CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CHƯƠNG 2: TÌM KIẾM VÀ SẮP XẾP

Nội dung

- 2.2. Các giải thuật sắp xếp nội
  - 2.2.1. Phương pháp chọn trực tiếp – SelectionSort
  - 2.2.2. Phương pháp chèn trực tiếp – InsertionSort
  - 2.2.3. Phương pháp đổi chỗ trực tiếp
  - 2.2.4. Phương pháp nổi bọt – BubbleSort
  - 2.2.5. Phương pháp trộn trực tiếp - MergeSort
  - 2.2.6. Phương pháp phân hoạch – QuickSort

Khái quát về sắp xếp

- Thuận tiện và giảm thiểu thời gian thao tác, đặc biệt là tìm kiếm
- Là một trong những thao tác cần thiết và thường gặp trong quá trình lưu trữ, quản lý dữ liệu.
- Thứ tự dữ liệu có thể là thứ tự tăng (không giảm dần) hoặc thứ tự giảm (không tăng dần). Chúng ta sẽ sắp xếp dữ liệu theo thứ tự tăng.
- Phân chia các thuật toán sắp xếp thành hai nhóm:
  - Các giải thuật sắp xếp nội (dữ liệu trong bộ nhớ trong),
  - Các giải thuật sắp xếp ngoại (dữ liệu trong bộ nhớ ngoài).
- Sắp xếp dựa trên khóa (key) của dữ liệu, do đó cấu trúc của 1 phần tử tương tự như tìm kiếm (xem lại bài trước).

Thuật giải sắp xếp chọn (Selection Sort)

- Ý tưởng:
  - Xét dãy A có thứ tự tăng, phần tử A_i luôn là min(A_i, A_{i+1},…, A_N). Ý tưởng của thuật giải chọn trực tiếp mô phỏng một trong những cách sắp xếp tự nhiên nhất trong thực tế: chọn phần tử nhỏ nhất trong N phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu dãy hiện hành; sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn N-1 phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2; lặp lại quá trình trên cho dãy hiện hành đến khi dãy hiện hành chỉ còn 1 phần tử. Dãy ban đầu có N phần tử, vậy tóm tắt ý tưởng thuật giải là: thực hiện N-1 lượt việc đưa phần tử nhỏ nhất trong dãy hiện hành về vị trí đúng ở đầu dãy.

Thuật giải sắp xếp chọn (Selection Sort)

- Mô tả thuật giải:
  - - Bước 1: i = 0;
  - - Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[N-1].
  - - Bước 3 : Hoán vị a[min] và a[i]
  - - Bước 4 : Nếu i < N-1 thì
    - i = i+1;
    - Lặp lại Bước 2
    - Ngược lại: Dừng. //N-1 phần tử đã nằm đúng vị trí.

Thuật giải sắp xếp chọn (Selection Sort)

- Ví dụ:

Cho mảng a như sau:

i	0	1	2	3	4	5	6	7
a[i]	25	7	15	8	18	6	4	0

Các bước chạy để sắp xếp mảng a có thứ tự tăng như sau:

i = 0: 0 7 15 8 18 6 4 25

i = 1: 0 | 4 15 8 18 6 7 25

i = 2: 0 4| 6 8 18 15 7 25

i = 3: 0 4 6 | 7 18 15 8 25

i = 4: 0 4 6 7 | 8 15 18 25

i = 5: 0 4 6 7 8 | 15 18 25

i = 6: 0 4 6 7 8 15 |18 25

i = 7: Dừng

Thuật giải sắp xếp chọn (Selection Sort)

- Cài đặt thuật giải sắp xếp chọn trực tiếp thành hàm

```
void SelectionSort(int a[],int N )
{ int i, j, Cs_min;    //chỉ số phần tử nhỏ nhất trong dãy hiện hành
     for (i=0; i<N-1 ; i++)
          {  Cs_min = i;
             for( j = i+1; j <N ; j++)
             if (a[j ] < a[Cs_min])
             Cs_min = j; // ghi nhận vị trí phần tử hiện nhỏ nhất
             Hoanvi(a[Cs_min], a[i]);
          }
}
```

Trong đó hàm Hoanvi là đổi chỗ 2 phần tử (tự làm)

Thuật giải sắp xếp chọn (Selection Sort)

- Đánh giá thuật giải

  Có thể thấy rằng ở lượt thứ i, bao giờ cũng cần (N-i) lần so sánh để xác định phần tử nhỏ nhất hiện hành. Số lượng phép so sánh này không phụ thuộc vào tình trạng của dãy số ban đầu, do vậy trong mọi trường hợp có thể kết luận số lần so sánh là:

$$\sum_{i=1}^{N-1}(N-i) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$$

- T/h tốt nhất:
  - Số phép so sánh: N(N-1)/2
  - Số phép gán: 0
- T/h xấu nhất:
  - Số phép so sánh: N(N-1)/2
  - Số phép gán: 3N(N-1)/2

Thuật giải sắp xếp chèn (Insertion Sort)

- Ý tưởng

Giả sử có một dãy $A_0, A_1, \ldots, A_{N-1}$ trong đó i phần tử đầu tiên $A_0, A_1, \ldots, A_{i-1}$ đã có thứ tự. Ý tưởng chính của phương pháp chèn trực tiếp là tìm cách chèn phần tử A_i vào vị trí thích hợp của đoạn đã được sắp để có dãy mới $A_0, A_1, \ldots, A_i$ có thứ tự.

