CHƯƠNG V

CÁC THUẬT TOÁN SẮP XẾP NỘI

• ***Phát biểu bài toán***: Sắp xếp dãy a_1, a_2, ..., a_n theo thứ tự tăng dần hoặc giảm dần. Dãy sắp xếp có thể được lưu trữ bằng danh sách đặc hoặc danh sách liên kết.

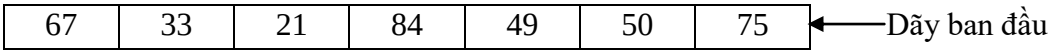
I. MỘT SỐ PHƯƠNG PHÁP ĐƠN GIẢN

1. Phương pháp chọn

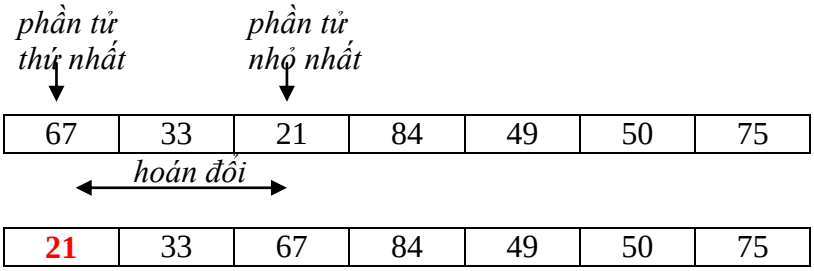
Ý tưởng của phương pháp chọn (*Selection Sort*) là tìm phần tử nhỏ nhất đặt vào vị trí thứ nhất a_1, phần tử nhỏ nhất thứ hai đặt vào vị trí thứ hai a_2, ...

◊ *Ví dụ*

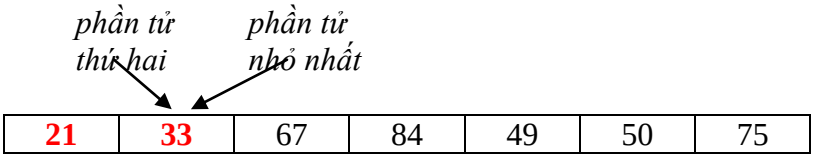
Giả sử ta phải sắp xếp thứ tự danh sách sau: 67, 33, 21, 84, 49, 50, 75. Thứ tự các bước tìm phần tử nhỏ nhất mô tả trong sơ đồ sau:



- *Bước* 1: Tìm phần tử nhỏ nhất, sau đó hoán đổi với phần tử thứ nhất

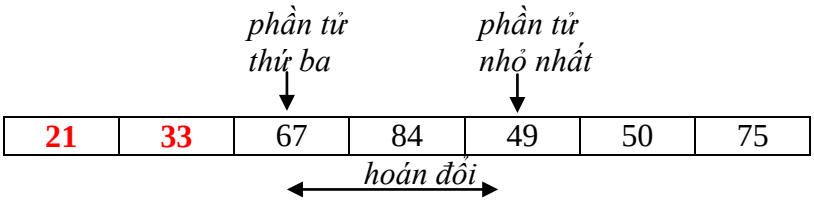


- *Bước* 2: Tìm phần tử nhỏ nhất trong dãy con còn lại (trừ phần tử thứ nhất), sau đó hoán đổi với phần tử thứ 2



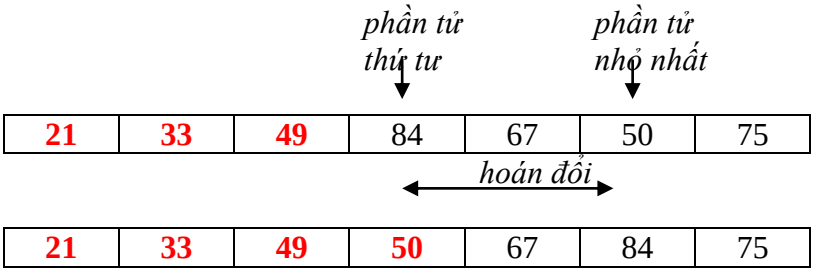
Ghi chú: ở đây phần tử thứ 2 và phần tử nhỏ nhất trùng nhau nên không cần hoán đổi.

- *Bước* 3: Tìm phần tử nhỏ nhất trong dãy con còn lại, sau đó hoán đổi với phần tử thứ 3

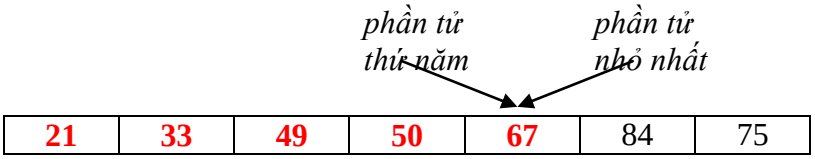


21	33	49	84	67	50	75

- *Bước 4:* Tìm phần tử nhỏ nhất trong dãy con còn lại, sau đó hoán đổi với phần tử thứ 4.

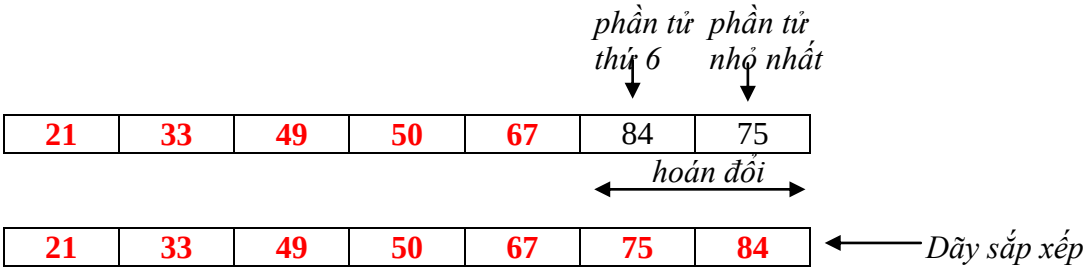


- *Bước 5:* Tìm phần tử nhỏ nhất trong dãy con còn lại, sau đó hoán đổi với phần tử thứ 5.



Ghi chú: Ở đây phần tử thứ 5 và phần tử nhỏ nhất trùng nhau nên không cần hoán đổi.

