BÀI 7

SẮP XẾP

TÌNH HUỐNG DẪN NHẬP

- Trong cuộc sống hàng ngày các đối tượng luôn được sắp xếp theo trật tự nào đó: Theo thời gian, theo chiều cao, cận nặng, theo học lực, theo chất lượng v.v... Các tiêu chí sắp xếp đó là khoá của sắp xếp;
- Có 2 loại sắp xếp phổ biến: Sắp tăng dần và giảm dần;
- Thời gian sắp xếp một dãy các đối tượng phù thuộc vào số lượng các phần tử được sắp;
- Các giải thuật sắp xếp hoàn toàn dựa vào ý tưởng của con người trong cuộc sống hàng ngày.

MỤC TIÊU

- Mô tả đúng khái niệm, bản chất và mục đích của việc sắp xếp.
- Trình bày và thực hiện cài đặt một cách chính xác các thuật toán sắp xếp bao gồm các thuật toán sắp xếp cơ bản, các thuật toán sắp xếp theo phương pháp chia để trị, HeapSort ...
- Đánh giá đúng về các thuật toán sắp xếp và tìm ra thuật toán sắp xếp phù hợp cho từng bài toán.

NỘI DUNG

- Giới thiệu về bài toán sắp xếp;
- Một số phương pháp sắp xếp cơ bản;
- Các phương pháp sắp xếp theo kiểu chia để trị: Quick_sort và Merge_sort;
- Khối (Heap) và HeapSort.

1. GIỚI THIỆU VỀ BÀI TOÁN SẮP XẾP

1. GIỚI THIỆU VỀ BÀI TOÁN SẮP XẾP

- Trong thế vận hội Bắc Kinh năm 2008 có trên 200 đoàn tham gia thi đấu 35 môn để tranh 3 loại huy chương: Vàng, bạc và đồng. Trong 18 ngày thi đấu, chúng ta đều được ban tổ chức công bố thứ hạng của các đoàn. Đó thực chất là một bài toán sắp xếp thống kê;
- Và trong rất nhiều các ứng dụng tin học, yêu cầu về sắp xếp thường được xuất hiện với các mục đích khác nhau: sắp xếp dữ liệu trong máy tính để thuận lợi cho việc tìm kiếm hay sắp xếp kết quả xử lí để in trên bảng biểu v.v…

1. GIỚI THIỆU VỀ BÀI TOÁN SẮP XẾP

Sắp xếp là quá trình xử lý một danh sách các phần tử (hoặc các mẫu tin) để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu giữ tại mỗi phần tử.

- Cho trước một dãy số a_1, a_2,..., a_N được lưu trữ trong cấu trúc dữ liệu mảng;
- Sắp xếp dãy số a_1, a_2,..., a_N là thực hiện việc bố trí lại các phần tử sao cho hình thành được dãy mới a_{k1}, a_{k2},..., a_{kN} có thứ tự (giả sử xét thứ tự tăng) nghĩa là $a_{ki} > a_{ki-1}$. Hai thao tác so sánh và gán là các thao tác cơ bản của hầu hết các thuật toán sắp xếp.

2. MỘT SỐ PHƯƠNG PHÁP SẮP XẾP CƠ BẢN

2. MỘT SỐ PHƯƠNG PHÁP SẮP XẾP CƠ BẢN

- Chọn trực tiếp – Selection sort;
- Chèn trực tiếp – Insertion sort;
- Nổi bọt – Bubble sort;
- Các phương pháp sắp xếp theo kiểu chia để trị:
  - Quick_sort;
  - Merge_sort;
- Khối (Heap) và HeapSort.

2.1. SẮP XẾP KIỂU CHÈN – INSERTION SORT

Tư tưởng thuật toán:

- Giả sử có một dãy $a_1, a_2, ..., a_n$ trong đó i phần tử đầu tiên $a_1, a_2, ..., a_{i-1}, a_i$ đã có thứ tự;
- Chèn phần tử thứ i + 1 vào i phần tử đầu tiên của dãy đã có thứ tự để được dãy mới $a_1, a_2, ..., a_{i+1}$ trở nên có thứ tự:
  - Tìm vị trí của phần tử i + 1 trong k phần tử đầu tiên bằng cách vận dụng thuật giải tìm kiếm (tuần tự hoặc nhị phân);
  - Dời các phần tử từ vị trí chèn đến phần tử thứ i sang phải 1 vị trí và chèn phần tử i + 1 vào vị đó.

2.1. SẮP XẾP KIỂU CHÈN – INSERTION SORT

Nội dung của thuật toán:

Thuật toán này sắp xếp một dãy a_1, a_2,..., a_n theo thứ tự tăng dần.

Bước 1: i = 1;

Bước 2: đặt

x = a[i] ;

j = i – 1 ;

Bước 3: while (j > =0) && (x < a[j])

a[j+1] = a[j]; j--;

Bước 4: đặt a[j+1] = x ; i++;

Bước 5: nếu i < = n-1 lặp lại bước 2 ngược lại thì kết thúc.

2.1. SẮP XẾP KIỂU CHÈN – INSERTION SORT

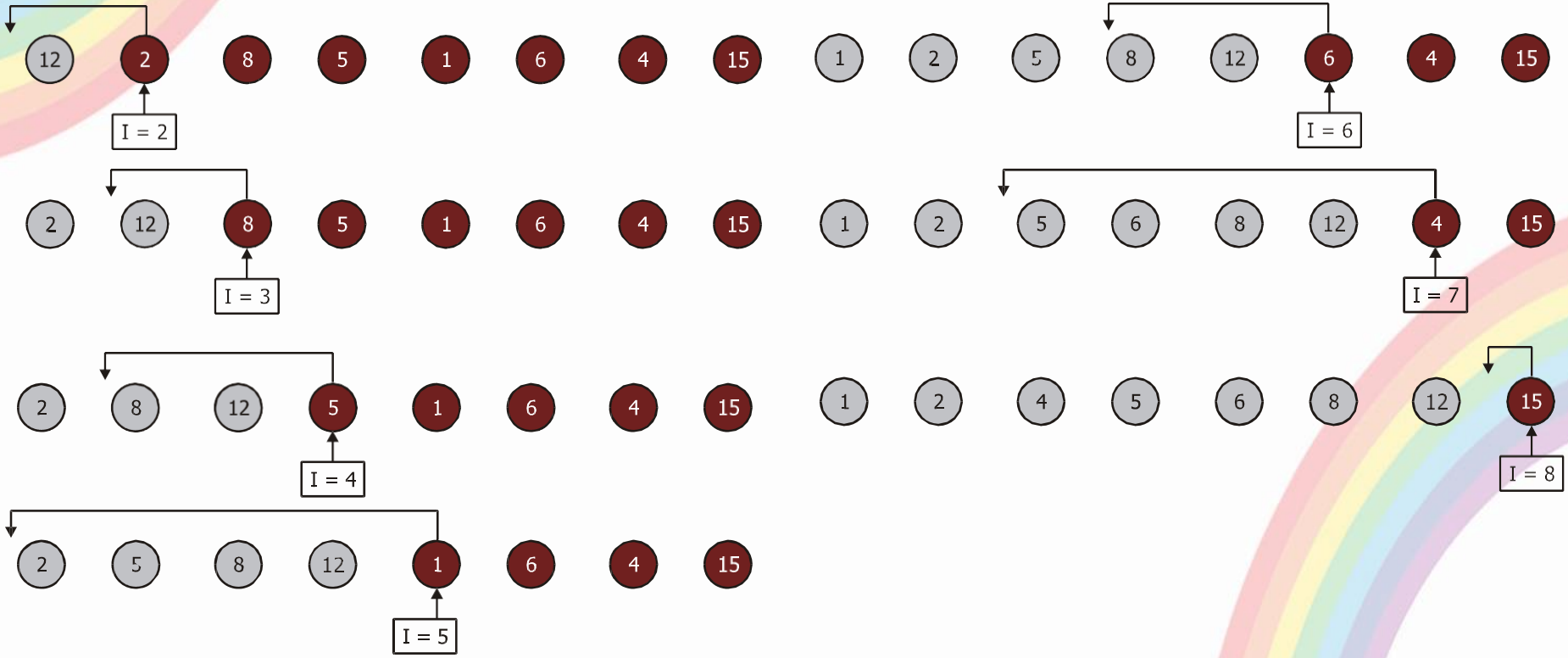
Cài đặt thuật toán:

```
#define     Max_Size…
typedef     Kieu_du_lieu KeyType;
typedef     struct KeyArray
{   KeyType Array[Max_Size];int n;};
KeyArray Sortinsert( KeyArray a)
{     int i,j;          KeyType x;        i = 1;
   while ( i <= a.n-1 )
   {     x = a.Array[i] ; j = i - 1;
                  while (( j>=0 )&&(x < a.Array[j]))
                        {   a.Array[j+1] = a.Array[j];
                         j--; }
                  a.Array[j+1] = x;              i++;
      }
    return a;
}
```

2.1. SẮP XẾP KIỂU CHÈN – INSERTION SORT

Ví dụ 7.1. Cho dãy số a: 12 2 8 5 1 6 4 15

Sắp xếp dãy số a này theo thứ tự tăng dần bằng phương pháp sắp xếp chèn ta thực hiện như sau:



2.1. SẮP XẾP KIỂU CHÈN – INSERTION SORT

Đánh giá thuật toán:

- Trường hợp tốt nhất, khi dãy phần tử cần sắp xếp có thứ tự tăng:
  - Số phép gán: $S_{gán} = 2(n - 1)$
  - Số phép so sánh: $S_{so\ sánh} = n - 1$
- Trường hợp tồi nhất, khi dãy phần tử cần sắp xếp có thứ tự giảm dần:
  - Số phép gán: $S_{gán} = n(n + 1)/2 - 1$
  - Số phép so sánh: $S_{so\ sánh} = n(n - 1)/2$

  Như vậy độ phức tạp của thuật toán sắp xếp chèn là $O(n^2)$ trong cả trường hợp tồi nhất và trường hợp trung bình.

