Phân tích và Thiết kế THUẬT TOÁN

Hà Đại Dương
duonghd@mta.edu.vn
Web: fit.mta.edu.vn/~duonghd

Bài 4 - Thiết kế thuật toán Chia để trị - Divide&Conquer

PHÂN TÍCH VÀ THIẾT KẾ THUẬ TOÁN

NỘI DUNG

I. Giới thiệu
II. Lược đồ chung
III. Bài toán áp dụng
IV. Bài tập

I. Giới thiệu

- Là một phương pháp được áp dụng rộng rãi
- Ý tưởng chung là phân rã bài toán thành bài toán nhỏ hơn “độc lập” với nhau.
- Giải các bài toán con theo cùng 1 cách thức
- “Tổng hợp” lời các bài toán con để có được kết quả bài toán ban đầu.

⇨ Tư tưởng chung của cách tiếp cận **Chia** để **trị**

II. Lược đồ chung

Chia:
- Bằng cách nào đó chia tập hợp các đối tượng của bài toán thành bài toán con “độc lập”
- Tiếp tục chia các bài toán con cho đến khi có thể giải trực tiếp (không cần, hoặc không thể chia nhỏ nữa)

Trị:
- Trên các bài toán con thực hiện cùng một cách thức: Chia nhỏ nếu cần hoặc giải trực tiếp

Tổng hợp:
- Khi mỗi bài toán con được giải, tổng hợp để có kết quả bài toán ban đầu.

II. Lược đồ chung

Nếu gọi D&C($\Re$) - Với $\Re$ là miền dữ liệu là hàm thể hiện cách giải bài toán theo phương pháp chia để trị thì ta có thể viết :

```
void D&C(ℜ)
{
        If (ℜ đủ nhỏ)
                giải bài toán;
        Else
        {
                Chia ℜ thành ℜ1 , …,ℜm ;
                for (i = 1; i <=m; i++)
                        D&C(ℜi);
                Tổng hợp kết quả;
        }
}
```

III. Bài toán áp dụng

1. Tìm kiếm nhị phân

The Manhattan phone book has 1,000,000+ entries.

How is it possible to locate a name by examining just a tiny, tiny fraction of those entries?

III. Bài toán áp dụng

1. Tìm kiếm nhị phân

Key idea of "phone book search": repeated halving

To find the page containing Pat Reed's number…

```
while (Phone book is longer than 1 page)
    Open to the middle page.
    if "Reed" comes before the first entry,
        Rip and throw away the 2nd half.
    else
        Rip and throw away the 1st half.
    end
end
```

III. Bài toán áp dụng

1. Tìm kiếm nhị phân

What happens to the phone book length?

```
Original:      3000 pages
After 1 rip:   1500 pages
After 2 rips:   750 pages
After 3 rips:   375 pages
After 4 rips:   188 pages
After 5 rips:    94 pages
      :
After 12 rips:    1 page
```

III. Bài toán áp dụng

1. Tìm kiếm nhị phân

- Repeatedly halving the size of the "search space" is the main idea behind the method of binary search.
- An item in a sorted array of length n can be located with just $\log_2 n$ comparisons.
- "Savings" is significant!

n	log2(n)
100	7
1000	10
10000	13

III. Bài toán áp dụng

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 1

Mid: 6

R: 12

v(Mid) <= x

So throw away the left half…

III. Bài toán áp dụng

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 6

Mid: 9

R: 12

x < v(Mid)

So throw away the right half…

III. Bài toán áp dụng

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 6

Mid: 7

R: 9

v(Mid) <= x

So throw away the left half…

III. Bài toán áp dụng

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 7

Mid: 8

R: 9

v(Mid) <= x

So throw away the left half…

III. Bài toán áp dụng

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 8

Mid: 8

R: 9

Done because
R-L = 1

III. Bài toán áp dụng

1. Tìm kiếm nhị phân

- Mô tả thuật toán:
  - Vào A[1..n]
  - Ra: Chỉ số k = -1 nếu không tìm thấy
    1<=k<=n nếu tìm thấy
- Độ phức tạp thuật toán: $O(\log_2 n)$

```
Tknp(a, x, Đầu, Cuối) ≡
    If (Đầu > Cuối)
        return 0 ; {dãy trống}
    Else
    {
            Giữa = (Đầu + cuối) / 2;
        If (x == a[Giữa])
            return 1;
        else
                if (x > a[Giữa])
                Tknp(a, x, Giữa + 1, Cuối) ;
                else
                Tknp(a, x, Đầu, Giữa - 1) ;
    }
```