Vị trí này có thể là:

- Trước A_0 , hoặc sau A_{i-1}
- Giữa hai phần tử A_{k-1} và A_k thỏa $A_{k-1} \leq A_i < A_k$ $(1 \leq k \leq i-1)$.

Dãy ban đầu $A_0, A_1, \ldots, A_{N-1}$, ta có thể xem như đã có đoạn gồm một phần tử A_0 đã được sắp, sau đó thêm A_1 vào đoạn A_0 sẽ có đoạn $A_0\ A_1$ được sắp; tiếp tục thêm A_2 vào đoạn $A_0\ A_1$ để có đoạn $A_0\ A_1\ A_2$ được sắp; tiếp tục cho đến khi thêm xong A_{N-1} vào đoạn $A_0\ A_1 \ldots A_{N-2}$ sẽ có dãy $A_0\ A_1 \ldots A_{N-1}$ được sắp.

Thuật giải sắp xếp chèn (Insertion Sort)

- Mô tả thuật giải:

  Bước 1: i = 1; *// đoạn có 1 phần tử a[0] đã được sắp*

  Bước 2: x = a[i]; Tìm vị trí **pos** thích hợp trong đoạn a[0] đến a[i-1] để chèn x vào.

  Bước 3: Dời các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chỗ cho a[i].

  Bước 4: a[pos] = x; *// có đoạn a[0]..a[i] đã được sắp*

  Bước 5: i = i+1;

  Nếu i < N-1 : Lặp lại Bước 2.

  Ngược lại : Dừng.

Thuật giải sắp xếp chèn (Insertion Sort)

- Xét dãy a trong ví dụ trên. Chạy thuật giải sắp xếp chèn

i = 1: 7 | 25 15 8 18 6 4 0

i = 2: 7 15 | 25 8 18 6 4 0

i = 3: 7 8 15 | 25 18 6 4 0

i = 4: 7 8 15 18 | 25 6 4 0

i = 5: 6 7 8 15 18 | 25 4 0

i = 6: 4 6 7 8 15 18 | 25 0

i = 7: 0 4 6 7 8 15 18 |25

Thuật giải sắp xếp chèn (Insertion Sort)

- Cài đặt Thuật giải sắp xếp chèn trực tiếp thành hàm

```
void InsertionSort(int a[], int N )
{  int pos, i, x;
     for( i=1 ; i<N ; i++) //đoạn a[0], ... a[i-1] đã sắp
          { x = a[i]; //lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các
            phần tử.
          pos = i-1;
while((pos >= 0)&&(a[pos] > x)) //tiến về trái tìm vị trí chèn x
               {// dời chỗ các phần tử sẽ đứng sau x
                    a[pos+1] = a[pos];
                    pos--;
               }
          a[pos+1] = x;// chèn x vào dãy
          }
}
```

Thuật giải sắp xếp chèn (Insertion Sort)

- Đánh giá thuật giải

Đối với thuật giải chèn trực tiếp, các phép so sánh xảy ra trong mỗi vòng lặp while tìm vị trí thích hợp pos, và mỗi lần xác định vị trí đang xét không thích hợp, sẽ dời chỗ phần tử a[pos] tương ứng. Thuật giải thực hiện tất cả N-1 vòng lặp while , do số lượng phép so sánh và dời chỗ này phụ thuộc vào tình trạng của dãy số ban đầu, nên chỉ có thể ước lược trong từng trường hợp như sau:

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{N-1} 1 = N-1$	$\sum_{i=1}^{N-1} 1 = 2(N-1)$
Xấu nhất	$\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$	$\sum_{i=1}^{N-1} (i+1) = \frac{N(N+1)}{2} - 1$

Thuật giải sắp xếp đổi chỗ trực tiếp (Interchange Sort)

- Để sắp xếp một dãy số, ta có thể xét các nghịch thế (tức là các cặp phần tử A[j], A[i] với A[j]<A[i] và j>i) có trong dãy và làm triệt tiêu dần chúng đi.
- Ý tưởng:
  - Xuất phát từ đầu dãy, tìm tất cả các nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ phần tử này với phần tử tương ứng trong cặp nghịch thế. Lặp lại việc xử lý trên với các phần tử tiếp theo trong dãy.
  - Nói cách khác: Phần tử nhỏ hơn phải đứng trước phần tử lớn hơn.