2.2. SẮP XẾP KIỂU CHỌN – SELECTION SORT

Tư tưởng thuật toán:

- Ban đầu dãy gồm n phần tử không có thứ tự. Ta chọn phần tử nhỏ nhất trong n phần tử của dãy cần sắp xếp và đổi giá trị của nó với a[1], khi đó a[1] có giá trị nhỏ nhất trong dãy;
- Lần thứ 2, ta chọn phần tử nhỏ nhất trong n – 1 phần tử còn lại từ a[2], a[3], ..., a[n] và đổi giá trị của nó với a[2];
- Ở lượt thứ i ta chọn trong dãy a[i ... n] ra phần tử nhỏ nhất và đổi chỗ giá trị của nó với a[i];
- Tới lượt thứ n – 1 chọn trong 2 phần tử a[n – 1] và a[n] ra phần tử có giá trị nhỏ hơn và đổi giá trị của nó với a[n – 1]. Khi đó dãy thu được có thứ tự không giảm.

2.2. SẮP XẾP KIỂU CHỌN – SELECTION SORT

Cài đặt thuật toán:

```
KeyArray SortSelect(KeyArray a)
{    int i,j,min;          KeyType tg;       i=0;
     for (i=0;i<a.n-2;i++)
     {       min=i;
             for (j=i+1;j<a.n-1;j++)
                     if (a.Array[min]>a.Array[j])  min = j;
                     tg=a.Array[i];
                     a.Array[i]=a.Array[min];
                     a.Array[min]=tg;
      }
     return a;
}
```

2.2. SẮP XẾP KIỂU CHỌN – SELECTION SORT

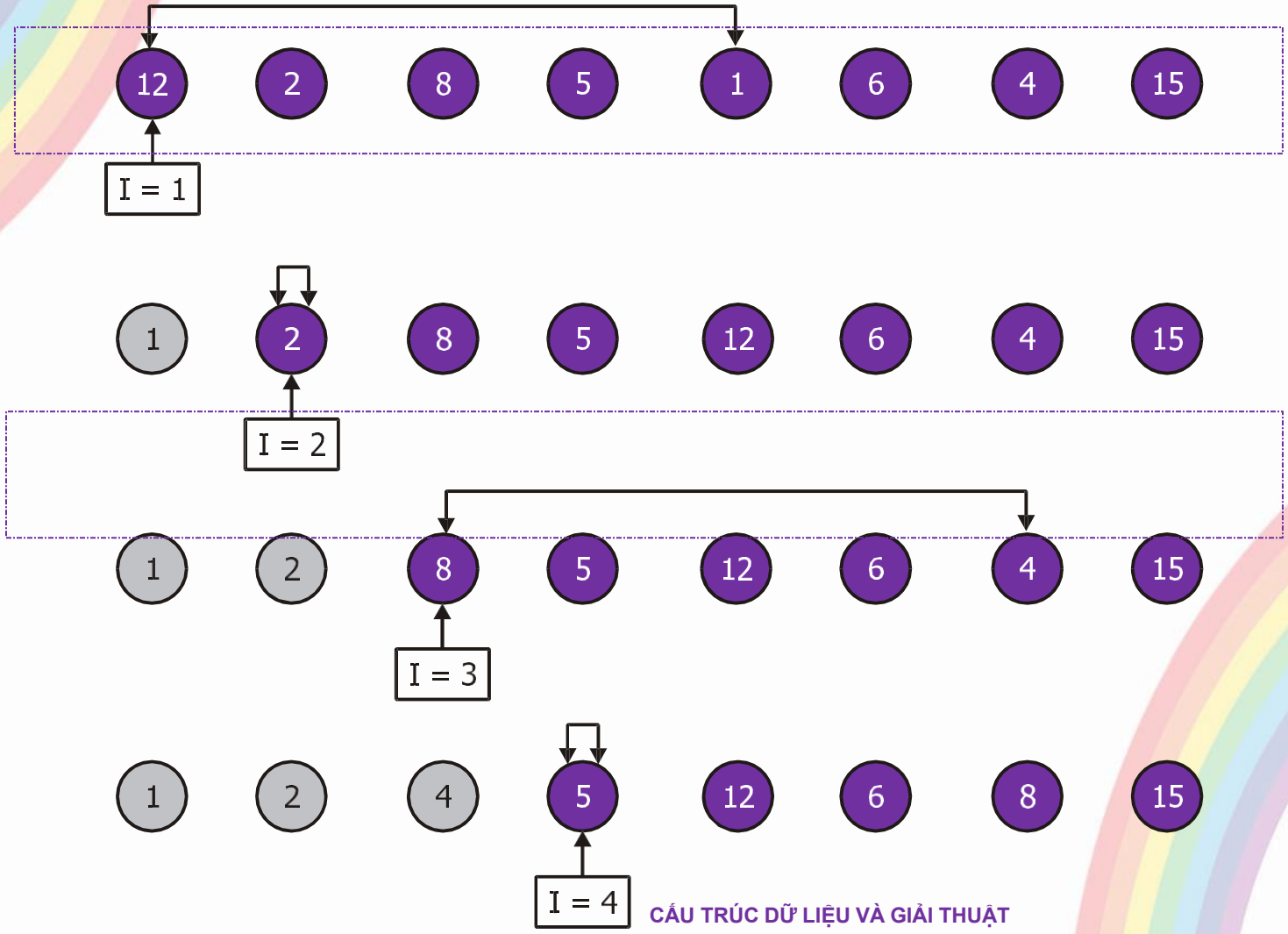
Nội dung và cách cài đặt của thuật toán:

Thuật toán sắp xếp n phần tử trong mảng a[1], a[2],..., a[n] theo thứ tự tăng dần:

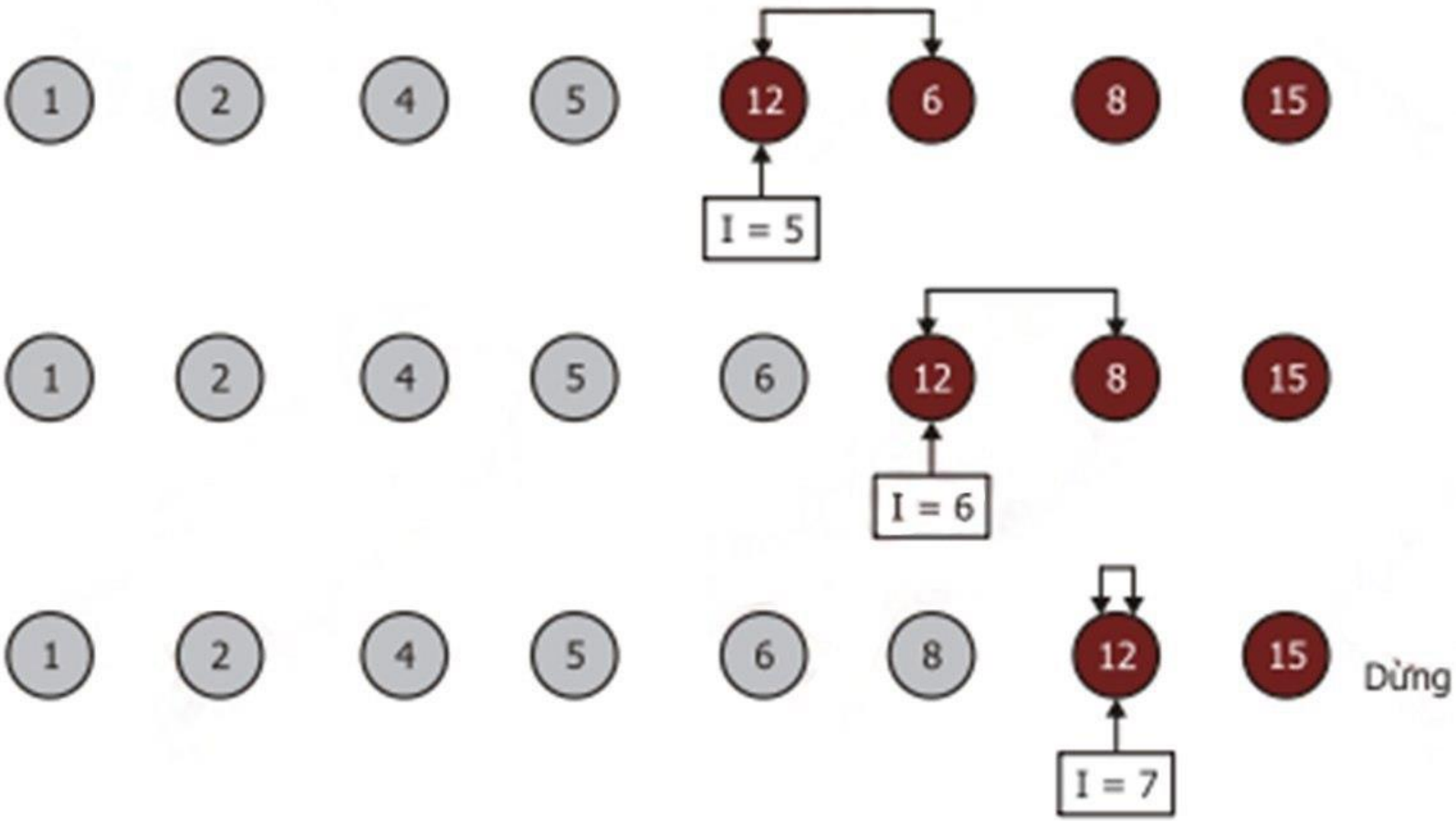
- Bước 1: i = 0;
- Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy từ a[i] tới a[n – 1];
- Bước 3: Hoán vị a[min] và a[i];
- Bước 4: Nếu i <= n – 2 thì i = i + 1, lặp lại bước 2, ngược lại thì dừng.

2.2. SẮP XẾP KIỂU CHỌN – SELECTION SORT

Ví dụ 7.2. Cho dãy số a: 12 2 8 5 1 6 4 15. Sắp xếp dãy số này theo thứ tự tăng dần bằng thuật toán sắp xếp lựa chọn được minh họa như sau:



2.2. SẮP XẾP KIỂU CHỌN – SELECTION SORT



2.2. SẮP XẾP KIỂU CHỌN – SELECTION SORT

Đánh giá thuật toán:

Tổng số phép toán so sánh là:

$$S_{so\ sánh} = (n - 1) + (n - 2) + ... + 2 + 1 = n * (n - 1) / 2$$

- Trong trường hợp tốt nhất là khi dãy đã có thứ tự tăng dần: Số phép gán $S_{gán} = 0$
- Trong trường hợp tồi nhất là khi dãy có thứ tự giảm dần:

$$S_{gán} = 3 * (n - 1) + n * (n - 1) / 2 = (n - 1) * (n + 6) / 2$$

Như vậy độ phức tạp của thuật toán sắp xếp lựa chọn này là $O(n^2)$.

2.3. SẮP XẾP KIỂU ĐỔI CHỖ

Tư tưởng thuật toán sắp xếp nổi bọt:

- Đi từ cuối mảng về đầu mảng;
- Trong quá trình đi nếu phần tử ở dưới (đứng phía sau) nhỏ hơn phần tử đứng ngay trên (trước) nó thì hai phần tử này sẽ được đổi chỗ cho nhau;
- Kết quả là phần tử nhỏ nhất (nhẹ nhất) sẽ được đưa về đầu dãy rất nhanh;
- Sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i. Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét và ta có dãy sắp xếp theo một thứ tự.

2.3. SẮP XẾP KIỂU ĐỔI CHỖ

Nội dung của thuật toán:

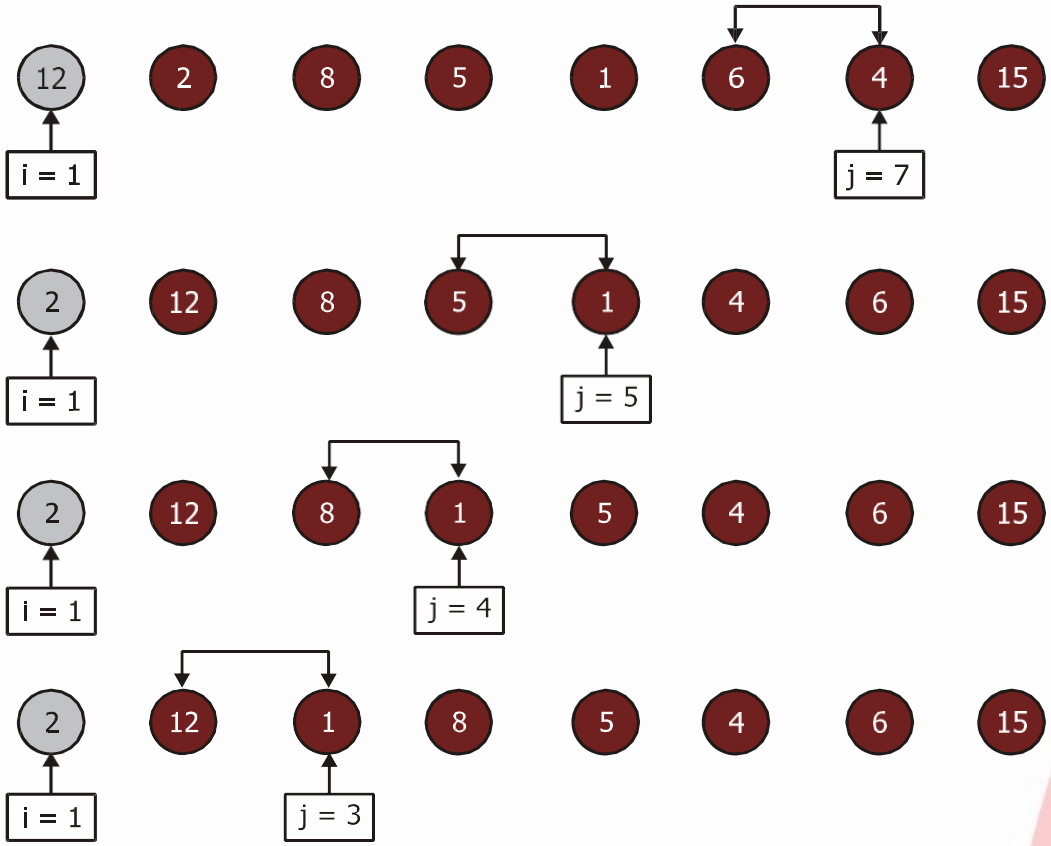
Thuật toán này sắp xếp kiểu nổi bọt một danh sách gồm các mục a[1], a[2],..., a[n] theo thứ tự tăng dần.

```
KeyType tg;
Bước 1: i=0;
for (i=0;i<n-2;i++)//lần lượt xử lý với mỗi i
Bước 2: duyệt từ cuối ngược về vị trí i
        for (j=n-1;j>=i+1;j--)
            if (a[j-1]>a[j])//hoán vị trí a[j] và a[j-1]
              {         tg=a[j-1];
                        a[j-1]=a[j];
                        a[j]=tg;
              }
```

2.3. SẮP XẾP KIỂU ĐỔI CHỖ

Ví dụ 7.3: Cho dãy số a: 12 2 8 5 1 6 4 15.