- *Bước 6:* Tìm phần tử nhỏ nhất trong dãy con còn lại, sau đó hoán đổi với phần tử thứ 6.



a. Thủ tục cài đặt thuật toán bằng danh sách đặc

Trong PASCAL

```
const    n = <số phần tử cực đại>;
type  ItemType = <kiểu dữ liệu sắp xếp>;
var a : array[1..n] of ItemType;
Procedure  SelectionSort;
var i, j, k : integer;
begin
    for i:=1 to n-1 do
        begin
        {chọn a[k] nhỏ nhất trong a[i],
            a[i+1], ..., a[n]}
            k := i;
```

Trong C

```
#define n  <số phần tử cực đại>
typedef  <kiểu dữ liệu sắp xếp> ItemType ;
ItemType  a[n+1];
void  SelectionSort()
{
    int  i, j, k;
    for (i=1; i <= n-1; i++)
    {
    /*chọn a[k] nhỏ nhất trong a[i],
        a[i+1], ..., a[n]*/
        k = i;
```

```
            for j := i + 1 to n do
                if a[j] < a[k] then
                    k := j;
            {đổi chỗ a[k] và a[i]}
            if  k<>i then
                swap(a[i],a[k]);
        end;
end;
Procedure  swap(var x, y:ItemType);
var  temp : ItemType;
begin
    temp := x;
    x := y;
    y := temp;
end;
```

```
        for ( j = i + 1; j <= n; j++)
        if (a[j] < a[k])
            k = j;
        //đổi chỗ a[k] và a[i]
        if  (k!=i)
            swap(&a[i], &a[k]);
    }
}
void  swap(ItemType *x,*y)
{
    ItemType   temp ;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

b. Thủ tục cài đặt thuật toán bằng danh sách liên kết

Trong PASCAL

```
Type ItemType = <kiểu dữ liệu>;
    pointer  = ^element;
    element = record
                item : ItemType;
                next : pointer;
              end;
Var  First;   {con trỏ đầu danh sách}
Procedure  SelectionSort;
var p,    {con trỏ chạy}
    smallPtr,     {con trỏ nhỏ}
    q: pointer; {con trỏ chạy}
begin
    p := first;
    while  p <> nil  do
      begin
    {tìm nút chứa giá trị item  nhỏ nhất}
        smallPtr := p;
        q := p^.next;
        while q <> nil  do
          begin
            if  q^.item < smallPtr^.item
            then
                smallPtr := q;
            q := q^.next;
          end
        {chuyển phần tử nhỏ nhất
        smallPtr^.item vào nút p}
```

Trong C

```
typedef   <kiểu dữ liệu > ItemType;
struct  Node {
                ItemType item;
                struct Node *next;
              };
typedef struct Node *NodePointer;
NodePointer first;
void   SelectionSort()
{
    NodePointer  p, //con trỏ chạy
                 smallPtr,     //con trỏ nhỏ
                 q ;  //con trỏ chạy
    p = first;
    while  (p != NULL)
    {
    //tìm nút chứa giá trị item  nhỏ nhất
        smallPtr = p;
        q = p->next;
        while (q != NULL)
          {
            if  (q->item < smallPtr->item)
                smallPtr = q;
            q = q->next;
          }
            /*chuyển phần tử nhỏ nhất
            smallPtr^.item vào nút p*/
        swap(&p->item,&smallPtr->item);
```

swap(p^.item, smallPtr^.item); *end;* *end;*	} }

• *Phân tích độ phức tạp:*

- Số lần so sánh : $C = n*(n-1)/2$

- Số lần gán ($k=i$): $n-1$

- Số lần đổi chỗ : $M_{max} = 3*(n-1)$

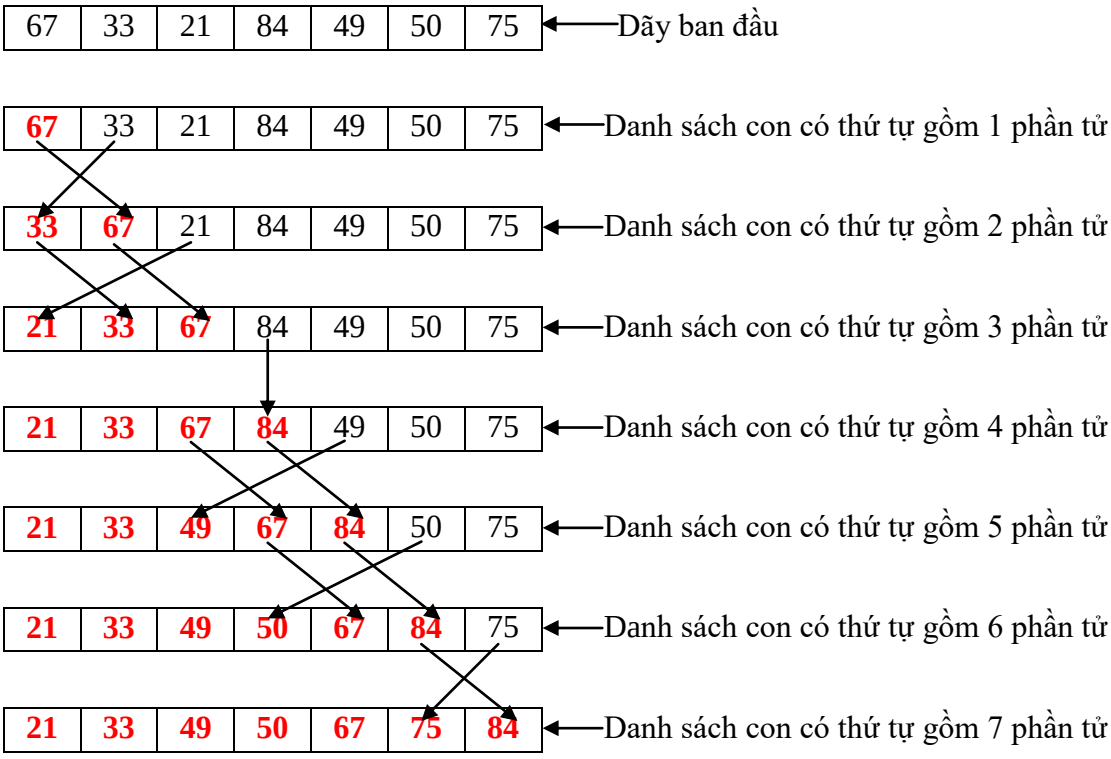
(*Ghi chú:* $M_{min} = 0$, $M_{ave} = n*(ln(n) + g)$, trong đó $g = 0.577216$ là hằng số Euler).

Vậy độ phức tạp là $n*(n-1)/2 + n-1 + 3*(n-1) = O(n^2)$.

2. Phương pháp chèn

Ý tưởng của phương pháp chèn (*Insertion Sort*) như sau: giả sử đã có a_1, a_2, ..., a_{i-1} đã có thứ tự, ta phải chèn a_i vào vị trí thích hợp, bằng cách tìm tuần tự từ a_{i-1} đến a_1, sao cho dãy $a_1, a_2, ..., a_i$ có thứ tự.

◊ *Ví dụ.* Giả sử ta phải sắp xếp thứ tự danh sách sau: 67, 33, 21, 84, 49, 50, 75. Thứ tự chèn mô tả trong sơ đồ sau:



• *Thủ tục cài đặt thuật toán bằng danh sách đặc.*

Trong thủ tục này mảng a có $n+1$ phần tử từ 0 đến n.

Trong PASCAL | Trong C