III. Bài toán áp dụng

1. Tìm kiếm nhị phân

- Cài đặt:

```
int tknp(int a[max],int x,int l, int r)
{
    int mid;
    if ( l > r )
        return 0;
    mid = (l+r)/2;
    if ( x == a[mid] )
        return 1;
    else
        if ( x > a[mid] )
            return tknp(a,x,mid+1,r);
        else
            return tknp(a,x,l,mid-1);
}
```

III. Bài toán áp dụng

2. Tìm giá trị MIN, MAX

- Phát biểu bài toán: Cho mảng A có n phần tử. Tìm giá trị lớn nhất (MAX) và giá trị nhỏ nhất (MIN) trên mảng A.
- Tìm kiếm "nhị phân":
  - Chia đôi mảng A, tìm kiếm MIN, MAX trên mỗi nữa sau đó tổng hợp kết quả trên hai nửa đó để tìm MIN, MAX của cả mảng A.
  - Nếu đoạn chia chỉ có một phần tử thì MIN=MAX=phần tử đó.

III. Bài toán áp dụng

2. Tìm giá trị MIN, MAX

- Mô tả thuật toán:
  - Vào: A[l..r]
  - Ra: MIN=Min(A[1],…,A[r])
    MAX=Max(A[1],…,A[r])

```
MinMax(a,l, r, Min, Max)
  {
   if (l == r)
   {
      Min  = a[l];
      Max = a[l];
   }
   Else
   {    MinMax(a,l, (l+r) / 2, Min1, Max1);
        MinMax(a,(l+r) /2 + 1, r , Min2, Max2);
        If (Min1 < Min2)
               Min = Min1;
        Else
               Min = Min2;
        If (Max1 > Max2)
               Max = Max1
        Else
               Max = Max2;
     }
  }
```

III. Bài toán áp dụng

2. Tìm giá trị MIN, MAX

- Độ phức tạp thuật toán:

Gọi T(n) là số phép toán so sánh

$$T(n) = \begin{cases} T(n/2) + T(n/2) + 2 \ ; \ n > 2 \\ 1 \ ; \ n = 2 \\ 0 \ ; \ n = 1 \end{cases}$$

Với $n = 2^k$, thì :

$$T(n) = 2 + 2T(n/2) = 2 + 2^2 + 2^2 T(n/2^2) = \cdots = 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i$$

$$= \sum_{i=1}^{k} 2^i - 2^{k-1} = 2^{k+1} - 2^{k-1} - 2 = \frac{3n}{2} - 2 .$$

Vậy $T(n) \in O(n)$.

III. Bài toán áp dụng

2. Tìm giá trị MIN, MAX

- Cài đặt:

```
void MinMax(int a[.], int  l, int  r, int &Min, int &Max )
{   int Min1,Min2,Max1,Max2;
    if (l == r )
    {  Min = a[l];
       Max= a[l];  }
    else
    { MinMax(a,l,(l+r)/2 , Min1, Max1);
      MinMax(a,(l+r) /2 + 1,r, Min2, Max2);
      if (Min1 < Min2)
             Min = Min1;
      else
             Min = Min2;
      if (Max1 > Max2)
             Max = Max1;
      else
             Max = Max2;
    }
}
```

III. Bài toán áp dụng

3. Thuật toán MergeSort

- Phát biểu bài toán: Cho mảng gồm n phần tử A[1..n], sắp xếp mảng A theo thứ tự tăng dần
- **Ý tưởng:**
  - 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

III. Bài toán áp dụng

3. Thuật toán MergeSort

If I have two helpers, I'd…

- Give each helper half the array to sort
- Then I get back the sorted subarrays and merge them.

What if those two helpers each had two sub-helpers?