Thuật giải sắp xếp đổi chỗ trực tiếp (Interchange Sort)

- Mô tả thuật giải:

  Bước 1: i=0; // bắt đầu từ đầu dãy

  Bước 2: j=i+1; //tìm các phần tử a[j]<a[i], j>i

  Bước 3:

  Trong khi j<N thực hiện

  Nếu a[j]<a[i]: a[i] ↔ a[j]; //xét cặp phần tử a[i], a[j]

  j = j+1;

  Bước 4: i=i+1;

  Nếu i<N-1: lặp lại bước 2

  Ngược lại: dừng

Thuật giải sắp xếp đổi chỗ trực tiếp (Interchange Sort)

- Ví dụ:
- Sắp xếp tăng dãy a (trong ví dụ trên) theo thuật giải đổi chỗ trực tiếp, kết quả các bước như sau:

  i = 0: **0** | 25 15 8 18 7 6 4

  i = 1: **0 4** | 25 15 18 8 7 6

  i = 2: **0 4 6** | 25 18 15 8 7

  i = 3: **0 4 6 7** | 25 18 15 8

  i = 4: **0 4 6 7 8** | 25 18 15

  i = 5: **0 4 6 7 8 15** | 25 18

  i = 6: **0 4 6 7 8 15 18** |25

Thuật giải sắp xếp đổi chỗ trực tiếp (Interchange Sort)

- Cài đặt

```
void InterchangeSort(int a[], int N )
{  int i, j;
for (i = 0 ; i<N-1 ; i++)
for (j =i+1; j < N ; j++)
     if(a[i ]>a[j]) Hoanvi(a[i],a[j]);
}
```

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{N-1}(N-i)=\frac{N(N-1)}{2}$	0
Xấu nhất	$\frac{N(N-1)}{2}$	$3\frac{N(N-1)}{2}$

Thuật giải sắp xếp nổi bọt (Bubble Sort)

- Ý tưởng
- Coi các phần tử của dãy như là các bong bóng nước chạy từ đáy bể chạy lên. Bong bóng nào nở ra to hơn (nhẹ hơn) thì nổi lên trên, nặng hơn thì nằm dưới. Xét hai bong bóng kề nhau, bóng nào nhẹ thì lên trên, nặng thì xuống dưới.

Thuật giải sắp xếp nổi bọt (Bubble Sort)

- Mô tả thuật giải:

  - Bước 1 : i = 0; // lần xử lý đầu tiên
  - Bước 2 : j = N-1; //Duyệt từ cuối dãy ngược về vị trí i

    Trong khi (j > i) thực hiện:

    Nếu a[j]<a[j-1]: a[j] ↔ a[j-1]; //xét cặp phần tử kế cận

    j = j-1;
  - Bước 3 : i = i+1; // lần xử lý kế tiếp

    Nếu i = N-1: Hết dãy. Dừng

    Ngược lại: lặp lại bước 2

Thuật giải sắp xếp nổi bọt (Bubble Sort)

- Kết quả của thuật giải khi thực hiện trên dãy số trong các ví dụ trên:

  i = 0: 0 | 25 7 15 8 18 6 4

  i = 1: 0 4 | 25 7 15 8 18 6

  i = 2: 0 4 6 | 25 7 15 8 18

  i = 3: 0 4 6 7 | 25 8 15 18

  i = 4: 0 4 6 7 8 | 15 25 18

  i = 5: 0 4 6 7 8 15 | 18 25

  i = 6: 0 4 6 7 8 15 18 | 25

Thuật giải sắp xếp nổi bọt (Bubble Sort)

- Cài đặt

```
void BubbleSort(int a[], int N )
{ int i, j;
    for (i = 0 ; i<N-1 ; i++)
    for (j =N-1; j >i ; j --)
    if(a[j]< a[j-1]) Hoanvi(a[j],a[j-1]);
}
```

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{N-1}(N-i)=\frac{N(N-1)}{2}$	0
Xấu nhất	$\frac{N(N-1)}{2}$	$3\frac{N(N-1)}{2}$

Thuật giải sắp xếp trộn (Merge Sort)

- Ý tưởng
  - Để sắp xếp dãy $a_0, a_1, ..., a_{N-1}$, thuật giải Merge Sort dựa trên nhận xét sau:
  - Mỗi dãy $a_0, a_1, ..., a_{N-1}$, bất kỳ đều có thể coi như là một tập hợp các dãy con liên tiếp mà mỗi dãy con đều đã có thứ tự. Ví dụ dãy 12, 2, 8, 5, 1, 6, 4, 15 có thể coi như gồm 5 dãy con không giảm (12); (2, 8); (5); (1, 6); (4, 15).
  - Dãy đã có thứ tự coi như có 1 dãy con.
  - Như vậy, để sắp xếp dãy là tìm cách làm giảm số dãy con không giảm của nó.