```
Procedure  InsertionSort;
var x : ItemType;
    i, j : integer;
begin
    for i := 2 to n do
        begin
            x := a[i];
            a[0] := x; {phần tử cầm canh}
            j := i – 1;
            while  x < a[j] do
                begin
                    a[j+1] := a[j];
                    j := j – 1; {giảm j}
                end
            a[j+1] := x
        end;
end;
```

```
void  InsertionSort()
{
    ItemType x;
    int  i, j ;
    for (i = 2; i <= n; i++)
      {
        x = a[i];
        a[0] = x; //phần tử cầm canh
        j = i – 1;
        while (x < a[j])
            {
                a[j+1] = a[j];
                j = j – 1; //giảm j
            }
        a[j+1] = x;
      }
}
```

- *Phân tích độ phức tạp:*

Số lần so sánh:

$C_{min} = n - 1$

$C_{ave} = (n^2 + n - 2) / 4$

$C_{max} = (n^2 + n - 2) / 2$

Số lần di chuyển:

$M_{min} = 0$

$M_{ave} = (n^2 + 9*n - 10) / 4$

$M_{max} = (n^2 + 3*n - 4) / 2$

Như vậy độ phức tạp là $O(n^2)$.

3. Phương pháp chèn nhị phân

Phương pháp chèn nhị phân (*Binary Insertion*) giống phương pháp chèn trực tiếp, nhưng dùng cách tìm nhị phân để chèn a_i vào dãy thứ tự $a_1, a_2, \ldots, a_{i-1}$.

- *Thủ tục cài đặt giải thuật bằng danh sách đặc*

Trong PASCAL | Trong C

```
Procedure  BinaryInsertion;
var x : ItemType;
    i, j, q, r, m : integer;
begin
    for i := 2 to n do
      begin
        x := a[i];
        {tìm nhị phân}
        q := 1;
        r := i - 1;
        while q <= r do
          begin
            m := (q + r) div 2;
            if x < a[m] then
                r := m - 1
            else
                q := m + 1
          end
        for j := i - 1 downto q do
            a[j+1] := a[j];
        a[q] := x;
      end;
end;
```

```
void  BinaryInsertion()
{
    ItemType x;
    int  i, j, q, r, m ;
    for (i = 2; i <= n; i++)
      {
        x = a[i];
        //tìm nhị phân
        q = 1;
        r = i – 1;
        while (q <= r)
          {
            m = (q + r) / 2;
            if (x < a[m])
                r = m - 1;
            else
                q = m + 1;
          }
        for (j = i – 1; j >= q; j--)
            a[j+1] = a[j];
        a[q] = x;
      }
}
```

• *Phân tích độ phức tạp:*

Số lần so sánh: $C = n*(log_2(n) – log_2(e) \pm 0.5)$. Số lần di chuyển giống phương pháp chèn trực tiếp. Như vậy độ phức tạp là $O(n^2)$.

Phương pháp chèn không thích hợp lắm với danh sách đặc, vì khi chèn một phần tử ta phải di chuyển các phần tử sau đi một vị trí. Phương pháp chèn thích hợp với danh sách liên kết.

4. Phương pháp nổi bọt

Ý tưởng phương pháp nổi bọt (*BubleSort*) như sau: duyệt dãy a_1, a_2, ... , a_n từ đáy lên đỉnh, nếu $a_i > a_{i+1}$ thì hoán chuyển chúng. Lặp lại quá trình này cho đến khi không hoán chuyển được nữa. Nếu hình dung dãy cần sắp xếp được đặt thẳng đứng thì sau từng đợt duyệt các giá trị nhỏ sẽ "nổi" dần lên giống như bọt nước.

◊ *Ví dụ*

Giả sử ta phải sắp xếp thứ tự danh sách sau: 67, 33, 21, 84, 49, 50, 75. Thứ tự các bước tìm phần tử nhỏ nhất mô tả trong sơ đồ sau:

i	a_i	duyệt từ a_7 đến a_2	duyệt từ a_7 đến a_3	duyệt từ a_7 đến a_4	duyệt từ a_7 đến a_5	duyệt từ a_7 đến a_6
1	67	**21**	**21**	**21**	**21**	**21**
2	33	67	**33**	**33**	**33**	**33**
3	21	33	67	**49**	**49**	**49**
4	84	49	49	67	**50**	**50**
5	49	84	50	50	67	**67**
6	50	50	84	75	75	**75**
7	75	75	75	84	84	**84**

◊ *Ghi chú*: ở lần duyệt cuối cùng, vì không có sự hoán đổi nào cả nên ta đã có dãy sắp xếp mà không cần duyệt tiếp.

• *Thủ tục cài đặt giải thuật bằng danh sách đặc*

Trong PASCAL

```
Procedure BubleSort;
var i, j : integer;
begin
    for i := 2 to n do
      for j:= n downto i do
        if a[j-1] > a[j] then {hoán chuyển}
            swap(a[j-1], a[j]);
end;
```

Trong C

```
void BubleSort()
{
    int  i, j ;
    for (i = 2; i <= n; i++)
      for ( j = n; j >= i; j--)
        if  (a[j-1] > a[j]) //hoán chuyển
            swap(&a[j-1], &a[j]);
}
```

• *Cải tiến:* Nếu ở lần duyệt dãy theo *j := n downto i* mà không có hoán chuyển thì dãy *a*[1], *a*[2], ..., *a*[*n*] đã có thứ tự và giải thuật kết thúc. Ta dùng cờ hiệu *flag* để ghi nhận điều này.

Trong PASCAL

```
Procedure FlagBubleSort;
var i, j : integer;
flag : boolean;
begin
    flag := true; i:=2;
    while flag and (i<=n) do
        begin
            flag := false;
            for  j := n downto i  do
                if a[j-1] > a[j] then {hoán
                chuyển}
                    begin
                        swap(a[j], a[j-1]);
                        flag := true
```

Trong C

```
void FlagBubleSort()
{
    int i, j ;
    int flag;
    flag = 1; i =2;
    while (flag && (i<=n))
    {
      flag = 0;
      for  (j = n; j >= i; j--)
        if (a[j-1] > a[j]) //hoán chuyển
            {
                swap(&a[j], &a[j-1]);
                flag = 1;
            }
```

```
                    end;
                i := i+1;
            end;
end;
```

```
        i = i+1;
    }
}
```

- *Phân tích độ phức tạp:*

Số lần so sánh: $C = n*(n-1)/2$

Số lần đổi chỗ: $M_{min} = 0$

$M_{ave} = 3*n*(n-1)/4$

$M_{max} = 3*n*(n-1)/2$

Như vậy độ phức tạp là: $n*(n-1)/2 + 3*n*(n-1)/2 = O(n^2)$.

5. Phương pháp ShakerSort

Phương pháp này cải tiến phương pháp nổi bọt. Tại mỗi lần duyệt, ta nhớ vị trí đổi chỗ lần đầu r và lần cuối k. Khi đó dãy $a[1], a[2], ..., a[k]$ và dãy $a[r+1], ..., a[n]$ đã có thứ tự và ở lần tiếp theo ta sẽ chỉ cần duyệt từ phần tử thứ $k+1$ đến phần tử thứ $r-1$.

- *Thủ tục cài đặt giải thuật:*

Trong PASCAL

