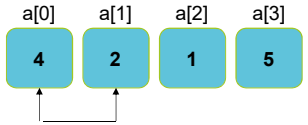
CHƯƠNG III
CÁC GIẢI THUẬT SẮP XẾP

3.1 Bài toán sắp xếp

- Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định.
- Để tìm hiểu các giải thuật sắp xếp ta sẽ minh họa bằng bài toán sắp xếp mảng a[n] theo thứ tự tăng dần.
- Để sắp xếp dãy a theo thứ tự tăng, ta tiến hành triệt tiêu tất cả các nghịch thế trong a.



a[0], a[1] là cặp nghịch thế vì a[0]>a[1]

Các giải thuật sắp xếp thông dụng

- Inter Change Sort
- Selection Sort
- Insertion Sort
- Bubble Sort
- Quick Sort
- Heap Sort
- Shell Sort
- Merge Sort
- Radix Sort

3.2 Đổi chỗ trực tiếp

Ý tưởng:

- Xuất phát từ đầu dãy, tìm tất các các nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ 2 phần tử trong cặp nghịch thế.
- Lặp lại xử lý trên với phần tử kế trong dãy.

Các bước tiến hành

- Bước 1: i = 0; *// bắt đầu từ đầu dãy*
- Bước 2: j = i+1; *//tìm các nghịch thế với a[i]*
- Bước 3:

  Trong khi j < N thực hiện

  Nếu a[j]<a[i] *//xét cặp a[i], a[j]*

  đổi chỗ (a[i],a[j]);

  j = j+1;
- Bước 4: i = i+1;

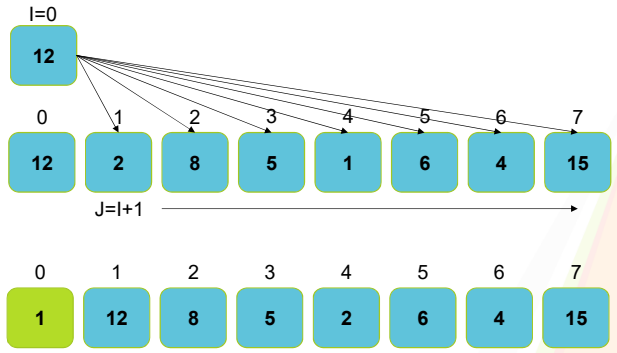
  Nếu i < N-1: Lặp lại Bước 2.

  Ngược lại: Dừng.

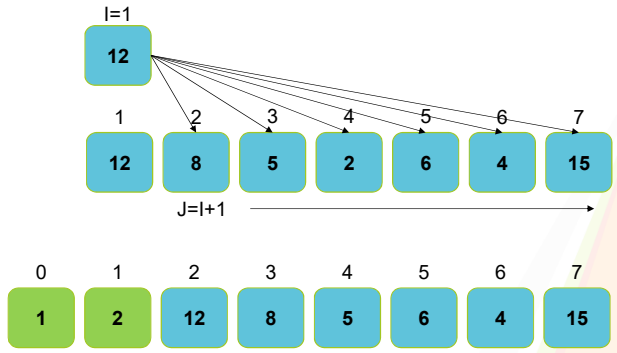
Cài đặt thuật toán

```
void InterchangeSort(int a[], int n){
    for(int  i=0; i<n-1; i++)
       for(int  j=i+1; j<n; j++)
          if(a[j]<a[i]){
             //Đổi chỗ a[i] và a[min] cho nhau
             int t=a[i];
             a[i]=a[j];
             a[j]=t;
          }
}
```