Thuật giải sắp xếp trộn (Merge Sort)

- Mấu chốt của vấn đề là cách phân hoạch dãy ban đầu thành các dãy con. Sau khi phân hoạch xong, dãy ban đầu sẽ được tách ra thành 2 dãy phụ theo nguyên tắc phân phối đều luân phiên. Trộn từng cặp dãy con của hai dãy phụ thành **một dãy** con của dãy ban đầu, ta sẽ nhận lại dãy ban đầu nhưng với số lượng dãy con ít nhất giảm đi một nửa. Lặp lại qui trình trên sau một số bước, ta sẽ nhận được 1 dãy chỉ gồm 1 dãy con không giảm. Nghĩa là dãy ban đầu đã được sắp xếp.
- Thuật giải trộn trực tiếp là phương pháp trộn đơn giản nhất. Việc phân hoạch thành các dãy con đơn giản chỉ là tách dãy gồm N phần tử thành N dãy con. Đòi hỏi của Thuật giải về tính có thứ tự của các dãy con luôn được thỏa trong cách phân hoạch này vì dãy gồm một phân tử luôn có thứ tự. Cứ mỗi lần tách rồi trộn, chiều dài của các dãy con sẽ được nhân đôi.

Thuật giải sắp xếp trộn (Merge Sort)

- Thuật giải

  - Bước 1: // Chuẩn bị

  k = 1; // k là chiều dài của dãy con trong bước hiện hành

  - Bước 2: Tách dãy $a_0, a_1, ..., a_{N-1}$, thành 2 dãy b, c theo nguyên tắc luân phiên từng nhóm k phần tử:

  b = $a_0, ., a_{k-1}$, $a_{2k}, ., a_{3k-1}$,

  c = $a_k, ., a_{2k-1}$, $a_{3k}, ., a_{4k-1}$, …

  - Bước 3: Trộn từng cặp dãy con gồm k phần tử của 2 dãy b, c vào a.

  - Bước 4:

    k = k*2;

    Nếu k < n thì trở lại bước 2.

    Ngược lại: Dừng

Thuật giải sắp xếp trộn (Merge Sort)

Cài đặt

```
void MergeSort(int a[], int N)
{   int p, pb, pc;  //các chỉ số trên các mảng a, b, c
    int i, k = 1;     //độ dài của dãy con khi phân hoạch
    do              // tách a thành b và c
      {  p = pb = pc = 0;
         while(p < N)
           {
                for(i = 0; (p < n)&&(i < k); i++)
                 b[pb++] = a[p++];
                for(i = 0; (p < n)&&(i < k); i++)
                c[pc++] = a[p++];
           }
      Merge(a, pb, pc, k); //trộn b và c thành a
      k *= 2;
    } while(k < N);
}
```

Thuật giải sắp xếp trộn (Merge Sort)

Trong đó hàm Merge có thể cài đặt như sau:

```
void Merge(int a[], int nb, int nc, int k)
{
int p, pb, pc, ib, ic, kb, kc;
p = pb = pc = 0; ib = ic = 0;
while((0 < nb)&&(0 < nc))
    {
      kb = min(k, nb); kc = min(k, nc);
      if(b[pb+ib] <= c[pc+ic])
                { a[p++] = b[pb+ib]; ib++;
                        if(ib == kb)
                                { for(; ic<kc; ic++)
                                    a[p++] = c[pc+ic];
                                    pb += kb; pc += kc; ib = ic = 0;
                                    nb -= kb; nc -= kc;
                                }
```

Thuật giải sắp xếp trộn (Merge Sort)

```
}
else //(ib != kb)
{
      a[p++] = c[pc+ic]; ic++;
      if(ic == kc)
{
for(; ib<kb; ib++)
a[p++] = b[pb+ib];
pb += kb;
pc += kc; ib = ic = 0;
nb -= kb; nc -= kc;
}
}
}
```

Thuật giải sắp xếp trộn (Merge Sort)

- Ví dụ minh họa thuật toán:

  Giả sử ta cần sắp xếp mảng A có 10 phần tử sau (N = 10):

  A: 41 36 32 47 65 21 52 57 70 50

  Ta thực hiện các lần phân phối và trộn các phần tử của A như sau:

  Lần 1: K = 1 (Độ dài mỗi đoạn con)

  Phân phối A thành 2 đoạn con B và C:

  A: 41 36 32 47 65 21 52 57 70 50

  B có số phần tử pb = 5

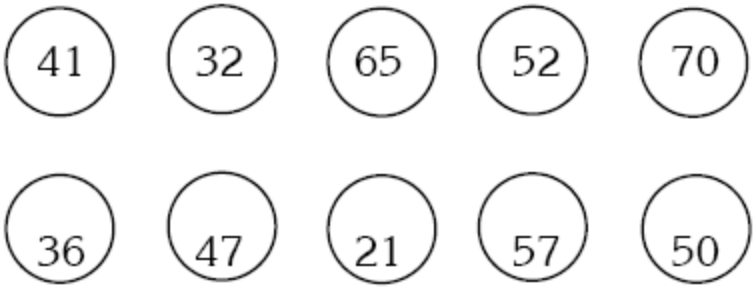
  41 32 65 52 70

  C có số phần tử pc = 5

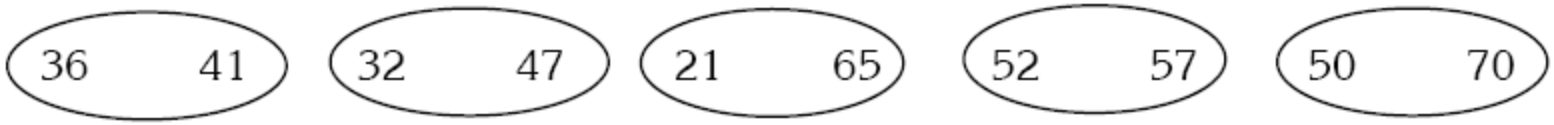
  36 47 21 57 50

Thuật giải sắp xếp trộn (Merge Sort)

Trộn B và C thành A: (Lần lượt từng cặp theo nguyên tắc: nhỏ đứng trước, lớn đứng sau)



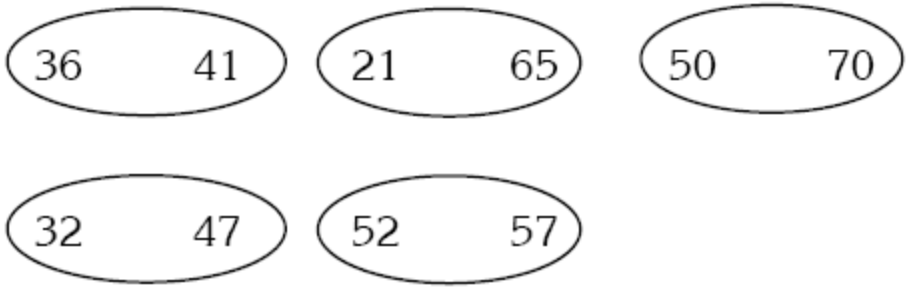
A



Lần 2: K = 2 (Độ dài mỗi đoạn con sau mỗi lần nhân đôi)

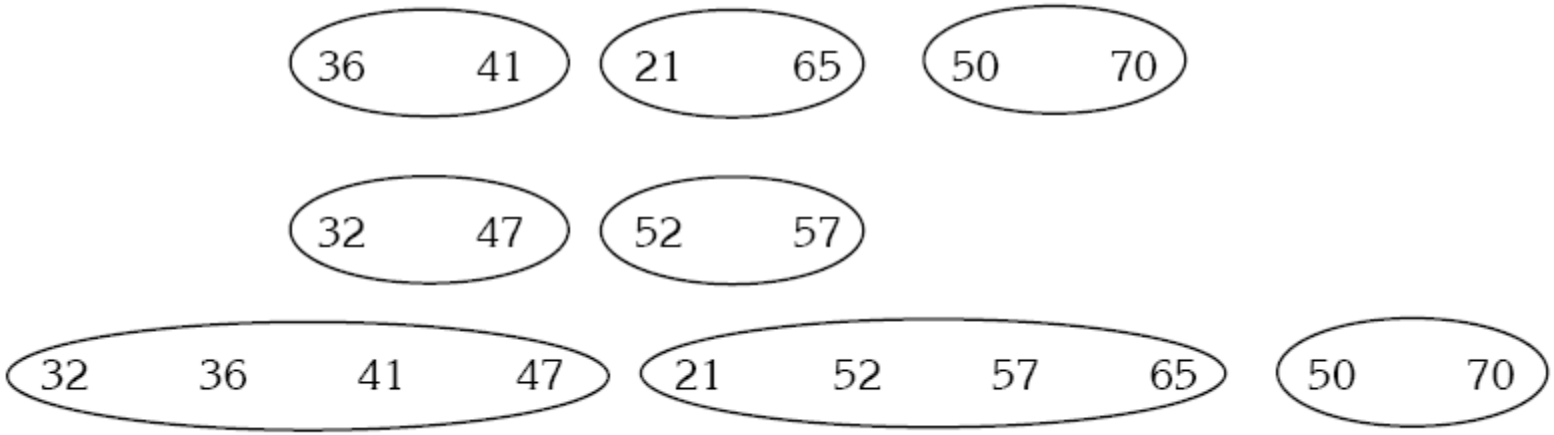
Phân phối A thành 2 đoạn con B và C:

Đoạn B có pb=6 phần tử, đoạn C có pc= 4 phần tử



Thuật giải sắp xếp trộn (Merge Sort)