```
Procedure ShakerSort;
var i, k, q, r : integer;
begin
    q := 2; r := n; k := 1;
    repeat
      for j := r downto q do
        if a[j-1] > a[j] then {hoán chuyển}
            begin
                swap(a[j-1], a[j]);
                k := j
            end;
      q := k + 1; {left}
      for j := q to r do
        if a[j-1] > a[j] then {hoán chuyển}
            begin
                swap(a[j-1], a[j]);
                k := j
            end;
      r := k - 1; {right}
    until q > r;
end;
```

Trong C

```
void ShakerSort()
{
    int  i, k, q, r ;
    q = 2; r = n; k = 1;
    do {
      for (j = r; j >= q; j--)
        if (a[j-1] > a[j]) //hoán chuyển
            {
                swap(&a[j-1], &a[j]);
                k = j;
            }
      q = k + 1; //left
      for ( j = q; j <= r; j++)
        if ( a[j-1] > a[j]) //hoán chuyển
            {
                swap(&a[j-1], &a[j]);
                k = j;
            }
      r = k - 1; //right
    } while  (q <= r);
}
```

• *Phân tích độ phức tạp:*

Số lần so sánh: $C_{min} = n - 1$

$C_{ave} = n*(n - ln(n) - (g + ln(2) -1))/2$

$C_{max} = n*(n - 1)/2$

Số lần đổi chỗ: $M_{min} = 0$

$M_{ave} = 3*n*(n - ln(n) - (g + ln(2) -1)) / 4$

$M_{max} = 3*n*(n - ln(n) - (g + ln(2) -1)) / 2$

với g = 0.577216 là hằng số Euler.

Như vậy độ phức tạp là $O(n^2)$.

6. Phương pháp đếm so sánh

Ý tưởng của phương pháp đếm so sánh (Comparison Counting Sort) là đếm số phần tử nhỏ hơn hay bằng *a*[*i*]. Nếu có *j* phần tử nhỏ hơn *a*[*i*] thì *a*[*i*] có vị trí thứ (*j*+1) trong dãy có thứ tự. Các phần tử *a*[*i*] được sắp thứ tự vào dãy có thứ tự s[*i*].

Trong giải thuật ta dùng mảng *count*[*i*] (*i* = 1,2,...,*n*) xác định số phần tử nhỏ hơn *a*[*i*]. Như vậy *count*[*i*]+1 là vị trí của phần tử *a*[*i*] trong dãy có thứ tự.

• *Thủ tục cài đặt giải thuật:*

Trong PASCAL

```
Procedure Comparison_Counting_Sort;
var i, j : integer;
    s : array[1..n] of ItemType;
    count : array[1..n] of integer;
begin
    for i := 1 to n do
        count[i] := 0;
    for i := n downto 1 do
        for j := i - 1  downto 1 do
            if a[i] < a[j] then
                count[j] := count[j] + 1
            else
                count[i] := count[i] + 1;
    {di chuyển các  phần tử a[i]}
    for i := 1 to n do
        s[count[i]+1] := a[i];
    for i := 1 to n do
        a[i] := s[i];
end;
```

Trong C

```
void Comparison_Counting_Sort()
{
    int  i, j ;
    ItemType  s[n+1];
    int count[n+1];
    for (i = 1; i <= n; i++)
        count[i] = 0;
    for (i = n; i >= 1; i--)
        for ( j = i – 1; j >= 1; j--)
            if (a[i] < a[j])
                count[j]++;
            else
                count[i]++;
    //di chuyển các  phần tử a[i]
    for (i = 1; i <= n; i++)
        s[count[i]+1] = a[i];
    for (i = 1; i <= n; i++)
        a[i] = s[i];
}
```

• *Phân tích độ phức tạp:*

Số lần so sánh: $C = n*(n - 1) / 2$

Số lần đổi chỗ: $M = n$

Độ phức tạp : $O(n^2)$.

7. Phương pháp đếm phân phối

Điều kiện để áp dụng phương pháp đếm phân phối (*Distribution Counting Sort*) là các phần tử *a*[*i*] phải nguyên và nằm trong khoảng [*u*,*v*].

Ý tưởng của phương pháp này là sắp thứ tự các phần tử *a*[*i*] vào dãy có thứ tự *s*[*i*]. Mảng *count* có phần tử *count*[*j*] đếm số phần tử của mảng *a* có giá trị nhỏ hơn hoặc bằng *j*.

• *Thủ tục cài đặt giải thuật:*

Trong PASCAL

```
Procedure Distribution_Counting_Sort;
var i, j : integer;
    s : array[1 .. n] of ItemType;
    count : array[1 .. n] of integer;
begin
    for i := u to v do
        count[i] := 0;
    for j := 1 to n do
        count[a[j]] := count[a[j]] + 1;
    for i := u+1 to v do
        count[i] := count[i] + count[i - 1];
    {Các phần tử trong dãy a có giá trị
    bằng i sẽ chiếm vị trí từ count[i-1]+1
    đến count[i] }
    for j := n downto 1 do
        begin
            i := count[a[j]];
            s[i] := a[j];
            count[a[j]] := i - 1;
        end
end;
```

Trong C

