ tự. Tiến hành qua nhiều bước trộn như sau:

▪ Bước 1:

- Xem dãy a_0, a_1, \dots, a_{n-1} cần sắp xếp như n dãy con đã có thứ tự, mỗi dãy con có 1 phần tử.

- Trộn từng cặp 2 dãy con kế cận, ta được $n/2$ dãy con đã có thứ tự, mỗi dãy con có 2 phần tử

▪ Bước 2:

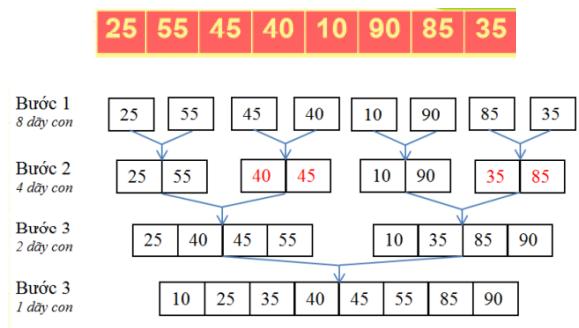
- Xem dãy a_0, a_1, \dots, a_{n-1} cần sắp xếp như $n/2$ dãy con đã có thứ tự, mỗi dãy con có 2 phần tử.

- Trộn từng cặp 2 dãy con kế cận, ta được $n/4$ dãy con đã có thứ tự, mỗi dãy con có 4 phần tử

▪ Quá trình tiếp diễn cho đến khi được 1 dãy con có n phần tử

60

Merge Sort – Ý tưởng



Merge Sort – Ý tưởng

- Thuật toán Merge Sort chia không gian cần sắp xếp thành 2 không gian con.
 - + Nếu không gian con thứ nhất có nhiều hơn một phần tử thì sắp xếp không gian con này bằng thuật toán Merge Sort.
 - + Nếu không gian con thứ hai có nhiều hơn một phần tử thì sắp xếp không gian con này bằng thuật toán Merge Sort.
- Trộn 2 không gian con đã được sắp xếp lại với nhau.

Merge Sort – Trộn

```
void MergeSort(int []M,int left,int right)
{
    if (left >= right) return;
    int mid = (left + right) / 2;
    MergeSort(M, left, mid);
    MergeSort(M, mid + 1, right);
    Merge(M, left, mid, right);
}
```

Merge Sort – Trộn

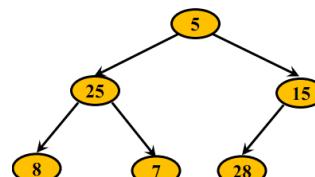
```
void Merge(int []M,int left,int mid,int right)
{
    int[] Temp = new int[right - left + 1];
    int pos = 0;
    int i = left;
    int j = mid + 1;
    while(!(i==mid && j>right))
    {
        if((i<=mid && j <=right && M[i]<M[j]) || j>right)
            Temp[pos++] = M[i++];
        else
            Temp[pos++] = M[j++];
    }
    for(i=0;i<Temp.Length;i++)
        M[left + i] = Temp[i];
}
```

Merge Sort – Nhận xét

- Không tối ưu bộ nhớ vì có dùng mảng tạm trong quá trình trộn.
- Nhanh hơn Quick Sort vì thời gian thực hiện Merge Sort có bậc nhỏ hơn O(nlogn), còn trong trường hợp tốt nhất Quick Sort mới có bậc O(nlogn)
- Thường dùng Merge Sort để sắp xếp khối dữ liệu lớn ở bộ nhớ ngoài.

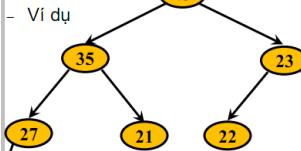
Heap Sort

- Định nghĩa Heap sơ khởi: Heap là một cây nhị phân đầy đủ
- Ví dụ 1:



Heap Sort

- Định nghĩa Heap sơ khởi: Heap là một cây nhị phân đầy đủ
- Mỗi nút trong Heap chứa một giá trị có thể so sánh với giá trị của nút khác.
- Đặc điểm của Heap là giá trị của mỗi nút \geq giá trị của các nút con của nó



67

Định nghĩa Heap

Heap là một cây nhị phân thỏa các tính chất sau:

- + Heap là một cây đầy đủ;
- + Giá trị của mỗi nút không bao giờ bé hơn giá trị của các nút con

Hệ quả:

- + Nút lớn nhất là ... ?