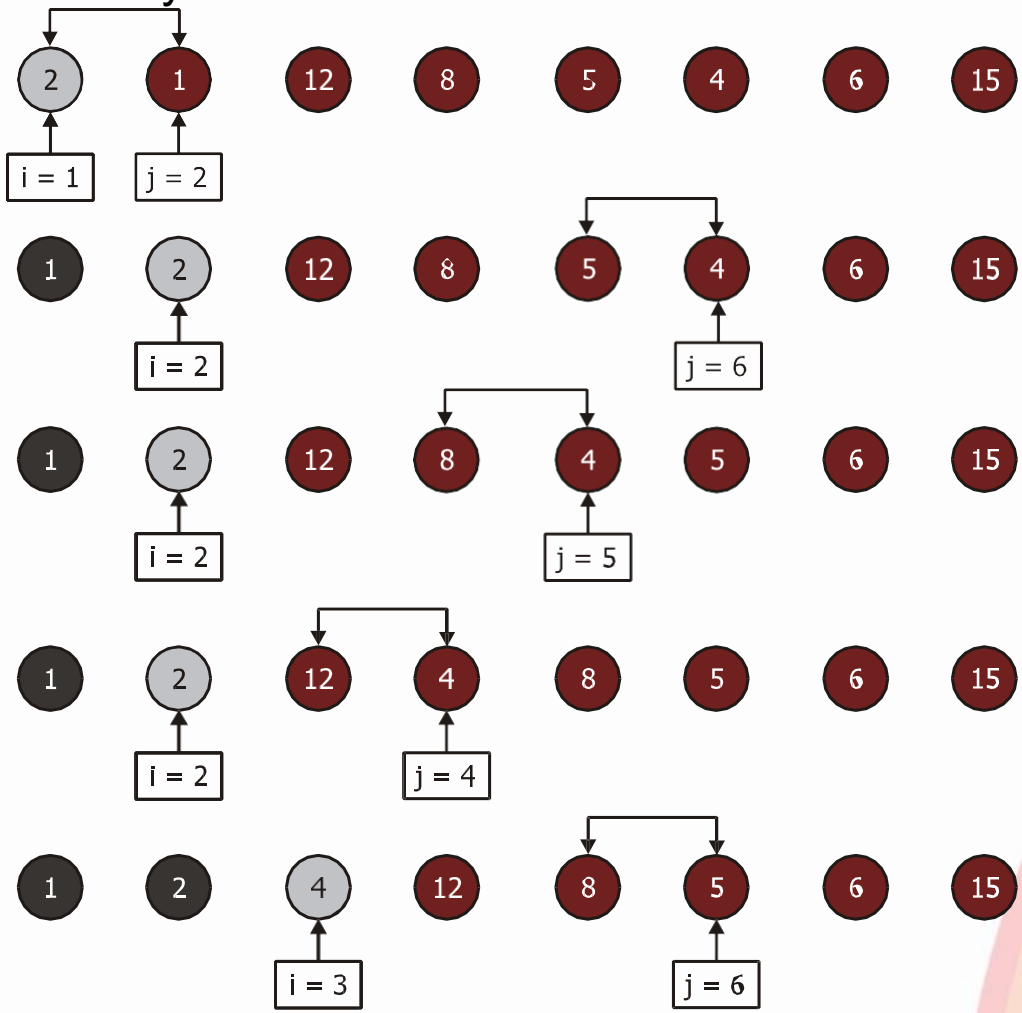
Sắp xếp dãy số a theo thứ tự tăng dần bằng thuật toán sắp xếp nổi bọt được minh họa dưới đây:



2.3. SẮP XẾP KIỂU ĐỔI CHỖ

Ví dụ 7.3: Cho dãy số a: 12 2 8 5 1 6 4 15.

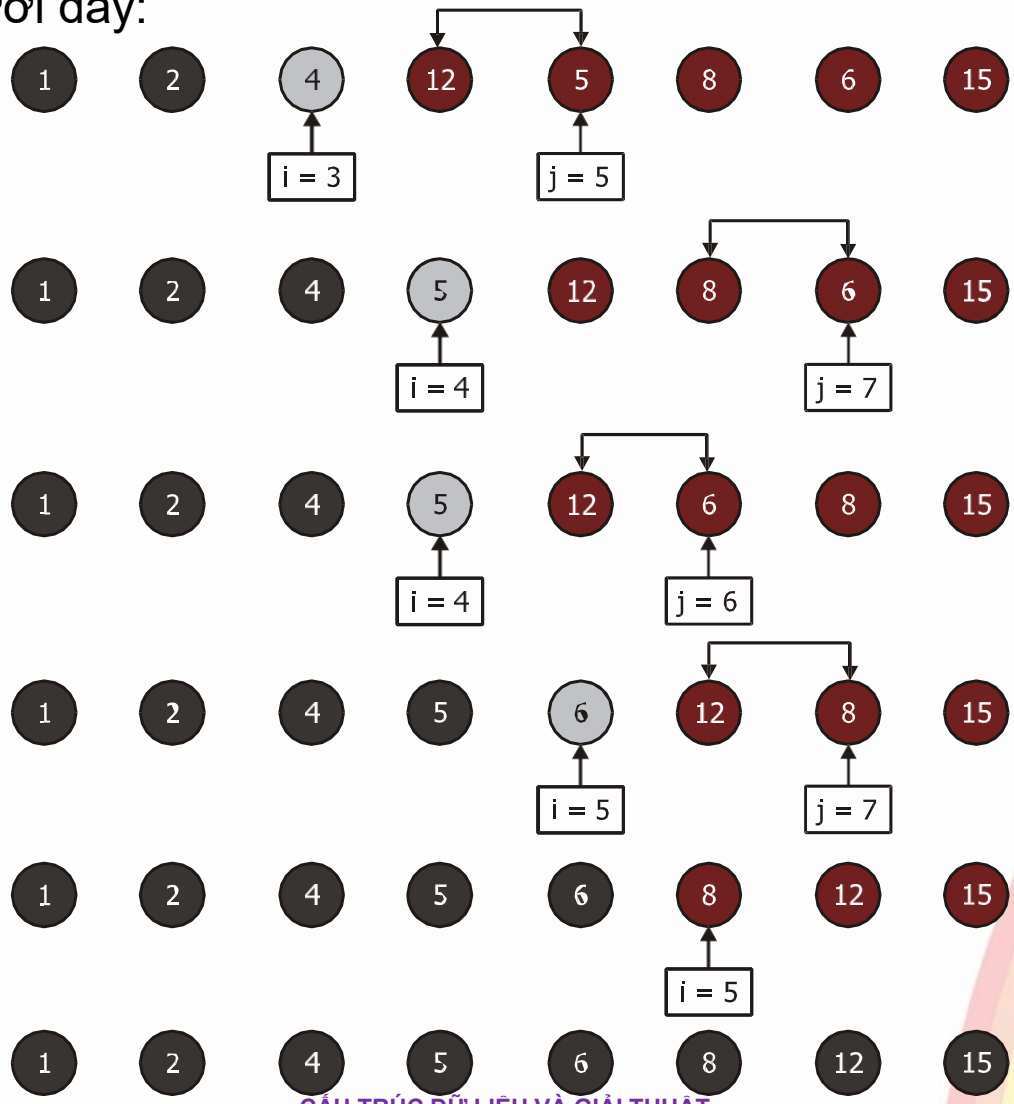
Sắp xếp dãy số a theo thứ tự tăng dần bằng thuật toán sắp xếp nổi bọt được minh họa dưới đây:



2.3. SẮP XẾP KIỂU ĐỔI CHỖ

Ví dụ 7.3: Cho dãy số a: 12 2 8 5 1 6 4 15.

Sắp xếp dãy số a theo thứ tự tăng dần bằng thuật toán sắp xếp nổi bọt được minh họa dưới đây:



2.3. SẮP XẾP KIỂU ĐỔI CHỖ

Cài đặt thuật toán:

```
KeyArray Sortbubble(KeyArray a)
{   int i,j;            KeyType tg;          i=0;
    for (i=0;i<a.n-2;i++)
        for (j=a.n-1;j>=i+1;j--)
            if (a.Array[j-1]>a.Array[j])
            {
                tg=a.Array[j-1];
                a.Array[j-1]=a.Array[j];
                a.Array[j]=tg;
            }
    return a;
}
```

2.3. SẮP XẾP KIỂU ĐỔI CHỖ

Đánh giá thuật toán:

- Trong trường hợp tốt nhất tức là dãy ban đầu đã có thứ tự tăng dần thì tổng số hoán vị Shv = 0.
- Trong trường hợp tồi nhất khi danh sách có thứ tự giảm dần:
  - Số lần so sánh là $S_{so\ sánh} = (n - 1) + (n - 2) + ... + 2 + 1$
    $= n(n - 2) / 2$
  - Số lần hoán vị là $S_{hoán\ vị} = (n - 1) + (n - 2) + ... + 2 + 1$
    $= n(n - 2) / 2$

Độ phức tạp của thuật toán này trong trường hợp tồi nhất là $O(n^2)$.

3. CÁC PHƯƠNG PHÁP SẮP XẾP THEO KIỂU CHIA ĐỂ TRỊ: QUICK_SORT VÀ MERGE_SORT

3. CÁC PHƯƠNG PHÁP SẮP XẾP THEO KIỂU CHIA ĐỂ TRỊ: QUICK_SORT VÀ MERGE_SORT

Tư tưởng chính của các phương pháp sắp xếp chia để trị:

- Ta nhận thấy rằng sắp xếp danh sách dài thì khó hơn là sắp xếp danh sách ngắn;
- Chia một danh sách ra thành hai phần có kích thước xấp xỉ nhau và thực hiện việc sắp xếp mỗi phần một cách riêng rẽ;
- Chia một bài toán thành nhiều bài toán tương tự như bài toán ban đầu nhưng nhỏ hơn và giải quyết các bài toán nhỏ này. Sau đó chúng ta tổng hợp lại để có lời giải cho toàn bộ bài toán ban đầu. Phương pháp này được gọi là “chia để trị” (*divide-and-conquer*);
- Do đó chúng ta sẽ đi nghiên cứu hai phương pháp sắp xếp là Qicksort và Mergesort.