```
void Distribution_Counting_Sort()
{
    int i, j ;
    ItemType s[n+1] ;
    int count[n+1] ;
    for (i = u; i <= v; i++)
        count[i] = 0;
    for (j = 1; j <= n; j++)
        count[a[j]] = count[a[j]] + 1;
    for (i = u+1; i <= v; i++)
        count[i] = count[i]+count[i-1];
    /*Các phần tử trong dãy a có giá trị
    bằng i sẽ chiếm vị trí từ count[i-1]
    +1 đến count[i] */
    for (j = n; j >= 1; j--)
        {
            i = count[a[j]];
            s[i] = a[j];
            count[a[j]] = i - 1;
        }
}
```

• *Phân tích độ phức tạp:*

Số lần tính : $C = O(v - u)$

Số lần đổi chỗ : $M = n$

Độ phức tạp : $O(v - u + n)$

II. PHƯƠNG PHÁP SẮP XẾP NHANH

Phương pháp sắp xếp nhanh (*QuickSort*) là thuật toán hiệu quả được sử dụng rộng rãi do A.R.Hoare phát minh năm 1960.

Trong các phương pháp sắp xếp trước đây, ý tưởng cơ bản là chọn phần tử nhỏ nhất (hay lớn nhất) trong một danh sách con nào đó của danh sách cần sắp xếp và đặt nó vào đúng vị trí trong danh sách con. Phương pháp sắp xếp nhanh (*QuichSort*) cũng chọn một mục và định vị đúng cho mục đó. Tuy nhiên mục chọn không nhất thiết phải là phần tử nhỏ nhất hay lớn nhất, mà là mục nào đó nằm giữa danh sách con.

Phần tử được chọn được định vị đúng bằng cách sắp xếp lại danh sách hay danh sách con sao cho tất cả phần tử bên trái phần tử chọn nhỏ hơn hoặc bằng nó, và tất cả phần tử bên phải phần tử chọn lớn hơn hoặc bằng nó. Như vậy danh sách (con) được chia thành những danh sách con nhỏ hơn, và cũng bằng cách đó mỗi danh sách con được sắp xếp độc lập. Phương pháp "chia để trị" này, một cách tự nhiên, dẫn đến một thuật toán đệ quy.

◊ *Ví dụ:* Để minh họa ta xét danh sách sau:

75	70	65	84	98	78	100	93	55	61	81	68

Để đơn giản ta chọn số đầu tiên là 75 và xác định vị trí đúng của nó trong danh sách được sắp xếp. Ta phải sắp xếp lại danh sách sao cho 70, 65, 55, 61 và 68 ở bên trái của **75** (nhưng không nhất thiết theo thứ tự ở đây) và các số còn lại ở bên phải **75.**

Ta thực hiện hai lần tìm kiếm:

- Tìm phần tử nhỏ hơn hay bằng 75 bắt đầu từ phần tử ngoài cùng bên phải.
  Trong ví dụ này là phần tử 68 (có dấu mũi tên ↓)
- Tìm phần tử lớn hơn 75 bắt đầu từ phần tử ngoài cùng bên trái.
  Trong ví dụ này là phần tử 84 (có dấu mũi tên ↑)

pt chọn ↓

75	70	65	84	98	78	100	93	55	61	81	68

↑

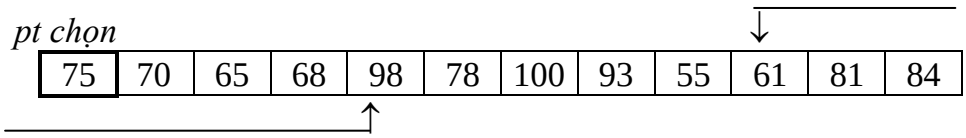
Hoán vị hai phần tử này ta có dãy

pt chọn ↓

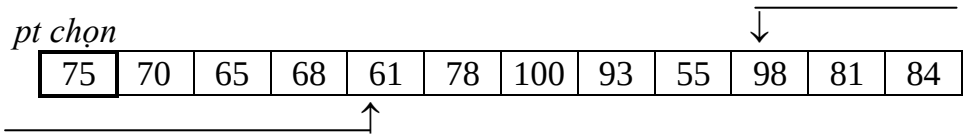
75	70	65	68	98	78	100	93	55	61	81	84

↑

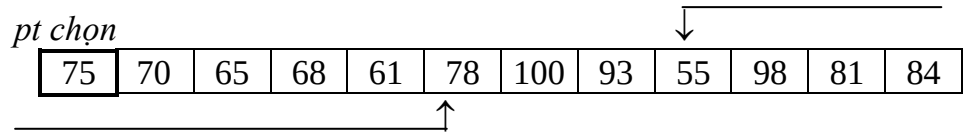
Lặp lại quá trình trên ta chọn được hai phần tử tiếp theo là 61 và 98.



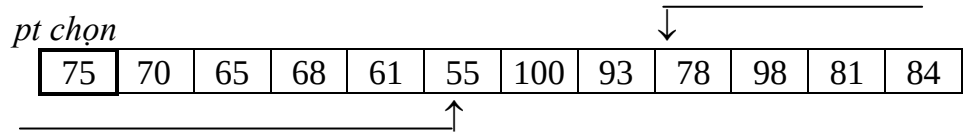
Hoán vị hai phần tử này ta có dãy



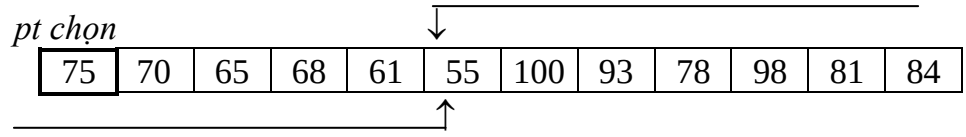
Lặp lại quá trình trên ta chọn được hai phần tử tiếp theo là 55 và 78.



Hoán vị hai phần tử này ta có dãy



Bây giờ, khi ta tiếp tục bắt đầu tìm từ bên phải, ta định vị được phần tử 55 đã được tìm ra trước đó từ bên trái.



Hai "con trỏ" (↓,↑) cho các lần tìm kiếm từ phải và từ trái gặp nhau. Điều này báo hiệu kết thúc tìm kiếm. Bây giờ ta hoán vị 55 và phần tử chọn 75 và được dãy:

55	70	65	68	61	75	100	93	78	98	81	84

Ta thấy rằng các phần tử bên trái 75 đều nhỏ hơn nó, và các phần tử bên phải 75 đều lớn hơn nó. Như vậy phần tử 75 đã xếp đúng vị trí của nó.

Danh sách con bên trái

55	70	65	68	61

và danh sách con bên phải

100	93	78	98	81	84

có thể được sắp xếp một cách độc lập, bằng bất cứ thuật toán nào.

Thủ tục *Split* dưới đây sẽ thực hiện quá trình phân tách trên.

◊ Trong PASCAL:

```
Procedure  Split(low, high : integer; var pos: integer);
{thủ tục này sắp lại các phần tử a[low], ..., a[high]  sao cho  phần tử  a[low]
được định vị đúng vị trí của nó là  pos}
var
    left,               {chỉ phần tử tìm từ bên trái}
    right: integer;     {chỉ phần tử tìm từ bên phải}
    item : ItemType;{mục đang được định vị}
begin {split}
    item := a[low];
    left := low;
    right := high;
    while  left < right  do
        begin {khi hai con trỏ chưa gặp nhau}
            {tìm từ bên phải các phần tử <= item}
            while  a[right] > item do
                right := right - 1;
            {tìm từ bên trái các phần tử > item}
            while  (left < right) and (a[left] <= item) do
                left := left + 1;
            {hoán vị hai phần tử tìm thấy nêu các con trỏ chưa gặp nhau}
            if  left < right  then
                swap(a[left], a[right]);
        end; {left < right}
    {tìm xong, đặt mục item được chọn vào vị trí đúng}
    pos := right;
    swap(a[low], a[pos]);
end;
```

◊ Trong C:

```
void  Split(int low, high; int  *pos)
{
    /*thủ tục này sắp lại các phần tử a[low], ..., a[high]  sao cho  phần tử  a[low]
được định vị đúng vị trí của nó là  pos */
    int left,       //chỉ phần tử tìm từ bên trái
        right;    // chỉ phần tử tìm từ bên phải
    ItemType  item ;       // mục đang được định vị
    item = a[low];
    left = low;
    right = high;
    while (left < right)
        {    //khi hai con trỏ chưa gặp nhau
            //tìm từ bên phải các phần tử <= item
```