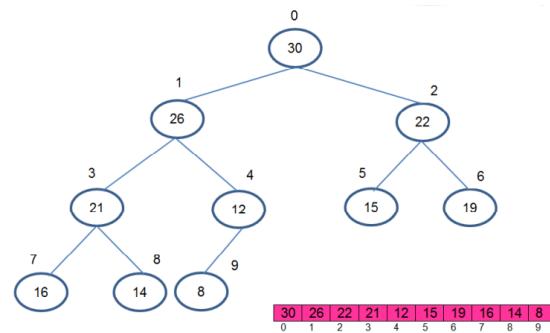
68

Heap Sort

- **Nút gốc (root):** nút có nội dung lớn nhất, đánh số 0. Tiếp theo là 1, 2, 3, ...
- **Đường đi (path):** đường đi từ nút gốc đến nút lá, đi qua các nút có nội dung giảm dần.

69

Heap Sort



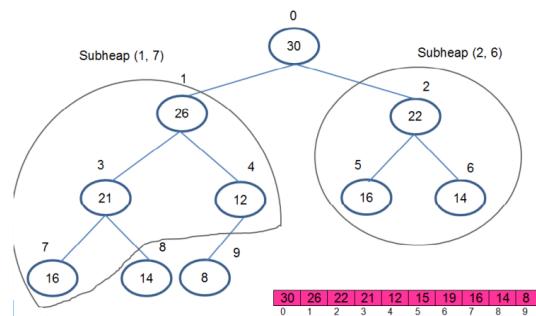
70

Heap Con (subheap)

- Trên Heap có rất nhiều **heap con**, mỗi heap con có nút gốc và các nút con của nó.
- Gọi **subheap(p, m)** là heap con có nút gốc ở vị trí p và các nút con có vị trí $\leq m$.
- Nếu $p > m$ thì subheap là rỗng.
- subheap($2p+1$) là heap con bên trái, subheap($2p+2$) là heap con bên phải của $subheap(p, m)$

71

Sub Heap



72

Heap Sort – Ý tưởng

- Chuyển dãy cần sắp xếp thành cấu trúc Heap
 - Đầu tiên xem heap chỉ có 1 nút ($a[0]$).
 - Gọi tác vụ Insert để thêm nút $a[1]$ vào heap, ta được heap có 2 nút ($a[0], a[1]$)
 - Tiếp tục gọi tác vụ Insert để lần lượt thêm các nút $a[i]$ vào heap, đến khi chuyển dãy ban đầu thành heap n nút.
- Lần lượt chuyển nút gốc của heap đưa về vị trí thích hợp (về phía cuối)

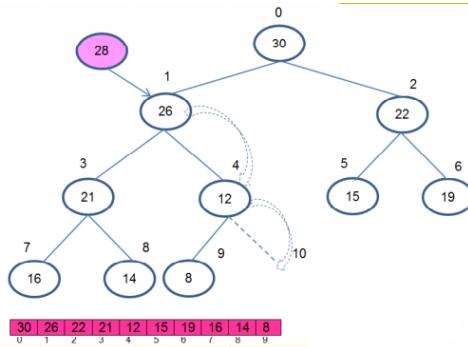
73

Heap Sort – Ý tưởng

- Lần lượt chuyển nút gốc của heap đưa về vị trí thích hợp (về phía cuối)
 - Đầu tiên heap có n nút ($a[0], a[1], \dots, a[n-1]$).
 - Gọi tác vụ removeRoot để xóa nút gốc (nút lớn nhất), được heap $n-1$ nút, chuyển nút gốc bị xóa thành $a[n-1]$.
 - Với heap $n-1$ nút ($a[0], a[1], \dots, a[n-2]$), gọi tác vụ removeRoot để xóa nút gốc, được heap $n-2$ nút, chuyển nút gốc bị xóa thành $a[n-2]$.
 - Tiếp tục chuyển nút gốc về vị trí thích hợp, cuối cùng ta được dãy đã sắp thứ tự.