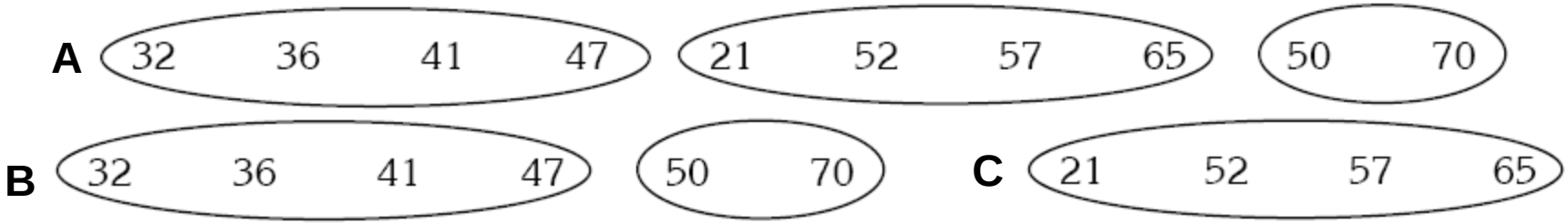
Trộn B và C thành A: (Lần lượt từng đoạn 2 phần tử theo nguyên tắc: nhỏ đứng trước, lớn đứng sau)



Lần 3: K = 4 (Độ dài mỗi đoạn con sau mỗi lần nhân đôi)

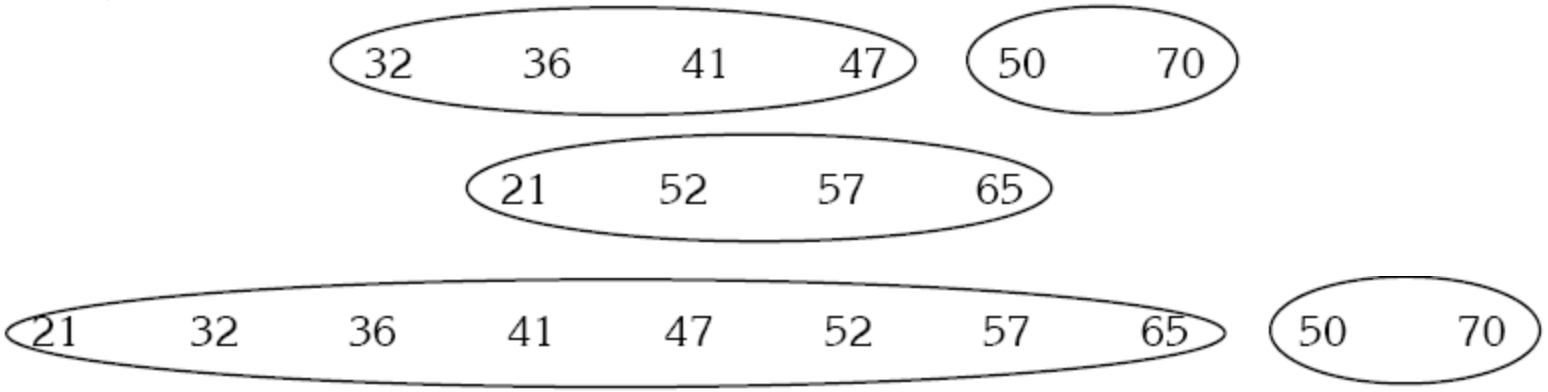
Phân phối A thành 2 đoạn con B và C:

Đoạn B có pb=6 phần tử, đoạn C có pc= 4 phần tử



Thuật giải sắp xếp trộn (Merge Sort)

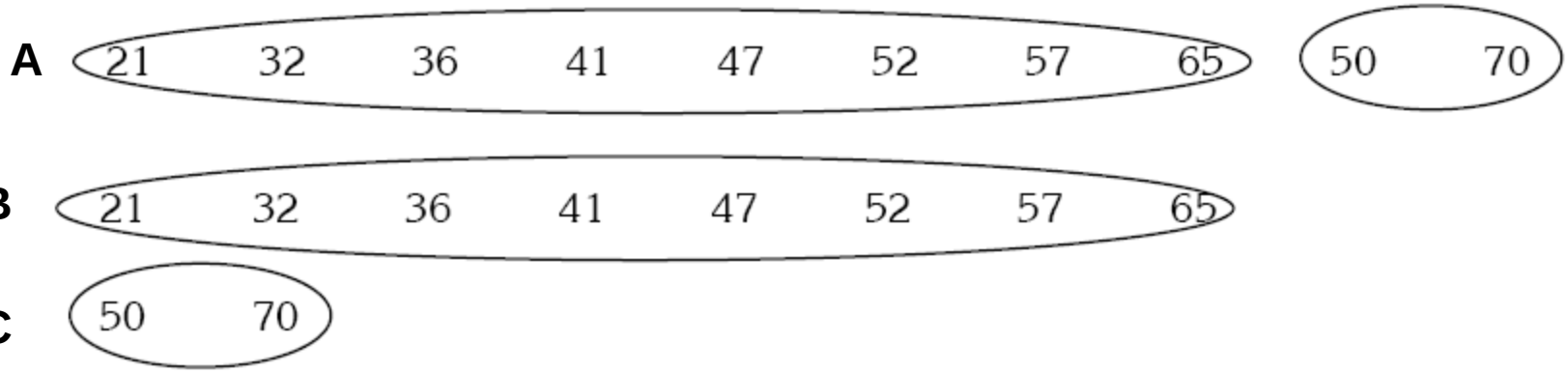
Trộn B và C thành A:



Lần 4: K = 8

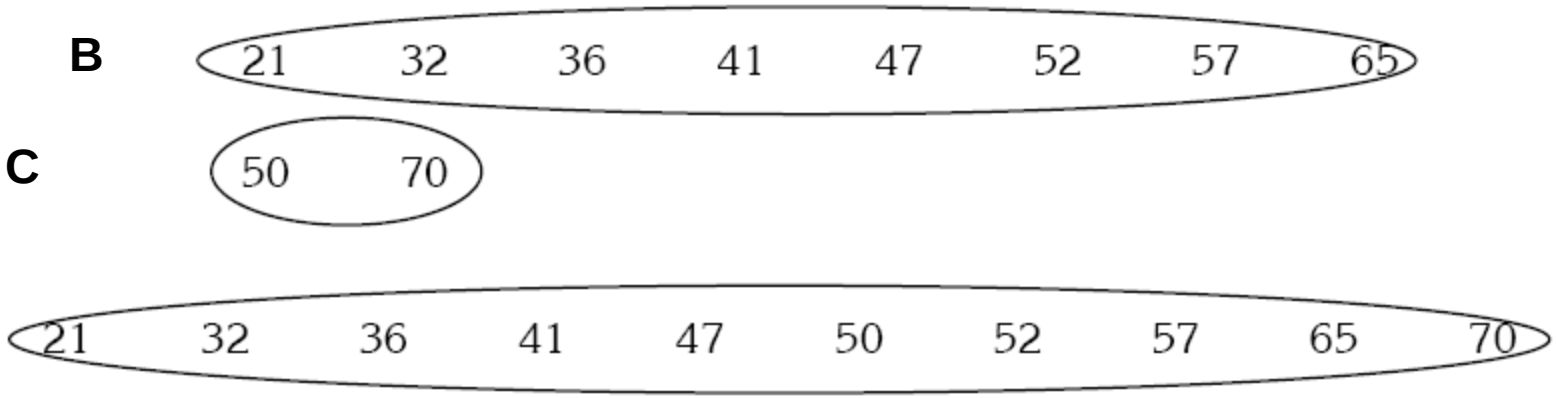
Phân phối A thành 2 đoạn con B và C:

Đoạn B có pb=8 phần tử, đoạn C có pc= 2 phần tử



Thuật giải sắp xếp trộn (Merge Sort)

Trộn B và C thành A:



Lần 5: K = 16>10: Dừng

Thuật giải sắp xếp trộn (Merge Sort)

- Đánh giá Thuật giải
  - Ta thấy rằng số lần lặp của bước 2 và bước 3 trong thuật giải MergeSort bằng log2n do sau mỗi lần lặp giá trị của k tăng lên gấp đôi. Dễ thấy, chi phí thực hiện bước 2 và bước 3 tỉ lệ thuận với N. Như vậy, chi phí thực hiện của thuật giải MergeSort sẽ là $O(N\log_2 N)$. Do không sử dụng thông tin nào về đặc tính của dãy cần sắp xếp, nên trong mọi trường hợp của Thuật giải chi phí là không đổi.
  - Nhược điểm lớn của Thuật giải là không tận dụng những thông tin đặc tính của dãy cần sắp xếp. Ví dụ trong trường hợp dãy đã có thứ tự sẵn.
  - Thực tế người ta ít chọn phương pháp trộn trực tiếp mà cải tiến nó gọi là phương pháp **trộn tự nhiên**. (Tự đọc thêm)

Thuật giải sắp xếp nhanh (Quick Sort)

- Ý tưởng

Để sắp xếp dãy $a_0, a_1, ..., a_{N-1}$ thuật giải QuickSort dựa trên việc phân hoạch dãy ban đầu thành hai phần :

- Dãy con 1: Gồm các phần tử $a_0 .. a_j$ có giá trị không lớn hơn x
- Dãy con 2: Gồm các phần tử $a_i .. a_{N-1}$ có giá trị không nhỏ hơn x

với x là giá trị của một phần tử tùy ý trong dãy ban đầu.

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

- $a_k < x$, với k = 0..j
- $a_k = x$, với k = j+1...i-1
- $a_k > x$, với k = i...N-1

trong đó dãy con thứ 2 đã có thứ tự, nếu các dãy con 1 và 3 chỉ có 1 phần tử thì chúng cũng đã có thứ tự, khi đó dãy ban đầu đã được sắp.