```
            while  ((left < right) &&(a[right] > item))
                right = right - 1;
            //tìm từ bên trái các phần tử > item
            while  ((left < right) && (a[left] <= item))
                left = left + 1;
            //hoán vị hai phần tử tìm thấy nêu các con trỏ chưa gặp nhau
            if  (left < right)
                swap(&a[left], &a[right]);
        }
    //tìm xong, đặt mục item được chọn vào vị trí đúng
    *pos = right;
    swap(&a[low], &a[*pos]);
}
```

Bây giờ ta có thể viết thủ tục đệ quy sắp xếp danh sách:

Trong PASCAL	Trong C
Procedure QuickSort(low, high : integer); *var pos : integer;* *begin* *if (low < high) then* *begin* *{chia thành hai danh sách con}* *Split(low, high, pos);* *{sắp xếp danh sách con bên trái}* *QuickSort(low, pos – 1);* *{sắp xếp danh sách con bên phải}* *QuickSort(pos + 1, high);* *end;* *end;*	*void QuickSort(int low, high)* { *int pos ;* *if (low < high)* { *//chia thành hai danh sách con* *Split(low, high, &pos);* *//sắp xếp danh sách con bên trái* *QuickSort(low, pos – 1);* *//sắp xếp danh sách con bên phải* *QuickSort(pos + 1, high);* } }

Để sắp xếp dãy $a[1], ..., a[n]$, thủ tục này được gọi bởi lệnh có dạng:

$$QuickSort(1, n)$$

◊ *Chú thích:*

Để đánh giá độ phức tạp ta có:

Số lần so sánh: $C = n*log_2(n)$

Số lần đổi chỗ: $M = n*log_2(n) / 6$

Suy ra độ phức tạp trung bình là $O(n*log_2(n))$.

Đây là phương pháp tương đối tốt. Trong thủ tục trên x là phần tử chốt, càng gần trung vị của dãy càng tốt. Có nhiều đề cách chọn x khác làm tăng tốc độ tính toán. Ta

có thể chọn phần tử chốt x là trung vị của $a[q]$, $a[(q+r)\ div\ 2]$ và $a[r]$ (theo đề nghị của R.C.Singleton).

Mặt khác khi kích thước của các phân đoạn quá nhỏ, việc tiếp tục thực hiện *QuickSort* không có lợi. Lúc đó nên dùng một phương pháp sẵp xếp đơn giản nào đó. Theo Knuth (1974) khi phân đoạn còn lại 9 phần tử thì nên ngưng phương pháp *QuickSort* và sử dụng phương pháp đơn giản khác.

III. PHƯƠNG PHÁP VUN ĐỐNG

1. Định nghĩa Heap

Dãy $a[1], a[2], ..., a[n]$ là một *Heap* (đống) nếu:

$$a[k] \le a[j] \quad \forall (k, j) : \; k = [j/2] \;\; \& \;\; 1 \le k < j \le n$$

(ký hiệu $[x]$ là số nguyên lớn nhất nhỏ hơn hoặc bằng x).

Sau đây là một số tính chất của *Heap*.

(1) Nếu dãy $a[1], a[2], ..., a[n]$ có thứ tự thì nó là *Heap* (điều ngược lại không đúng).

(2) Nếu dãy $a[1], a[2], ..., a[n]$ là *Heap* thì $a[1]$ là nhỏ nhất.

(3) Đối với dãy $a[1], a[2], ..., a[n]$ thì dãy $a[j], a[j+1], ..., a[n]$ với $j = [n/2] + 1$ được xem là *Heap*.

Thật vậy, ta có

$$j \le i \le n \;\Rightarrow 2*i > n \;\; \& \; 2*i + 1 > n$$

nên không tồn tại phần tử *a[2*i]* và *a[2*i + 1]*.

(4) Một Heap $a[1], a[2], ..., a[n]$ có thể biểu diễn như một cây nhị phân:

$a[1]$ là nút gốc.

$a[1]$ là có 2 nút con $a[2]$ và $a[3]$.

$a[1] \le a[2]$ & $a[1] \le a[3]$

$a[2]$ là có 2 nút con $a[4]$ và $a[5]$.

$a[2] \le a[4]$ & $a[2] \le a[5]$

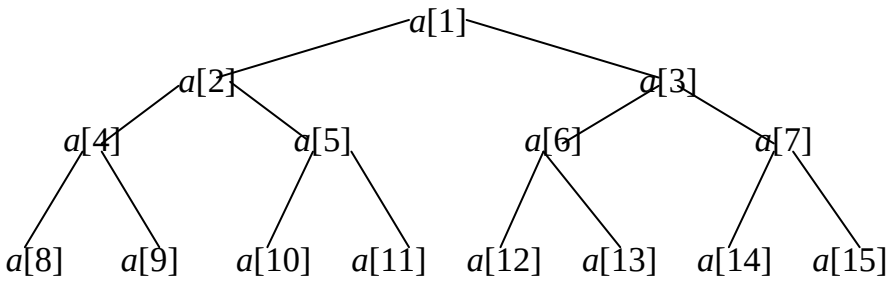
.....

Tổng quát

$a[j]$ là có 2 nút con $a[2*j]$ và $a[2*j+1]$.

$a[j] \le a[2*j]$ & $a[j] \le a[2*j+1]$

◊ *Ví dụ:* Với $n = 15$ ta có cây



Trong phương pháp vun đống (*HeapSort*), trước hết từ dãy ban đầu ta tạo một *Heap*, sau đó sắp xếp dãy đã được tạo thành *Heap*.

2. Tạo Heap ban đầu

Trước hết ta xét thủ tục *Sift* tạo *heap* con, sau đó cài đặt thuật toán tạo *heap* ban đầu.

(*i*) *Thủ tục Sift:* Tạo heap *a*[*q*], *a*[*q*+1], ... , *a*[*r*] nếu đã có heap *a*[*q*+1], *a*[*q*+2], ... , *a*[*r*].

Trong PASCAL

```
Procedure Sift(q, r: index);
var i, j : integer; x : ItemType; cont :
boolean;
begin
     i := q;
     j := 2 * i;
     x := a[i];
     cont := true;
     while (j <= r) and cont do
          begin
               if j < r  then
               {tìm nút nhỏ hơn trong 2 nút
               con của nút i là  j  và  j+1}
                    if  a[j] > a[j+1]  then
                         j := j + 1;
               if  x < a[j]  then
                    cont := false
               else
                    begin
                    {chép nút con j vào nút i}
                         a[i] := a[j];
                         i := j;
                         j := 2 * i;
                    end;
          end;
     a[i] := x
end;
```

Trong C