And the sub-helpers each had two sub-sub-helpers? And…

III. Bài toán áp dụng

3. Thuật toán MergeSort

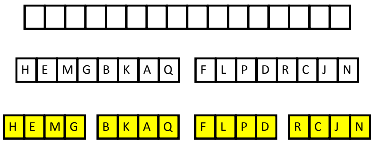
H	E	M	G	B	K	A	Q	F	L	P	D	R	C	J	N

H	E	M	G	B	K	A	Q

F	L	P	D	R	C	J	N

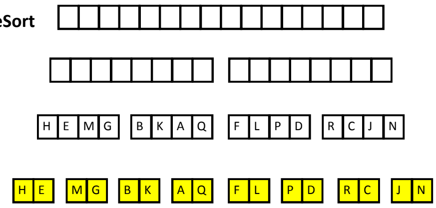
III. Bài toán áp dụng

3. Thuật toán MergeSort



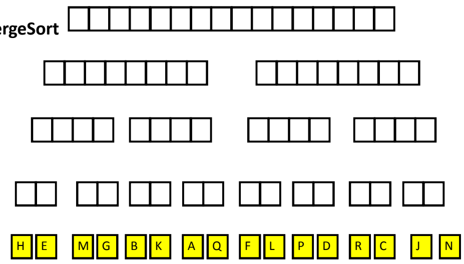
III. Bài toán áp dụng

3. Thuật toán MergeSort



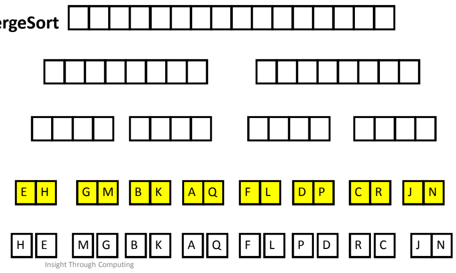
III. Bài toán áp dụng

3. Thuật toán MergeSort



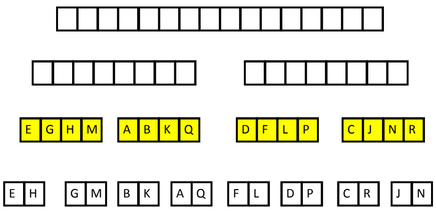
III. Bài toán áp dụng

3. Thuật toán MergeSort



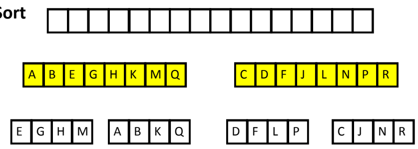
III. Bài toán áp dụng

3. Thuật toán MergeSort



III. Bài toán áp dụng

3. Thuật toán MergeSort



III. Bài toán áp dụng

3. Thuật toán MergeSort

A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q	R

A	B	E	G	H	K	M	Q

C	D	F	J	L	N	P	R

III. Bài toán áp dụng

3. Thuật toán MergeSort

H	E	M	G	B	K	A	Q	F	L	P	D	R	C	J	N
A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q	R

III. Bài toán áp dụng

3. Thuật toán MergeSort

- **Ý tưởng 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

III. Bài toán áp dụng

3. Thuật toán MergeSort

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

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

III. Bài toán áp dụng

3. Thuật toán MergeSort

12	33	35	45

15	42	55	65	75

12	15	33	35	42	45	55	65	75

Merge

x:

12	33	35	45

ix: 1

y:

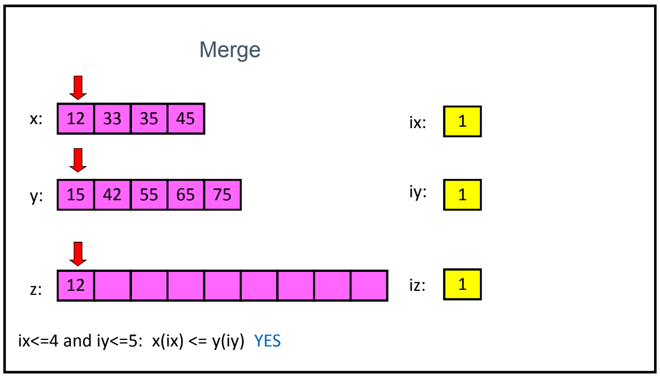
15	42	55	65	75

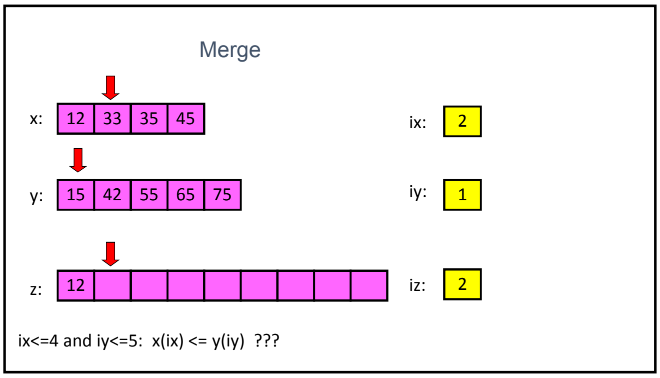
iy: 1

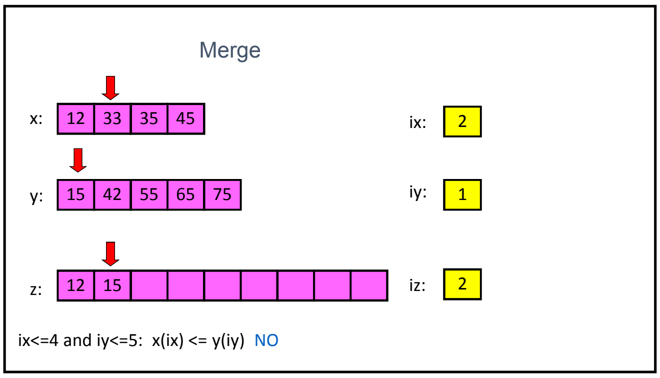
z:

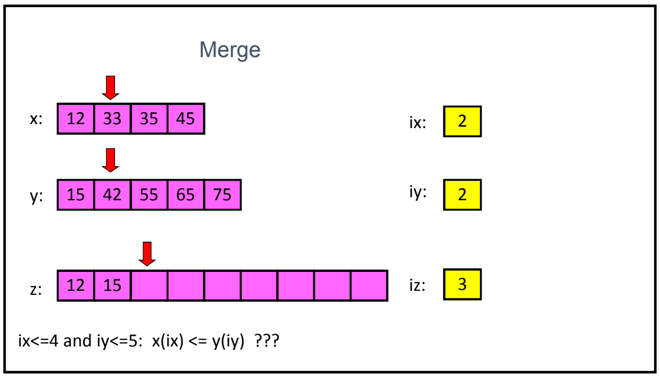
iz: 1

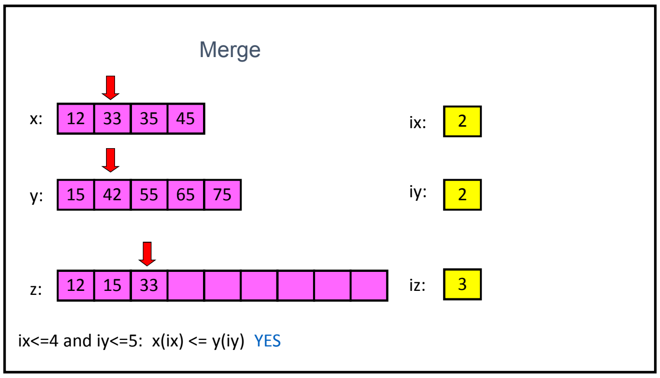
ix<=4 and iy<=5: x(ix) <= y(iy) ???