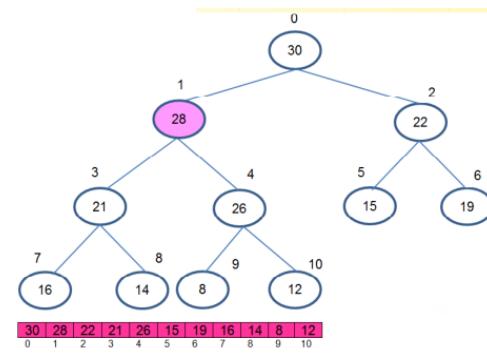
74

Thêm một nút vào Heap



75

Thêm một nút vào Heap



76

Thêm một nút vào Heap

Mô tả: thêm nút mới x vào heap

Input: heap có k nút (từ $a[0], \dots, a[k-1]$)
Output: heap có $k+1$ nút (từ $a[0], \dots, a[k]$)

Các bước thực hiện:

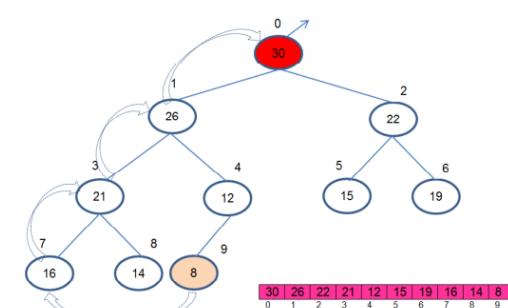
- Gán $s = k$; // s xuất phát từ vị trí k
 $f = (s-1)/2$; // f là nút cha của s
- Lần theo đường đi từ vị trí k về vị trí 0 để lần lượt kéo các nút có giá trị nhỏ hơn x "lên" 1 mức.


```
while (s > 0 && a[f] < x)
    {
        a[s] = a[f]; // di chuyển nút đang xét a[f] từ vị trí f xuống vị trí con là s
        s = f; // tiếp tục lần lượt nút cha phía trên
        f = (s-1)/2;
    }
```
- Khi gặp nút đầu tiên $a[f] \geq x$ thì đưa nút x vào vị trí nút con của nút này.


```
a[s] = x;
```

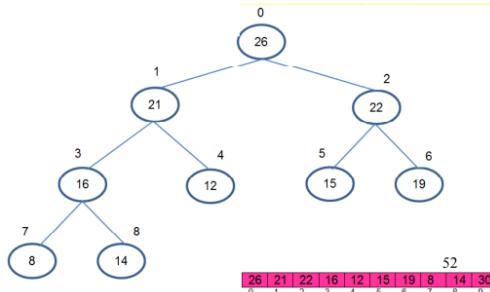
77

Xóa nút gốc từ Heap



78

Xóa nút gốc từ Heap



79

Xóa nút gốc từ Heap

Input: heap có k nút (từ $a[0]$, ..., $a[k-1]$)

Mô tả: xóa nút gốc $a[0]$ và điều chỉnh lại các nút từ $a[1]$ đến $a[k-1]$ thành các nút từ $a[0]$ đến $a[k-2]$

Output: heap có k-1 nút (từ $a[0]$, ..., $a[k-2]$)