```
void  Sift(int q, r)
{
     int   i, j, cont;
     ItemType x;
     i = q;
     j = 2*i;
     x = a[i];
     cont = 1;
     while ((j <= r) && cont)
          {
               if  (j < r)
               //tìm nút nhỏ hơn trong 2
               nút con của nút i là  j  và
               j+1
                    if  (a[j] > a[j+1])
                         j := j + 1;
               if  (x < a[j])
                    cont = 0;
               else
                    {
                   //chép nút con j vào nút i
                         a[i] = a[j];
                         i = j;
                         j = 2*i;
                    }
          }
     a[i] = x;
}
```

(*ii*) *Thuật toán tạo Heap ban đầu:*

Trong PASCAL

```
Procedure InitHeap;
var q : integer;
begin
    q := (n div 2) + 1;
    while  q > 1 do
        begin
            q := q - 1;
            Sift(q, n)
        end;
end;
```

Trong C

```
void InitHeap()
{
    int q ;
    q = (n / 2) + 1;
    while  (q > 1)
        {
            q = q - 1;
            Sift(q, n);
        }
}
```

3. Heap Sort

Trước hết ta tạo dãy có *thứ tự giảm dần*. Từ *Heap* đầu *a*[1], *a*[2], ..., *a*[*n*] ta có *a*[1] nhỏ nhất. Đổi chỗ *a*[1] với *a*[*n*], được *a*[*n*] nhỏ nhất và *a*[2], …, *a*[*n*-1] là heap. Gọi *Sift*(1, *n*-1) ta sẽ có *a*[1] nhỏ nhất trong heap *a*[1], *a*[2], ..., *a*[*n*-1]. Lại đổi chỗ *a*[1] với *a*[*n*-1] và lặp lại quá trình trên cho đến khi heap chỉ còn 1 phần tử.

Sau đó ta đảo ngược vị trí của dãy giảm dần và nhận được dãy tăng dần.

Thủ tục HeapSort:

- Đầu vào: dãy *a*[1], ..., *a*[*n*] bất kỳ.
- Đầu ra : dãy *a*[1], ..., *a*[*n*] có thứ tự tăng dần.

Trong PASCAL

```
Procedure HeapSort;
var  r : integer;
Begin
    InitHeap;
    r := n;
    {sắp thứ tự giảm dần}
    while r > 1 do
        begin
            {đổi chỗ a[1] với a[r]}
            swap(a[1], a[r]);
            r := r - 1;
            {tạo heap  a[1], a[2], ... , a[r]}
            Sift(1, r);
        end;
    {sắp thứ tự tăng dần}
    for  r := 1 to (n div 2) do
        swap(a[r], a[n-r+1])
End;
```

Trong C

```
void HeapSort()
{
    int  r ;
    InitHeap();
    r = n;
    //sắp thứ tự giảm dần
    while  (r > 1)
        {
            //đổi chỗ a[1] với a[r]
            swap(&a[1], &a[r]);
            r = r - 1;
            //tạo heap a[1],a[2],...,a[r]
            Sift(1, r);
        }
    //sắp thứ tự tăng dần
    for (r = 1; r <=  (n / 2); r++)
        swap(&a[r], &a[n-r+1]);
}
```

• *Phân tích độ phức tạp:*

Trong trường hợp xấu nhất ta cần

$$HS_1 = log_2(n/2) + log_2(n/2 - 1) + ... + 1 < (n/2)*log_2(n/2)$$

phép toán đổi chỗ để tạo heap ban đầu.

Sau đó ta cần

$$HS_2 = log_2(n-1) + log_2(n-2) + ... + 1 < n*log_2(n)$$

phép toán đổi chỗ để sắp thứ tự. Thêm vào đó là n-1 lần chuyển đổi để đưa phần tử được dịch chuyển về phía phải.

Như vậy độ phức tạp là $O(n*log_2(n))$.

IV. PHƯƠNG PHÁP GIẢM ĐỘ TĂNG

Phương pháp giảm độ tăng (*Diminishing Increment Sort*), còn gọi là *Shell Sort*, do Donald L.Shell đưa ra năm 1959. Khi thực hiện thuật toán này ta dùng dãy phụ chứa độ tăng $h[t], h[t-1], ..., h[1]$, với $h[t] = 1$, để điều khiển quá trình sắp thứ tự. Đặt $h1 = h[1]$ ta khai báo lại dãy *a* như sau:

var a : array[*-h*1..*n*] *of ItemType;*

Các phần tử $a[i]$ với $i \le 0$ là những phần tử cầm canh được sử dụng trong giải thuật chèn trực tiếp.

- *Thủ tục cài đặt thuật toán:*

Trong PASCAL

```
Procedure ShellSort;
const   t = 4;
var  i, j, k, s, m : integer;
     x : ItemType;
     h : array[1..t] of integer;
begin
     h[1] := 9; h[2] := 5;
     h[3] := 3; h[4] := 1;
     for m := 1 to t do
       begin
          k := h[m];
          s := -k;
          for i := k + 1 to n do
            begin
               x := a[i];
               j := i - k;
               if s = 0 then s := -k;
               s := s + 1;
               {sắp thứ tự chèn trực tiếp}
               a[s] := x; {phần tử cầm canh}
               while x < a[j] do
                 begin
                    a[j+k] := a[j];
                    j := j - k;
                 end;
               a[j+k] := x;
            end;
       end;
end;
```

Trong C