3.1. QUICK_SORT

Tư tưởng của thuật toán:

- Giả sử để sắp xếp dãy số a gồm n phần tử: a_1, a_2, ..., a_n giải thuật Quicksort dựa trên việc phân hoạch dãy ban đầu thành hai phần:
  - Dãy con 1 gồm các phần tử: a_1,..., a_{i-1} có giá trị nhỏ hoặc bằng hơn a_i;
  - Dãy con 2 gồm các phần tử: a_{i+1},..., a_n có giá trị lớn hơn a_i;
- Với a_i là một phần tử bất kỳ trong dãy.
- Để sắp xếp dãy con 1 và 2, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu.

3.1. QUICK_SORT

Nội dung của thuật toán:

KeyType Item ,X;

Bước 1: first=0;

Bước 2: last = n-1;

Bước 3: Xuất phát từ đầu dãy phần tử để tìm có giá trị > X

i = first;

Bước 4: Xuất phát từ cuối dãy để tìm phần tử có giá trị < X

j = last;

Bước 5: Khóa chốt là phần tử giữa

X = a[(i +j)/ 2];

3.1. QUICK_SORT

Nội dung của thuật toán:

Bước 6: Lặp tìm phần tử có giá trị lớn hơn X và nhỏ hơn X

```
do
{   while (a[i] < X)                    i = i + 1;
    while (a[j] > X)                    j = j - 1;
      if (i <= j)
     {Item = a[i];          a[i] = a[j];          a[j] = Item;
      i = i + 1;            j = j - 1;
     }
}
while (i <= j);
```

Bước 7: Phân hoạch đệ quy dãy con từ phần tử thứ First đến phần tử thứ j

```
a=Quicksort(a, i, last);
```

Bước 8: Phân hoạch đệ quy dãy con từ phần tử thứ i đến phần tử thứ Last

```
a=Quicksort(a, first, j);
```

3.1. QUICK_SORT

Cài đặt thuật toán:

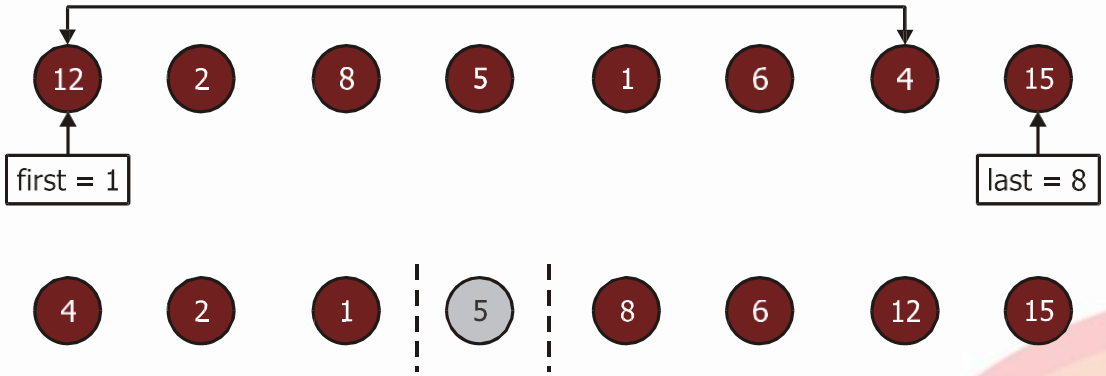
```
KeyArray Quicksort(KeyArray a,int first, int last)
{       int i, j;                       KeyType Item ,X;
        if      (first > last)
                printf("day con khong co phan tu");
       i =    first;          j = last;          X =   a.Array[(i        +j)/ 2];
       do  {          while   (a.Array[i]        <  X)             i   =  i + 1;
                      while   (a.Array[j]        >  X)             j   =  j - 1;
                      if (i      <= j)
                                    {Item = a.Array[i]; a.Array[i]  =  a.Array[j];
                                       a.Array[j] = Item;i = i + 1;      j = j - 1;}
            }while (i <= j);
        a=Quicksort(a, i, last);
        a=Quicksort(a, first, j);
        return a;
}
```

3.1. QUICK_SORT

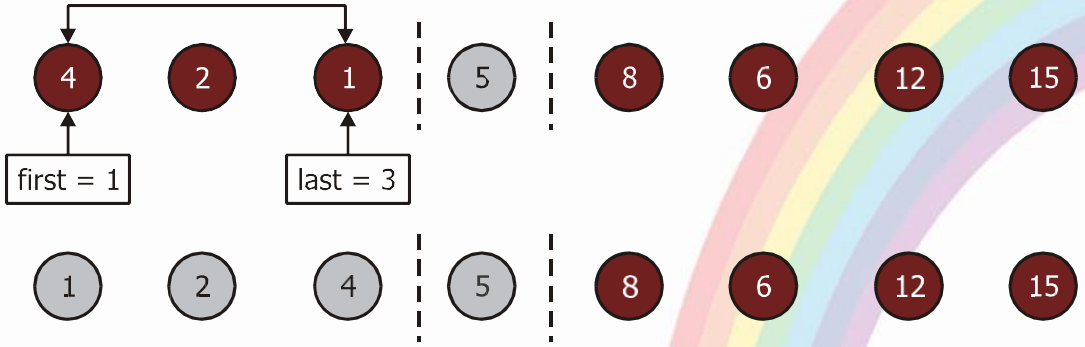
Ví dụ 7.4. Cho dãy số a: 12 2 8 5 1 6 4 15

Dãy a được sắp xếp theo thứ tự tăng dần bằng thuật toán sắp xếp quicksort như sau:

Phân hoạch đoạn first = 1, last = 8: x = A[4] = 5



Phân hoạch đoạn first = 1, last = 3: x = A[2] = 2

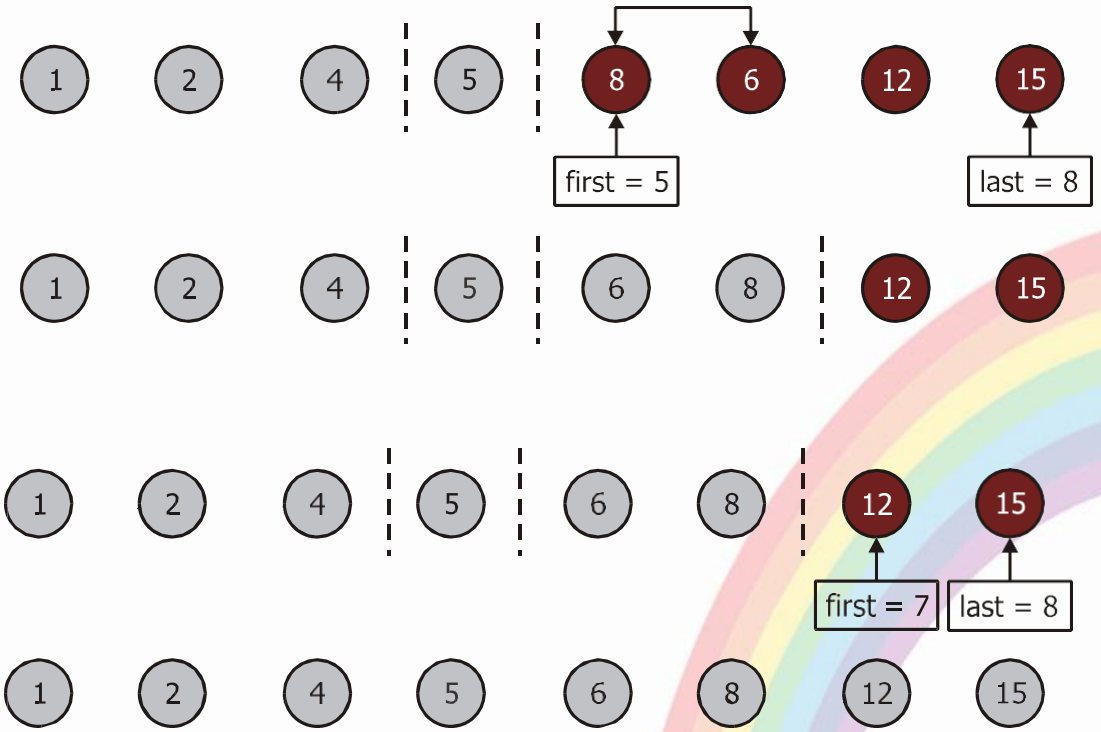


3.1. QUICK_SORT

Ví dụ 7.4. Cho dãy số a: 12 2 8 5 1 6 4 15

Dãy a được sắp xếp theo thứ tự tăng dần bằng thuật toán sắp xếp quicksort như sau:

Phân hoạch đoạn first = 5, last = 8: x = A[6] = 6



Phân hoạch đoạn first = 7, last = 8: x = A[7] = 6

3.1. QUICK_SORT

- Nếu chọn khóa chốt là phần tử đầu tiên hay phần tử cuối cuối cùng của danh sách thì độ phức tạp của Quicksort là tồi nhất với danh sách được sắp xếp theo thứ tự từ phần tử đầu tiên đến cuối cùng;
- Nếu chọn khóa chốt là phần tử ở giữa danh sách thì Quicksort có độ phức tạp tồi nhất với danh sách được sắp xếp bắt đầu từ phần tử ở giữa danh sách trở về hai đầu của danh sách;
- Nếu chọn khóa chốt là phần tử ở vị trí ngẫu nhiên trong danh sách thì thật khó tìm ra bộ dữ liệu khiến cho Quicksort suy biến.