80

Xóa nút gốc từ Heap

```

1. Gán p = a[0];
2. Điều chỉnh lại heap sau khi xóa nút gốc.
   2.1. f = 0;           // root
      s = largeson (f, k-2); // Khi xóa nút gốc thi phải chọn nút khác thay thế
      cho nút gốc. Nút được chọn là nút lớn nhất trong 3 nút: nút con bên trái nút gốc, nút con bên phải, nút cuối a[f-1]. s trả về vị trí nút con được chọn, nếu không có nút con thì s = -1
   2.2. while (s >=0 && a[k-1] < a[s])           // có nút con lên thay thế
      {
         // chọn nút thay thế kế tiếp là max (2 nút con của nó và nút cuối a[f-1]).
         a[f] = a[s];
         f = s;
         s = largeson (f, k-2);
      }
   2.3. a[f] = a[k-1]; // nút cuối a[k-1] được chọn thay thế
3. Trả về p là nội dung của nút bị xóa

```

81

Heap Sort

Input: mảng a: $a[0], a[1], \dots, a[n-1]$ chưa sắp xếp

Output: mảng a đã sắp xếp

Các bước thực hiện:

- Chuyển dãy cần sắp xếp thành cấu trúc heap bằng cách tuần tự thêm nút $x = a[i]$ vào heap (vẫn bảo toàn cấu trúc heap)

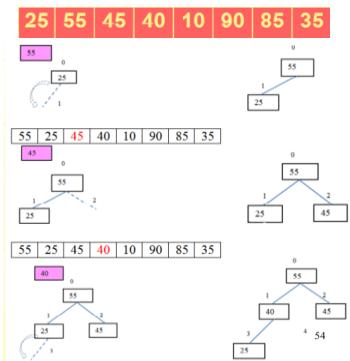

```
for (i = 1; i < n; i++)
        insert (a[], i, a[i])
```
- Sắp xếp dãy số dựa trên heap bằng cách lần lượt chuyển nút gốc của heap đưa về vị trí thích hợp (vẫn bảo toàn cấu trúc heap)
 - Đưa phần tử lớn nhất về vị trí đúng ở cuối dãy:


```
cuoiheap = a[i]; // lưu lại nút cuối heap
a[i] = a[0];
```
 - Xóa nút lớn nhất ra khỏi heap và điều chỉnh lại heap


```
a[i] = removeroot(a[], i+1)
```

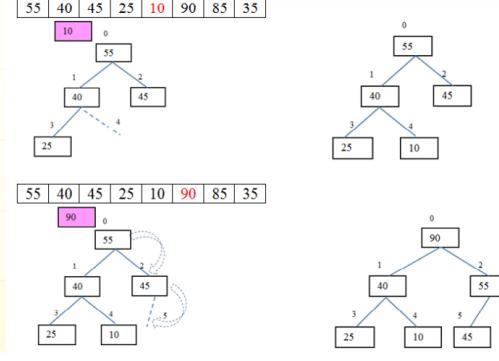
82

Heap Sort -Ví dụ minh họa – Tạo Heap

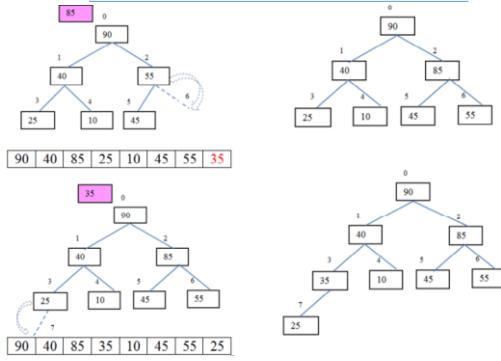


83

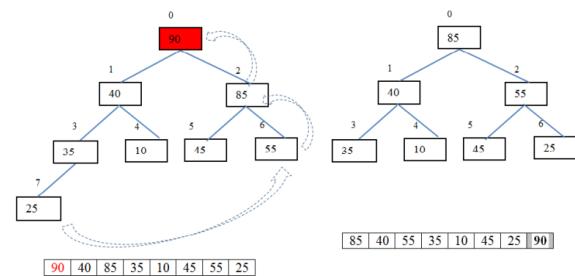
Heap Sort -Ví dụ minh họa – Tạo Heap



Heap Sort - Ví dụ minh họa – Tạo Heap



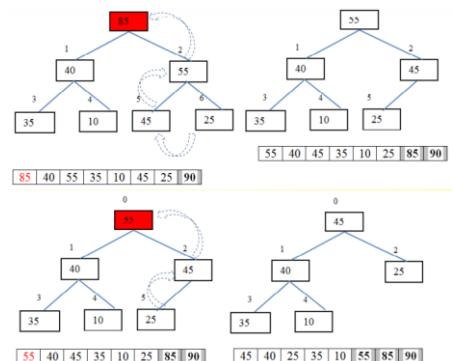
Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