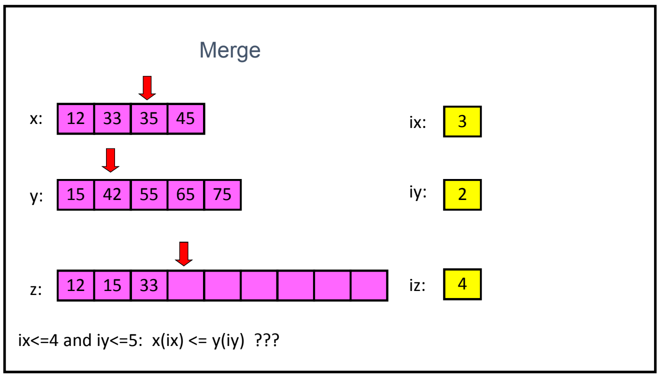

Merge
x: 12 33 35 45
ix: 1
y: 15 42 55 65 75
iy: 1
z: 12
iz: 1
ix<=4 and iy<=5: x(ix) <= y(iy) YES

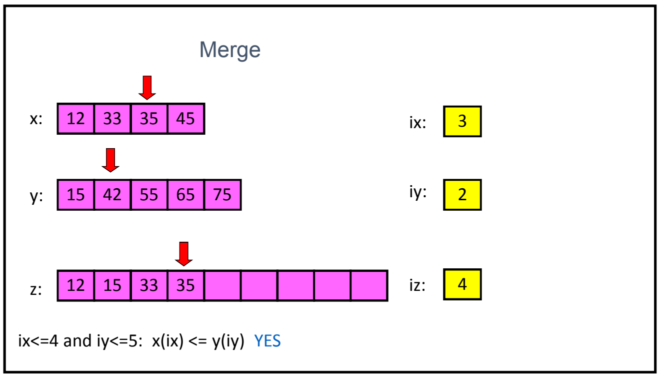

Merge
x: 12 33 35 45
ix: 2
y: 15 42 55 65 75
iy: 1
z: 12
iz: 2
ix<=4 and iy<=5: x(ix) <= y(iy) ???

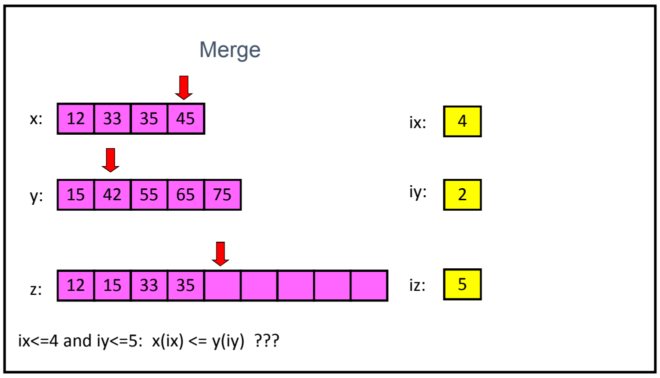

Merge
x: 12 33 35 45
ix: 2
y: 15 42 55 65 75
iy: 1
z: 12 15
iz: 2
ix<=4 and iy<=5: x(ix) <= y(iy) NO

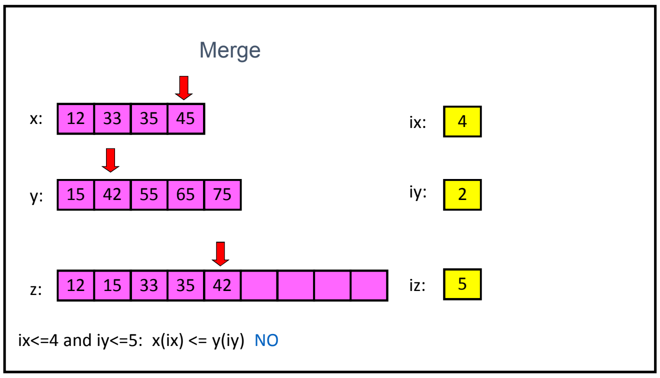

Merge
x: 12 33 35 45
ix: 2
y: 15 42 55 65 75
iy: 2
z: 12 15
iz: 3
ix<=4 and iy<=5: x(ix) <= y(iy) ???


Merge
x: 12 33 35 45
ix: 2
y: 15 42 55 65 75
iy: 2
z: 12 15 33
iz: 3
ix<=4 and iy<=5: x(ix) <= y(iy) YES


Merge
x: 12 33 35 45
ix: 3
y: 15 42 55 65 75
iy: 2
z: 12 15 33
iz: 4
ix<=4 and iy<=5: x(ix) <= y(iy) ???