3.1. QUICK_SORT

Đánh giá thuật toán:

- Trường hợp tốt nhất: Độ phức tạp tính toán của Quicksort là O(nlgn);
- Nếu mỗi lần chọn khóa chốt để phân đoạn chọn phần tử có giá trị cực đại (hay cực tiểu) là khóa chốt, 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 n lần phân đoạn mới sắp xếp xong, khi đó độ phức tạp tính toán của Quicksort là O(n2). Thời gian thực hiện giải thuật quicksort trung bình là O(nlgn).

3.2. MERGESORT

Tử tưởng của thuật toán Mergesort:

- 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 quy 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;
- Các thuật toán sắp xếp bằng phương pháp trộn bao gồm:
  - Thuật toán sắp xếp trộn thẳng hay trộn trực tiếp (straight merge sort);
  - Thuật toán sắp xếp trộn tự nhiên (natural merge sort).

3.2.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

Tư tưởng:

- Tách dãy gồm n phần tử thành n dãy con;
- Sau đó trộn tương ứng từng cặp dãy con thành các dãy con mới có chiều dài bằng 2 để đưa dãy ban đầu thành n/2 dãy con;
- Như vậy, sau mỗi lần phân phối và trộn các dãy con thì số dãy con sẽ giảm đi một nửa, đồng thời chiều dài mỗi dãy con sẽ tăng gấp đôi;
- Do đó, sau $Log_2(N)$ lần phân phối và trộn thì dãy ban được sắp xếp có thứ tự.

3.2.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

Nội dung và cách cài đặt của thuật toán:

"Merge(KeyArray A, KeyArray B, int a, int b, int c)": Trộn mảng A[a...b] với mảng A[b + 1...c] để được mảng B[a...c].

```
void  Merge( KeyArray A,KeyArray B,int a,int                    b,int c)
{     int i, j, p;              p = a; i = a; j =   b+1;
      while ((i <=b)&&(j <= c))
      {           if    (A.Array[i] <= A.Array[j])
                  {     B.Array[p] = A.Array[i];                 i++;}
                  else
                  {       B.Array[p] = A.Array[j];                 j++;          }
                      p = p + 1;
      }
      if (i <= b)
      {   int tg1,tg2; tg1=p;                     tg2=i;
          while((tg1<=c)&&(tg2<=b))
          {B.Array[tg1] = A.Array[tg2];                    tg1++; tg2++; }
      }
      else
      {   int tg1,tg2; tg1=p;                     tg2=j;
          while((tg1<=c)&&(tg2<=c))
          {B.Array[tg1] = A.Array[tg2];                    tg1++;      tg2++; }
      }
}
```

3.2.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

"MergeByLenght(KeyArray A, KeyArray B, int len)": Trộn lần lượt các cặp dãy con theo thứ tự.

```
void MergeByLenght(KeyArray A, KeyArray B,int len)
{    int a,b,c; a = 1; b =                len;     c  =  2  * len;
       while (c <= A.n-1)
   {      Merge(A, B, a, b,          c); a     =  a  +  2 * len;
          b = b + 2 * len;                c     =  c  +  2 * len;          }
             if (b < A.n - 1  )        Merge(A, B,      a, b, A.n - 1);
     else
               if (a <= A.n  - 1)
                  for (int i  = a; i < A.n - 1;              i++)
                           B.Array[i] = A.Array[i];
}
```

3.3.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

Thủ tục MergeSort(keyarray k):

```
 void MergrSort(KeyArray K)
{     KeyArray T; //     mảng phụ
      int       len; int      Flag; // Flag=1: trộn K vào T hoặc  ngược    lại
                Flag = 1;          len = 1;
      while (len < K.n-1)
      {       if      (Flag==1) MergeByLenght(K, T, len);
               else MergeByLenght(T, K, len);len = len *  2;        Flag = 0 ;
      }
    if (Flag!=1)                              K = T;
}
```

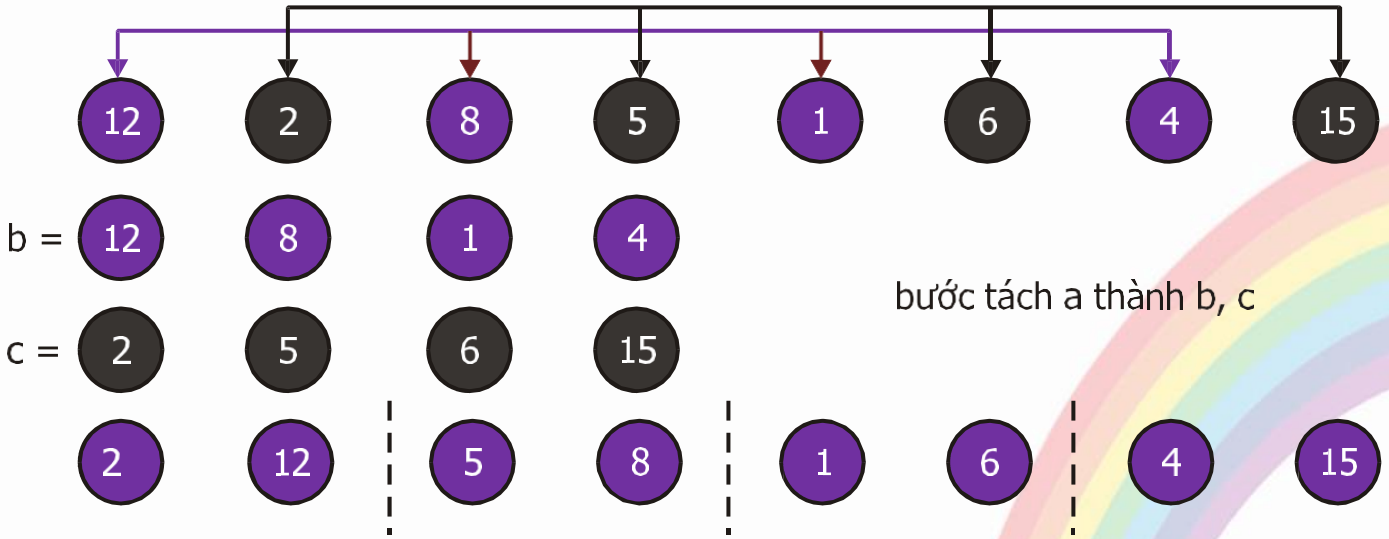
3.2.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

Ví dụ 7. 5.

Cho dãy số a: 12 2 8 5 1 6 4 15

Sắp xếp dãy trên theo thứ tự không giảm bằng thuật toán trộn trực tiếp được minh họa như sau:

k = 1: trộn



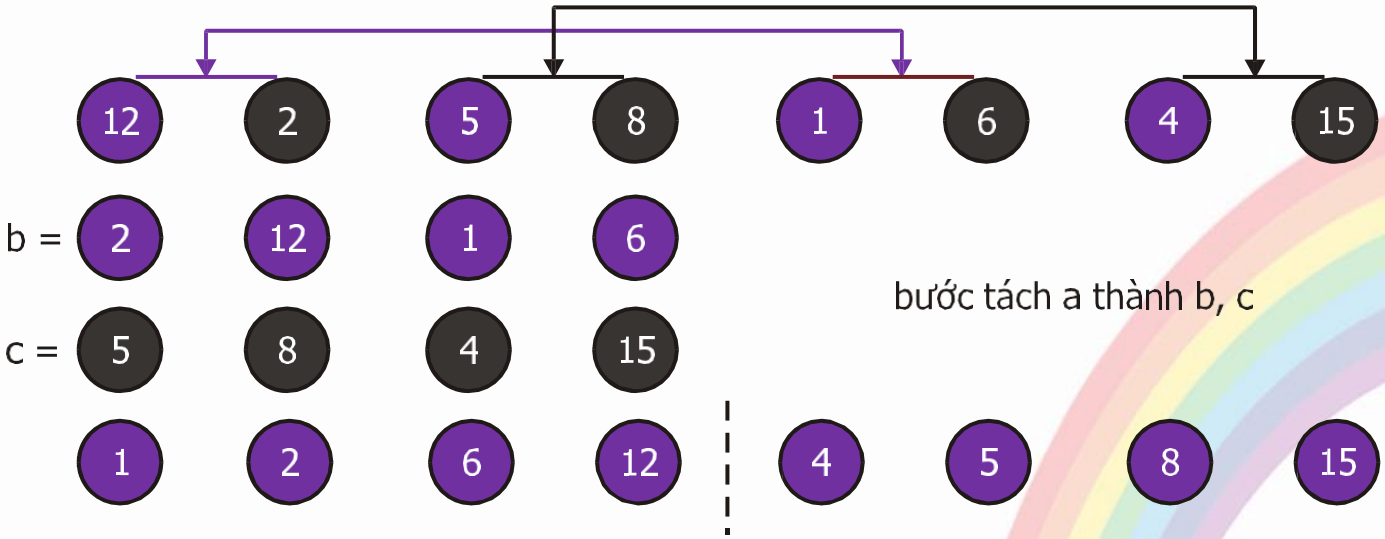
3.2.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

Ví dụ 7. 5.