85

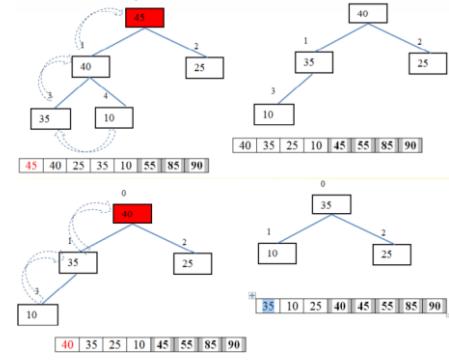
86

Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



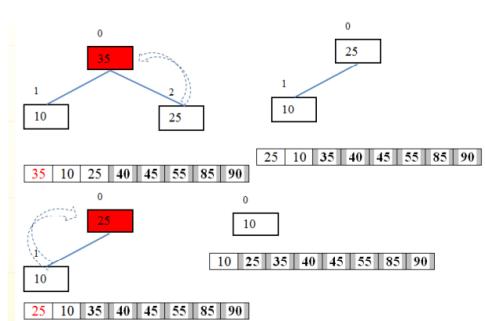
87

Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



88

Heap Sort - Ví dụ minh họa – Sắp xếp từ Heap



89

Heap Sort – Cài đặt

```
void HeapSort (int a[], int n)
{
    int i, x, s, f, cuoiheap;
    /* I. chuyển dãy thành cây tríc heap bằng cách tuân tự thêm nút a[i] vào heap*/
    for (i = 1; i < n; i++)
    {
        x = a[i];
        /* insert (a[], i, x)*/
        s = i;
        f = (s-1)/2;
        while (s > 0 && a[f] < x) // lần theo đường đi từ vị trí i về vị trí 0 để lần lượt
            // kéo các nút có giá trị nhỏ hơn x lên 1 mức
        {
            a[s] = a[f];
            s = f;
            f = (s-1)/2;
        }
        a[s] = x;
        // khi gặp nút đầu tiên có giá trị >= x thì đưa nút x
        // vào vị trí nút con của nút này
    }
}
```

90

Heap Sort – Cài đặt

```

/*2. Lần lượt xóa nút gốc (a[0]), đưa về vị trí thích hợp*/
for (i = n-1; i > 0; i--)
{
    cuoheap = a[i];           // lưu lại giá trị nút cuối của heap a[n-1]
    a[i] = a[0];              // chuyển nút gốc a[0] bị xóa về phía cuối
    /* removeroot(a[], i+1)*/
    f = 0;
    s = largeson(f, i-1);
    while (s >= 0)
    {
        a[f] = a[s];
        f = s;
        s = largeson (f, i-1);
    }
    a[f] = cuoheap;
} //end for
} //kết thúc

```

Heap Sort – Nhận xét

- Chỉ thao tác trên 1 mảng a[] : đầu tiên chuyển a[] thành cấu trúc heap, sau đó lần lượt chuyển các nút gốc của heap về vị trí phù hợp trên a[].
- Không phụ thuộc vào tính thứ tự của dãy cần sắp.
- Một trong những giải thuật sắp xếp nội hiệu quả nhất.

Độ phức tạp tính toán : O(nlgn)

Slide được tham khảo từ

- **Slide được tham khảo từ:**

- Slide CTDL GT, Khoa Khoa Học Máy Tính, ĐHCNTT
- Slide CTDL GT, Thầy Nguyễn Tân Trần Minh Khang, ĐH CNTT
- Slid CTDL GT, Cô Trần Thị Thưương, ĐH CNTT
- Congdongcviet.com
- Cplusplus.com



91

92

93

94