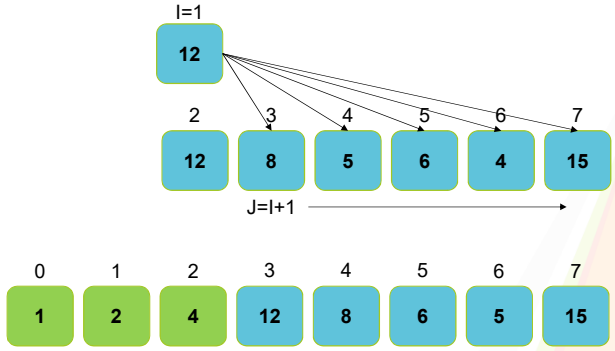

Minh họa



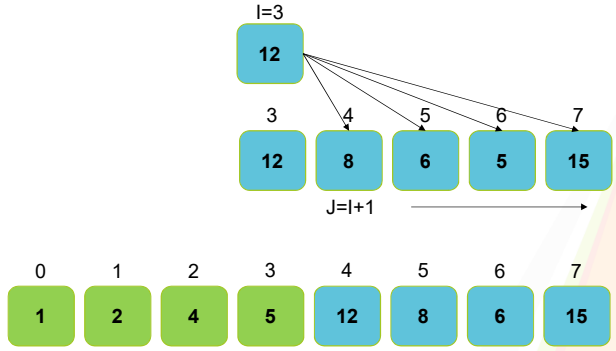
Minh họa



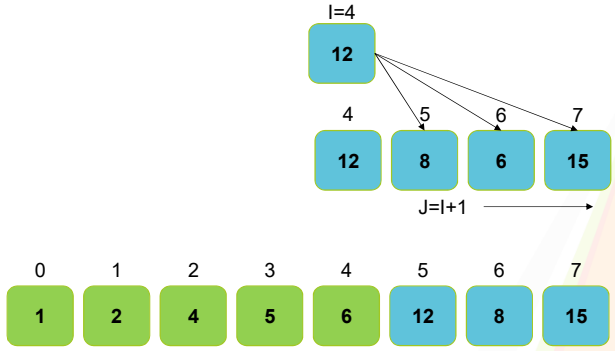
Minh họa



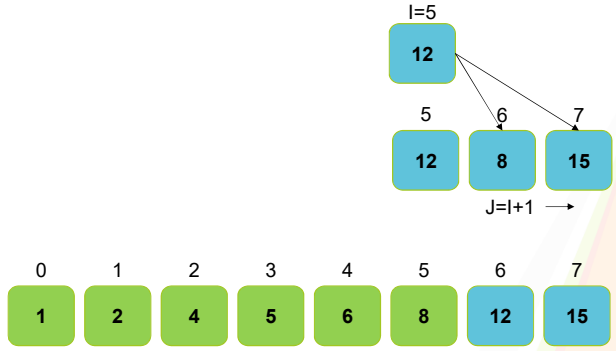
Minh họa



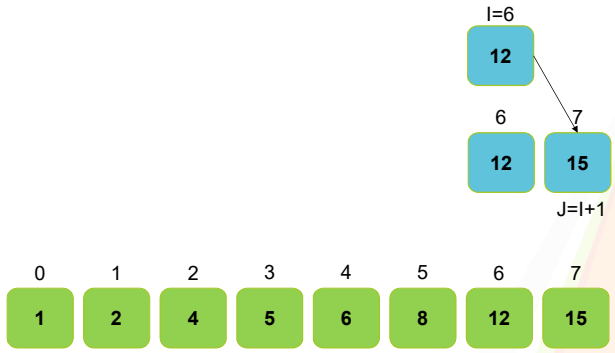
Minh họa



Minh họa



Minh họa



3.3 Chọn trực tiếp

Ý tưởng:

- Tại vị trí đầu tiên của dãy, chọn phần tử nhỏ nhất trong N phần tử trong dãy hiện hành ban đầu và đưa phần tử này về vị trí đầu dãy hiện hành.
- Xem dãy hiện hành chỉ còn N-1 phần tử của dãy hiện hành ban đầu. Bắt đầu từ vị trí thứ 2, chọn phần tử nhỏ nhất trong số N-1 phần tử kể trên và đưa nó về vị trí đầu của dãy gồm N-1 phần tử đang xét.
- Lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử.

3.3 Chọn trực tiếp

Các bước thực hiện:

- Bước 1: i = 0;
- Bước 2: Tìm phần tử **a[min]** nhỏ nhất trong dãy hiện hành từ a[i] đến a[N]
- Bước 3 : Đổi chỗ a[min] và a[i]
- Bước 4 : Nếu i < N-1 thì

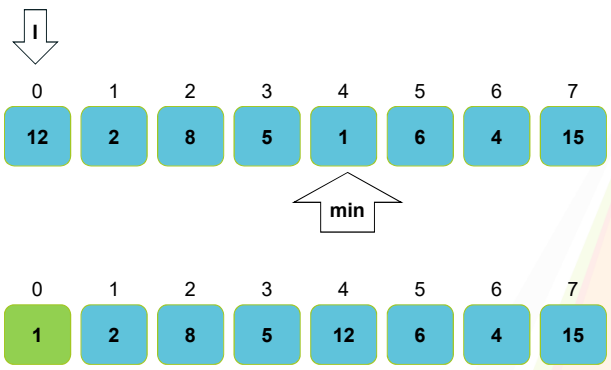
  i = i+1; Lặp lại Bước 2;

  Ngược lại: Dừng.

Cài đặt thuật toán

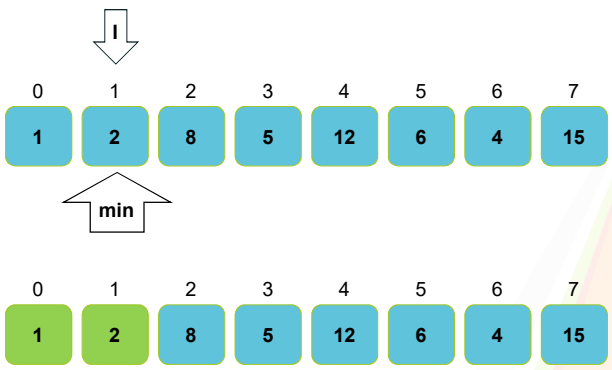
```
void SelectionSort(int a[], int n ){
    int min;//dùng để lưu vị trí ptử nhỏ nhất
    for (int i=0; i<n-1 ; i++){
          min = i;
          for(int j= i+1; j<n ; j++)
              if (a[j]< a[min])
                   min = j;
              int t=a[i];
          a[i]=a[min];
          a[min]=t;
    }
}
```