Cho dãy số a: 12 2 8 5 1 6 15

Sắp xếp dãy trên theo thứ tự không giảm bằng thuật toán trộn trực tiếp được minh họa như sau:

k = 2:

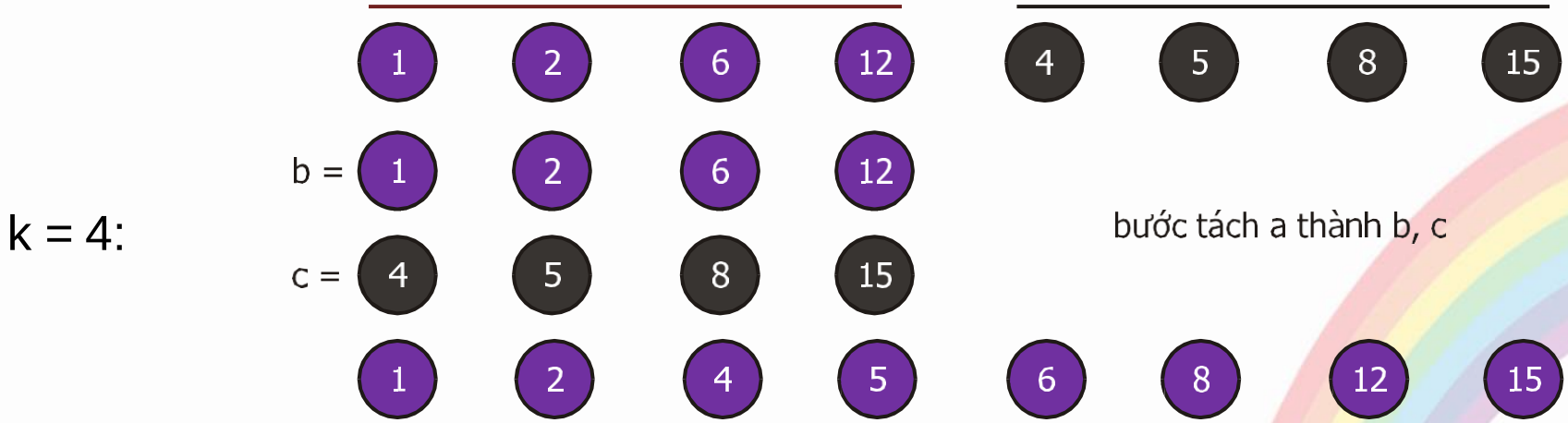


3.2.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

Ví dụ 7. 5.

Cho dãy số a: 12 2 8 5 1 6 15

Sắp xếp dãy trên theo thứ tự không giảm bằng thuật toán trộn trực tiếp được minh họa như sau:



3.2.1. THUẬT TOÁN SẮP XẾP TRỘN TRỰC TIẾP

Đánh giá thuật toán:

- Phép toán chủ đạo là thao tác đưa một phần tử vào miền sắp xếp;
- Độ phức tạp của thủ tục MergeByLenght là O(n);
- Thủ tục MergeSort có vòng lặp thực hiện không quá [lgn] lời gọi MergeByLenght;

- Độ phức tạp của MergeSort là O(nlgn) và trong mọi trường hợp độ
- phức tạp của thuật toán là không đổi;
- Nhược điểm của MergeSort là phải dùng thêm một vùng nhớ để chứa một danh sách phụ có chiều dài bằng danh sách cần sắp xếp ban đầu.

4.KHỐI (HEAP) VÀ HEAPSORT.

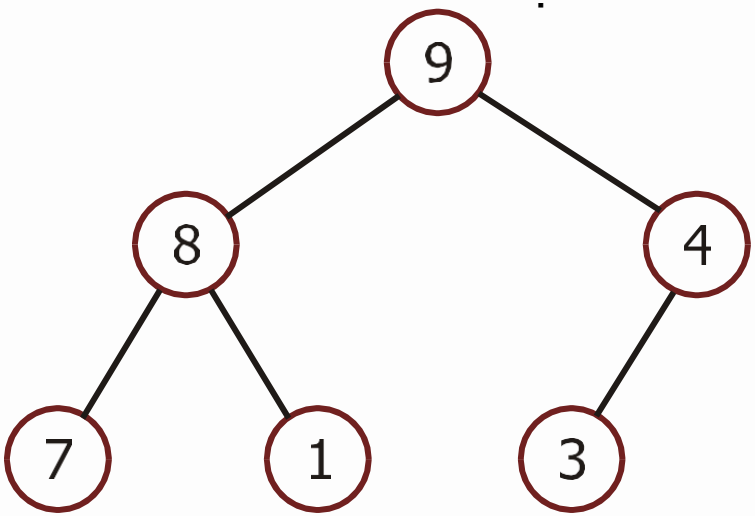
4.KHỐI (HEAP) VÀ HEAPSORT.

4.1. KHỐI (HEAP)

Khối là một loại cây nhị phân đặc biệt nhưng nó khác với các cây nhị phân tìm kiếm ở hai điểm:

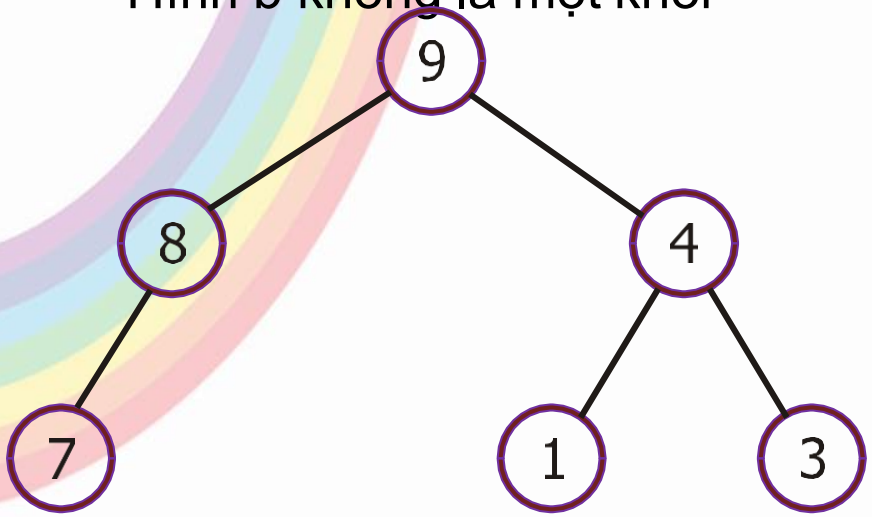
- Khối là **đầy đủ** nghĩa là các lá trên cây nằm nhiều nhất ở 2 mức liên tiếp nhau và các lá ở mức dưới nằm ở những vị trí ”bên trái nhất”;
- Giá trị của mục dữ liệu trong mỗi nút lớn hơn giá trị của những mục dữ liệu ở các con nó (nếu mục dữ liệu là bản ghi thì một trường nào đó trong bản ghi phải thỏa mãn điều kiện này).