Thuật giải sắp xếp nhanh (Quick Sort)

- Ngược lại, nếu các dãy con 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các dãy con 1, 3 được sắp. Để sắp xếp dãy con 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 .
- Thuật giải phân hoạch dãy a_L, a_{L+1}, .,a_R thành 2 dãy con:

Bước 1 : Chọn tùy ý một phần tử a[k] trong dãy là giá trị mốc, $L \le k \le R$:
x = a[k]; i = L; j = R;

Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ :

Trong khi (a[i]<x) i++;

Trong khi (a[j]>x) j--;

Nếu i< j // $a[i] \ge x \ge a[j]$ mà a[j] đứng sau a[i]

Hoán vị (a[i],a[j]);

Bước 3 :

Nếu i < j: Lặp lại Bước 2.//chưa xét hết mảng

Nếu $i \ge j$: Dừng

Thuật giải sắp xếp nhanh (Quick Sort)

- Nhận xét: Về nguyên tắc, có thể chọn giá trị mốc x là một phần tử tùy ý trong dãy, nhưng để đơn giản, dễ diễn đạt thuật giải, phần tử có vị trí giữa thường được chọn, khi đó k = (l+r)/ 2.
- Thuật giải sắp xếp phân hoạch

Sắp xếp dãy a_L ,a_{L+1}, .,a_R

Có thể phát biểu thuật giải sắp xếp QuickSort một cách đệ qui như sau :

- Bước 1 : Phân hoạch dãy $a_L \ldots a_R$ thành các dãy con :
  - Dãy con 1 : $a_L .. a_j \leq x$
  - Dãy con 2 : $a_{j+1} .. a_{i-1} = x$
  - Dãy con 3 : $a_i .. a_R \geq x$
- Bước 2 : Nếu (L < j) // dãy con 1 có nhiều hơn 1 phần tử

  Phân hoạch dãy $a_L .. a_j$

  Nếu (i < R) // dãy con 3 có nhiều hơn 1 phần tử

  Phân hoạch dãy $a_i .. a_R$

Thuật giải sắp xếp nhanh (Quick Sort)

Ví dụ

i	0	1	2	3	4	5	6	7
a[i]	25	7	15	8	18	6	4	0

Phân hoạch đoạn [L..R], L = 0; R = 7, x = a[(L+R) /2] = a[3] = 8

0 7 4 6 18 8 15 25

Phân hoạch đoạn [L..R], L = 0; R = 3, x = a[(L+R) /2] = a[1] = 7

0 6 4 7

Phân hoạch đoạn [L..R], L = 0; R = 2, x = a[(L+R) /2] = a[1] = 6

0 4 6

Phân hoạch đoạn [L..R], L = 0; R = 1, x = a[(L+R) /2] = a[0] = 0

0 4

Thuật giải sắp xếp nhanh (Quick Sort)

Phân hoạch đoạn [L..R], L = 4; R = 7, x = a[(L+R) /2] = a[5] = 8

8 18 15 25

Phân hoạch đoạn [L..R], L= 5; R = 7, x = a[(L+R) /2] = a[6] = 15

15 18 25

Phân hoạch đoạn [L..R], L = 6; R = 7, x = a[(L+R) /2] = a[6] = 18

18 25

Kết quả: 0 4 6 7 8 15 18 25

Thuật giải sắp xếp nhanh (Quick Sort)

Cài đặt

```
    void QuickSort(int a[], int L, int R)
{ int i,j,x;
    x = a[(L+R)/2]; //chọn phần tử giữa làm mốc
    i =L; j = R;
    do {
            while(a[i] < x) i++;
            while(a[j] > x) j--;
            if(i <= j)
            { Hoanvi(a[i],a[j]);
                i++ ; j--; }
        } while(i <= j);
    if(L < j) QuickSort(a,L,j);
    if(i < R) QuickSort(a,i,R);
}
```

Thuật giải sắp xếp nhanh (Quick Sort)

- Đánh giá Thuật giải
  - Hiệu quả thực hiện của thuật giải QuickSort phụ thuộc vào việc chọn giá trị mốc.
  - Trường hợp tốt nhất xảy ra nếu mỗi lần phân hoạch đều chọn được phần tử median (phần tử lớn hơn (hay bằng) nửa số phần tử, và nhỏ hơn (hay bằng) nửa số phần tử còn lại) làm mốc, khi đó dãy được phân chia thành 2 phần bằng nhau và cần log2(N) lần phân hoạch thì sắp xếp xong. Nhưng nếu mỗi lần phân hoạch lại chọn nhằm phần tử có giá trị cực đại (hay cực tiểu) làm mốc, dãy sẽ bị phân chia thành 2 phần không đều: một phần chỉ có 1 phần tử, phần còn lại gồm (N-1) phần tử, do vậy cần phân hoạch n lần mới sắp xếp xong.

Tìm kiếm nhị phân (Binary Search)

- **Lưu ý:**
  - Thuật toán **tìm nhị phân** chỉ có thể vận dụng trong trường hợp **dãy/mảng đã có thứ tự**. Trong trường hợp **tổng quát** chúng ta chỉ có thể áp dụng thuật toán **tìm kiếm tuần tự**.
  - Các thuật toán đệ quy có thể ngắn gọn song tốn kém bộ nhớ để ghi nhận mã lệnh chương trình (mỗi lần gọi đệ quy) do vậy có thể làm cho chương trình chạy chậm lại. Trong thực tế, khi viết chương trình nếu có thể chúng ta **nên sử dụng thuật toán không đệ quy**.

BÀI TẬP