```
void ShellSort()
{
     #define  t  4
     int  i, j, k, s, m;
     ItemType  x;
     int  h[t+1];
     h[1] = 9; h[2] = 5;
     h[3] = 3; h[4] = 1;
     for (m = 1; m <= t; m++)
       {
          k = h[m];
          s = -k;
          for (i = k + 1; i <= n; i++)
            {
               x = a[i];
               j = i - k;
               if (s == 0) s = -k;
               s = s + 1;
               //sắp thứ tự chèn trực tiếp
               a[s] = x;//phần tử cầm canh
               while  (x < a[j])
                 {
                    a[j+k] = a[j];
                    j = j - k;
                 }
               a[j+k] = x;
            }
       }
}
```

• *Phân tích độ phức tạp:*

Các bước nhảy $h[1]$, ..., $h[t]$ không được là bội của nhau. KNUTH đưa ra cách chọn hợp lý (viết theo thứ tự ngược):

Dãy 1: 1, 4, 13, 40, ...

$t = [log_3(n)] - 1$

$h[t] = 1;\ h[k\text{-}1] = 3*h[k] + 1$

Dãy 2: 1, 3, 7, 15, ...

$t = [log_2(n)] - 1$

$h[t] = 1;\ h[k\text{-}1] = 2*h[k] + 1$

Độ phức tạp của cách chọn sau là $O(n^{1.25})$.

V. SO SÁNH CÁC PHƯƠNG PHÁP SẮP THỨ TỰ NỘI

Gọi : n là số phần tử cần sắp thứ tự

C là số lần so sánh

M là số lần di chuyển

Trước tiên ta tổng kết đánh giá cho 3 phương pháp trực tiếp.

Phương pháp	Thao tác	Min	Trung bình	Max
Chèn	C =	n - 1	$(n^2 + n - 2)/4$	$n(n-1)/2 - 1$
	M =	$2(n-1)$	$(n^2 + 9n - 10)/4$	$(n^2 + 3n - 4)/2$
Chọn	C =	$n(n-1)/2$	$n(n-1)/2$	$n(n-1)/2$
	M =	$3(n-1)$	$n(ln(n) + 0,57)$	$n^2/4 + 3(n - 1)$
Nổi bọt	C =	$n(n-1)/2$	$n(n-1)/2$	$n(n-1)/2$
	M =	0	$0,75.n.(n-1)$	$1,5.n.(n-1)$

Đối với các phương pháp cải tiến khác thì công thức đánh giá phức tạp hơn. Độ phức tạp tính toán cần thiết cho

Shell Sort là $C*n^{1.25}$

Heap Sort và Quick Sort là $C.n*log_2(n)$

Dựa vào công thức ta có thể phân các phương pháp trên thành các phương pháp đơn giản trực tiếp (tỉ lệ n^2) và các phương pháp cải tiến logarithm (tỉ lệ $log_2(n)$). Tuy nhiên để có thể đánh giá chính xác hơn cần phải có số liệu thực nghiệm về các hệ số C.

Bảng dưới đây cho ta thời gian (milisecond) tính viết bằng PASCAL thực hiện trên máy CDC-6400. Có ba cột ứng với ba trường hợp: dãy có thứ tự, dãy ngẫu nhiên, dãy có thứ tự ngược. Trong mỗi cột có hai cột con ứng với n = 256 và n = 512.

Phương pháp	Có thứ tự		Ngẫu nhiên		Thứ tự ngược	
	n=256	n=512	n=256	n=512	n=256	n=512
Chèn trực tiếp	12	23	366	1444	704	2836
Chèn nhị phân	56	125	373	1327	662	2490
Chọn trực tiếp	489	1907	509	1956	695	2675
Bubble Sort	540	2165	1026	4054	1492	5931
Bubble Sort có cờ	5	8	1104	4270	1645	6542
Shaker Sort	5	9	961	3642	1619	6520
Shell Sort	58	116	127	349	157	492
Heap Sort	116	253	110	241	104	226
Quick Sort	31	69	60	146	37	79

Các số liệu cho thấy sự phân biệt rõ ràng giữa các phương pháp có độ phức tạp O(n^2) và các phương pháp có độ phức tạp O($n*log_2n$).

(*i*) Cải tiến của phương pháp chèn nhị phân so với phương pháp chèn trực tiếp không có ý nghĩa lắm và trong trường hợp dãy có thứ tự ngược thì sự cải tiến này còn xấu hơn.

(*ii*) Phương pháp *Buble Sort* là phương pháp chậm nhất. Ngay cả phương pháp *Shaker Sort* (là phương pháp cải tiến của *Buble Sort*) vẫn còn chậm hơn phương pháp chèn trực tiếp (ngoại trừ trường hợp dãy đã có thứ tự).

(*iii*) *Quick Sort* nhanh hơn *Heap Sort* từ hai đến ba lần. *Quick Sort* sắp dãy có thứ tự ngược vẫn nhanh như trường hợp dãy đã có thứ tự.

Số liệu trên chỉ xét đến các giá trị khoá mà chưa xét đến các dữ liệu phụ kèm theo. Điều này không thực tế lắm. Bảng dưới cho ta kết quả so sánh các phương pháp trên trong trong trường hợp $n = 256$. Cột con trái ứng với mẩu tin không có dữ liệu phụ, cột con phải ứng với mẩu tin có dữ liệu phụ lớn gấp bảy lần khoá.

Phương pháp	**Có thứ tự**		**Ngẫu nhiên**		**Thứ tự ngược**	
		Có DL phụ		Có DL phụ		Có DL phụ
Chèn trực tiếp	12	46	366	1129	704	2150
Chèn nhị phân	56	76	373	1105	662	2070
Chọn trực tiếp	489	547	509	607	695	1430
Bubble Sort	540	610	1026	3212	1492	5599
Bubble Sort có cờ	5	5	1104	3237	1645	5762
Shaker Sort	5	5	961	3071	1619	5757
Shell Sort	58	186	127	373	157	435
Heap Sort	116	264	110	246	104	227
Quick Sort	31	55	60	137	37	75

Từ bảng này ta có nhận xét:

(*i*) Phương pháp chọn trực tiếp khá nhanh và là phương pháp trực tiếp tốt nhất.

(*ii*) *Bubble Sort* vẫn là phương pháp chậm nhất, chỉ có *Shaker Sort* nhanh hơn một ít trong trường hợp dãy có thứ tự ngược.

(*iii*) *Quick Sort* vẫn là phương pháp nhanh nhất, tốt nhất.

BÀI TẬP

1. Viết chương trình nhập các số nguyên n_1, n_2, n_3, a, b ($0<n_1 \le 10$, $0<n_2,n_3 \le 100$, $a < b$, n_1, n_2, n_3 << $b-a < 1000$) và thực hiện các công việc sau:

a) Tạo danh sách đặc *L1* gồm n_1 số nguyên trong khoảng [a;b] nhập vào từ bàn phím.

b) Tạo danh sách đặc *L2* gồm n_2 số nguyên ngẫu nhiên trong khoảng [a; b].

c) Tạo danh sách đặc *L3* gồm n_3 số nguyên ngẫu nhiên *khác nhau* trong khoảng [a; b].

d) Sắp xếp danh sách *L1* tăng dần bằng phương pháp chèn, phương pháp nổi bọt.

e) Sắp xếp danh sách *L2* tăng dần bằng phương pháp chọn.

f) Sắp xếp danh sách *L3* tăng dần bằng phương pháp sắp xếp nhanh.

g) Trộn hai danh sách *L1* và *L2*.

h) Trộn ba danh sách *L1, L2* và *L3*.

2. Tương tự như bài **1**, trong đó yêu cầu sắp xếp các danh sách giảm dần.

3. Viết chương trình nhập các số nguyên n_1, n_2, n_3, a, b ($0<n_1 \le 10$, $0<n_2,n_3 \le 100$, $a < b$, n_1, n_2, n_3 << $b-a < 1000$) và thực hiện các công việc sau:

a) Tạo danh sách liên kết *L1* gồm n_1 số nguyên ngẫu nhiên trong khoảng [a;b].

b) Tạo danh sách liên kết *L2* gồm n_2 số nguyên ngẫu nhiên khác nhau trong khoảng [a; b].

c) Sắp xếp danh sách *L1* tăng dần bằng phương pháp chèn.

d) Sắp xếp danh sách *L2* tăng dần bằng phương pháp chọn.

e) Trộn hai danh sách *L1* và *L2*.

4. Tương tự như bài 3, trong đó yêu cầu sắp xếp các danh sách giảm dần.