Hình a là một khối



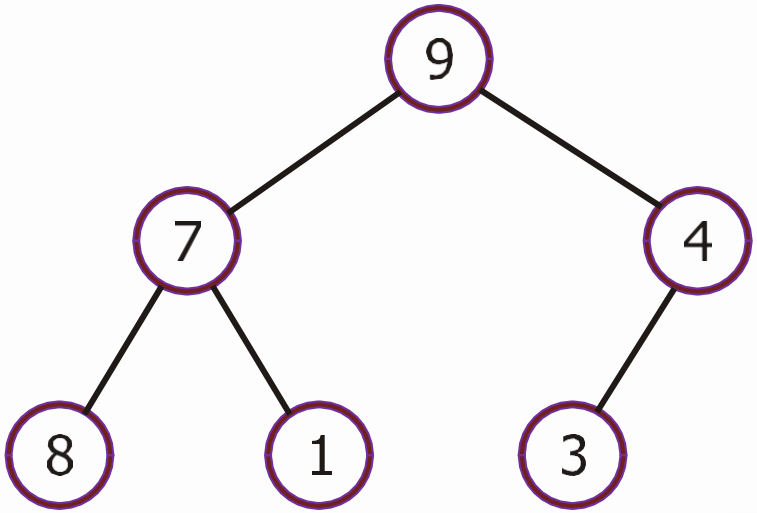
4.1. KHỐI (HEAP)

Hình b không là một khối



Hình b

Hình c không là một khối



Hình c

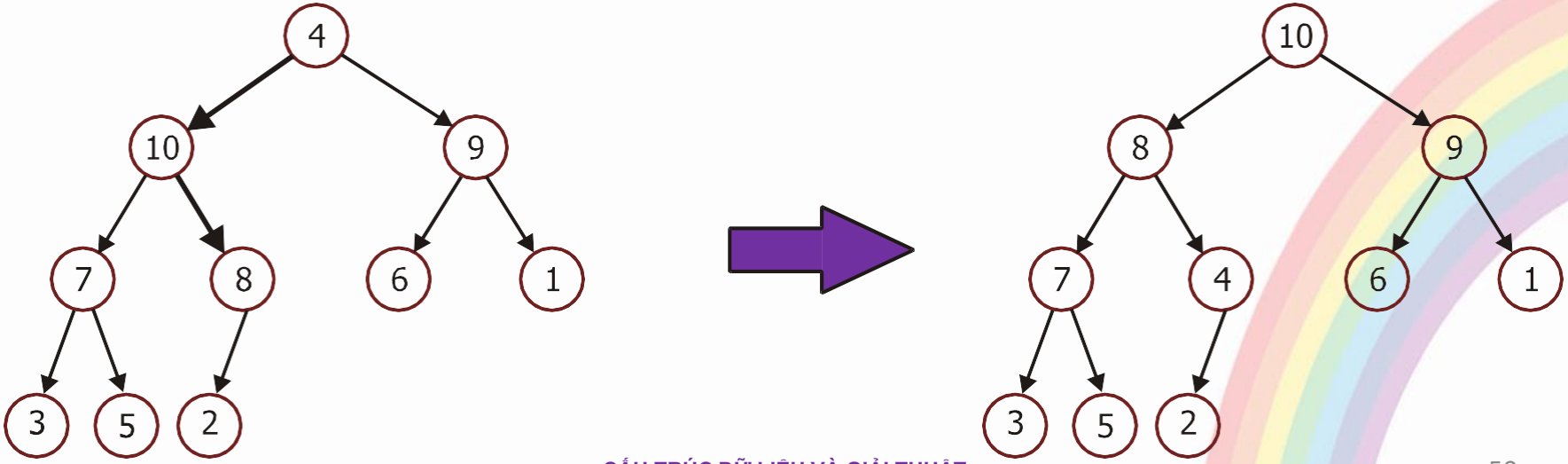
Cài đặt khối:

```
#define      HeapLimit …//giới hạn số các nút trong khối
typedef Kieu_du_lieu ElementType;
struct HeapType
    {          ElementType Array[HeapLimit];  int
               m;//số nút trong khối
   } Heap;
```

4.1. KHỐI (HEAP)

Cài đặt khối:

```
#define      HeapLimit …//giới hạn số các nút trong khối
typedef Kieu_du_lieu ElementType;
struct HeapType
    {          ElementType Array[HeapLimit];  int
               m;//số nút trong khối
    } Heap;
```



4.2. HEAPSORT

Tư tưởng:

- Danh sách A[1…n] được vun thành một khối;
- Phần tử đầu tiên A[1] tương ứng với nút gốc của khối là khóa lớn nhất;
- Đảo giá trị của phần tử này cho phần tử A[n] và không tính tới A[n] nữa;
- Còn lại danh sách A[1... n – 1] biểu diễn cây nhị phân hoàn chỉnh mà hai nhánh cây con nút thứ 2 và nút thứ 3 (hai nút cây con của nút 1) đã là khối rồi;
- Vậy chỉ cần vun lại một lần, ta lại được một khối, đảo giá trị A[1] cho A[n – 1] và tiếp tục cho tới khi khối chỉ còn lại một nút.

4.2. HEAPSORT

Nội dung của thuật toán sắp xếp HeapSort: 2 thủ tục chính:

Thủ tục Swapdown chuyển một cây nhị phân đầy đủ lưu tại các vị trí r...n của mảng A với các cây con bên trái và bên phải đã tạo thành một khối thành một khối.

Bước 1: Khởi tạo giá trị

Done = 0; C = 2 * r;

Bước 2:

```
while ((Done!=0)&&(C <= n-1))//tìm con lớn nhất
 {   if (C < n-1)
                 if (A[C] < A[C+1])          C = C + 1;
                 if (A[r]<A[C])
                 {Interchange(A[r],A[C]);
                  //hàm để hoán vị A[r]với A[C]
                     r = C;
                    C = C* 2;                      }
                  else                Done = 1;
   }
```

4.2. HEAPSORT

Thủ tục HeapiFy để chuyển một cây nhị phân đầy đủ lưu tại các vị trí từ 1...n của mảng A thành một khối.

Bước 1:

```
HeapiFy(A);//hàm chuyển cây thành khối
```

Bước 2:

```
for (int i= n-1;i>=1;i++)
    { Interchange(A[0],A[i]);
      //hàm hoán vị A[0] và A[i]
        SwapDown(A, 0);
     }
```

4.2. HEAPSORT

Cài đặt thuật toán HeapSort:

```
void Interchange(ElementType X,ElementType Y)
{    ElementType Temp;
     Temp = X;
     X = Y;
     Y = Temp;
}
```

4.2. HEAPSORT

Cài đặt thuật toán HeapSort:

```
void SwapDown(HeapType Heap,int r)
{    int Child,Done;
     Done = 0;              Child = 2 * r;
     while ((Done!=0)&&(Child <= Heap.m-1))
     { if (Child < Heap.m-1)
          if(Heap.Array[Child]<Heap.Array[Child+1])
          Child = Child + 1;
          if (Heap.Array[r]<Heap.Array[Child])
            {Interchange(Heap.Array[r],Heap.Array[Child]);
            r = Child;
                    Child = Child * 2;                          }
          else       Done = 1;

    }
}
```

4.2. HEAPSORT

Cài đặt thuật toán HeapSort:

```
void HeapiFy(HeapType Heap)
{    int r;
     for (r = Heap.m/2-1;r>=0;r--)
              SwapDown(Heap, r);
}
```

```
void HeapSort(HeapType A)
{    HeapiFy(A);
     for (int i= A.m-1;i>=1;i++)
     {     Interchange(A.Array[0],A.Array[i]);
           SwapDown(A, 0);
     }
}
```

4.2. HEAPSORT

Đánh giá thuật toán:

- Độ phức tạp của HeapSort phụ thuộc chủ yếu vào độ phức tạp của các thuật toán SwapDown và HeapiFy;
- Trong SwapDown, độ phức tạp của thuật toán này trong trường hợp tồi nhất là $O(\log_2 n)$;
- Trong thuật toán HeapiFy thực hiện SwapDown n/2 lần, độ phức tạp của nó trong trường hợp tồi nhất là $O(n\log_2 n)$;
- HeapSort chỉ thực hiện HeapiFy một lần và SwapDown n – 1 lần; từ đó suy ra độ phức tạp của HeapSort trong trường hợp tồi nhất là $O(n\log_2 n)$.

TÓM LƯỢC CUỐI BÀI

- Những thuật toán có độ phức tạp $O(n^2)$ như sắp xếp kiểu chọn, sắp xếp chèn hay sắp xếp đổi chỗ thì chỉ nên áp dụng cho các chương trình sắp xếp có ít lần sắp xếp và với kích thước n nhỏ;
- Trong đó về tốc độ, BubbleSort luôn đứng bét nhưng nó lại hết sức đơn giản cho người mới học lập trình;
- Thuật toán chèn tỏ ra nhanh hơn các thuật toán còn lại và cũng có tính ổn định, mã lệnh dễ nhớ;
- Thuật toán sắp xếp chia để trị như QuickSort, MergeSort hay thuật toán sắp xếp HeapSort là những thuật toán sắp xếp tổng quát;
- QuickSort gặp nhược điểm trong trường hợp suy biến (dù trường hợp này là rất nhỏ);
- MergeSort phải đòi hỏi thêm một không gian nhớ phụ;
- HeapSort thì mã lệnh hơi phức tạp và khó nhớ;
- Tuy nhiên những nhược điểm này là quá nhỏ so với ưu điểm chung của chúng là nhanh hơn nhiều so với các thuật toán khác.