Merge
x: 12 33 35 45
ix: 3
y: 15 42 55 65 75
iy: 2
z: 12 15 33 35
iz: 4
ix<=4 and iy<=5: x(ix) <= y(iy) YES


Merge
x: 12 33 35 45
ix: 4
y: 15 42 55 65 75
iy: 2
z: 12 15 33 35
iz: 5
ix<=4 and iy<=5: x(ix) <= y(iy) ???


Merge
x: 12 33 35 45
ix: 4
y: 15 42 55 65 75
iy: 2
z: 12 15 33 35 42
iz: 5
ix<=4 and iy<=5: x(ix) <= y(iy) NO

Merge

x:

12	33	35	45

ix: 4

y:

15	42	55	65	75

iy: 3

z:

12	15	33	35	42				

iz: 5

ix<=4 and iy<=5: x(ix) <= y(iy) ???

Merge

x:

12	33	35	45

ix: 4

y:

15	42	55	65	75

iy: 3

z:

12	15	33	35	42	45			

iz: 5

ix<=4 and iy<=5: x(ix) <= y(iy) YES

Merge

x:

12	33	35	45

ix: 5

y:

15	42	55	65	75

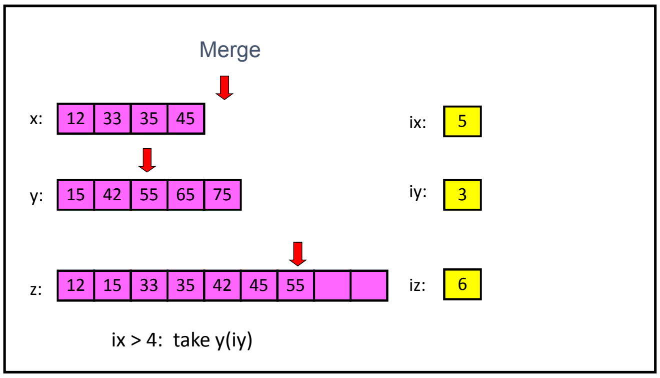
iy: 3

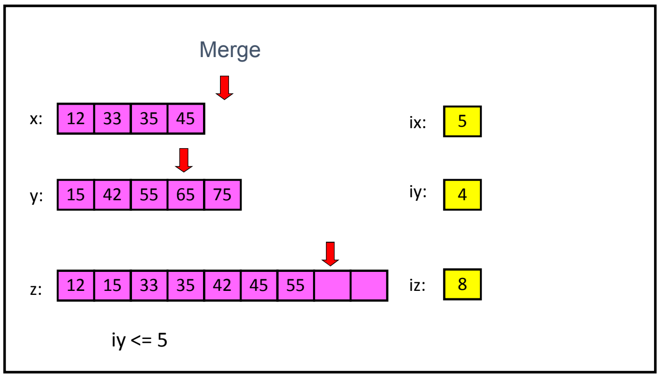
z:

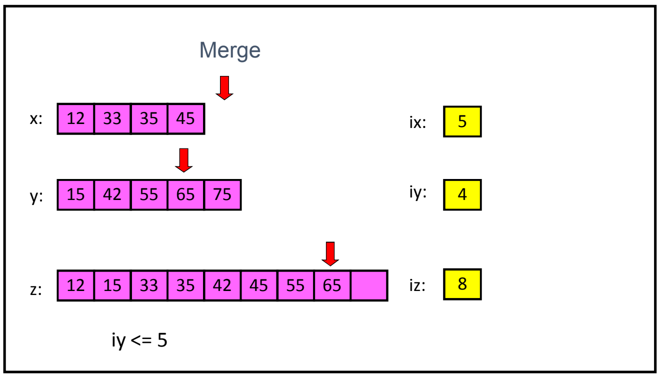
12	15	33	35	42	45			

iz: 6

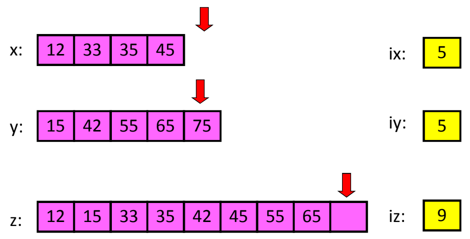
ix > 4


Merge
x: 12 33 35 45
ix: 5
y: 15 42 55 65 75
iy: 3
z: 12 15 33 35 42 45 55
iz: 6
ix > 4: take y(iy)


Merge
x: 12 33 35 45
ix: 5
y: 15 42 55 65 75
iy: 4
z: 12 15 33 35 42 45 55
iz: 8
iy <= 5

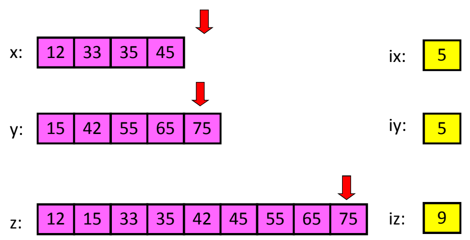

Merge
x: 12 33 35 45
ix: 5
y: 15 42 55 65 75
iy: 4
z: 12 15 33 35 42 45 55 65
iz: 8
iy <= 5

Merge



iy <= 5

Merge



iy <= 5

III. Bài toán áp dụng

3. Thuật toán 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]);

III. Bài toán áp dụng

3. Thuật toán 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]);

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

III. Bài toán áp dụng

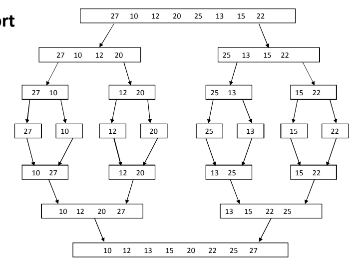
3. Thuật toán MergeSort

- 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))

III. Bài toán áp dụng

3. Thuật toán MergeSort

- **Ví dụ**



III. Bài toán áp dụng

4. Thuật toán QuickSort

- Phát biểu bài toán: Cho mảng gồm n phần tử A[1..n], sắp xếp mảng A theo thứ tự tăng dần.
- 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

III. Bài toán áp dụng

4. Thuật toán QuickSort

- Phát biểu bài toán: Cho mảng gồm n phần tử A[1..n], sắp xếp mảng A theo thứ tự tăng dần.
- Ý tưởng:
  - 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

III. Bài toán áp dụng

4. Thuật toán QuickSort

- Phân đoạn (chia):
  - 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

III. Bài toán áp dụng

4. Thuật toán QuickSort

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

III. Bài toán áp dụng

4. Thuật toán QuickSort

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

i	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
KQ		**2**	**1**	**3**	**8**	**7**	**6**	**9**

III. Bài toán áp dụng

4. Thuật toán QuickSort

- 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)

III. Bài toán áp dụng

4. Thuật toán QuickSort

- 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)

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

III. Bài toán áp dụng

4. Thuật toán QuickSort

- 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))

IV. Bài tập

Cho mảng A={3, 5, 8, 9, 4, 2, 7, 5, 3,9,8}

1. Thực hiện từng bước thuật toán MIN, MAX với mảng A.
2. Thực hiện thuật toán QuickSort và thể hiện kết quả từng bước với mảng A.
3. Thực hiện từng bước thuật toán tìm kiếm nhị phân các giá trị x=5, 6, 7 với mảng đã sắp xếp ở bài 2.
4. Thực hiện thuật toán MergeSort và thể hiện kết quả từng bước với mảng A.
5. Cài đặt thuật toán tìm kiếm nhị phân, đánh giá bằng thực nghiệm và so sánh với lý thuyết.
6. Cài đặt thuật toán MIN-MAX, đánh giá bằng thực nghiệm và so sánh với lý thuyết.
7. Cài đặt chương trình QuickSort, đánh giá bằng thực nghiệm và so sánh với lý thuyết.
8. Cài đặt chương trình MergeSort, đánh giá bằng thực nghiệm và so sánh với lý thuyết.
9. Thử nghiệm QuickSort và MergeSort trên cùng các bộ dữ liệu, so sánh thời gian thực hiện các thuật toán đó.

NỘI DUNG BÀI HỌC

I. Giới thiệu
II. Lược đồ chung
III. Bài toán áp dụng
IV. Bài tập