Minh họa



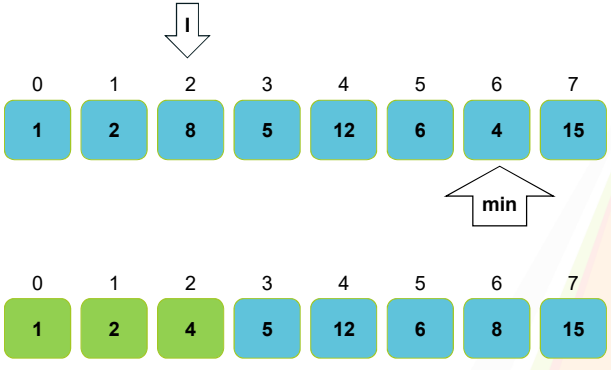
17

Minh họa



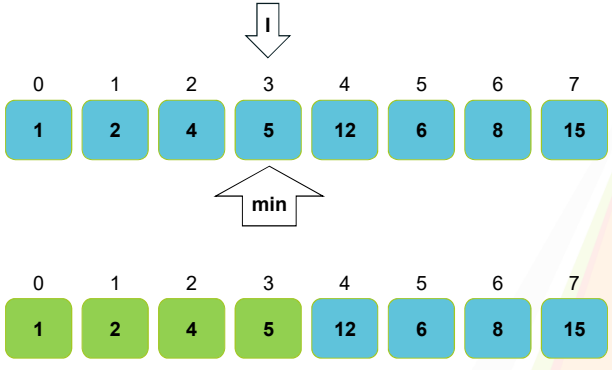
18

Minh họa



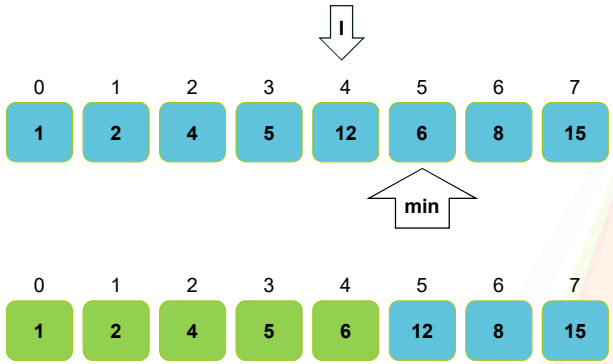
19

Minh họa

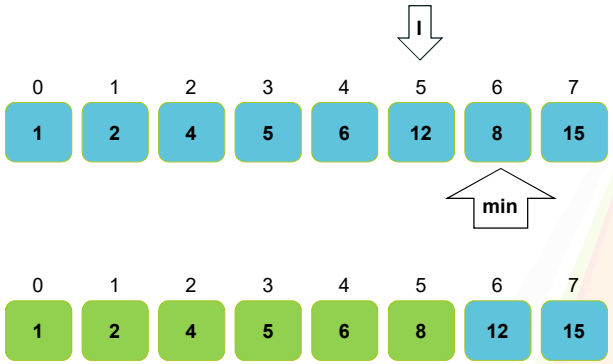


20

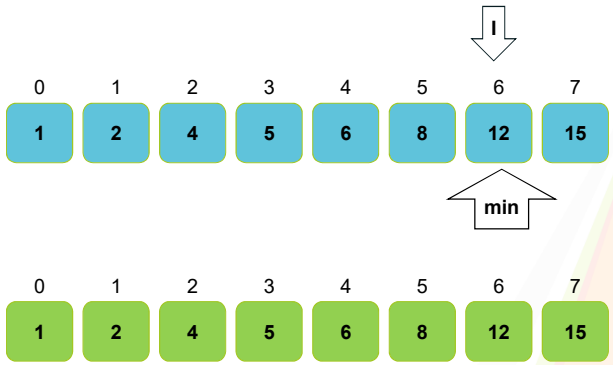
Minh họa



Minh họa



Minh họa



3.4 Chèn trực tiếp

Ý tưởng:

- Ban đầu coi như phần tử đầu tiên đã sắp xếp.
- So sánh phần tử thứ 2 với phần tử đầu tiên để đổi chỗ nếu cần để sao cho 2 phần tử này được sắp xếp theo thứ tự chỉ định.
- Tìm vị trí để chèn phần tử thứ 3 của dãy hiện hành vào dãy 2 phần tử đầu đã được sắp xếp ở trên sao cho dãy vẫn được sắp xếp đúng thứ tự.
- Lặp lại liên tục các quá trình trên cho tới khi hết dãy.

Các bước thực hiện

- Bước 1: i = 1; //giả sử có đoạn a[0] đã được sắp
- Bước 2: x = a[i]; Tìm vị trí pos thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào
- Bước 3: Dời chỗ 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[1]..a[i] đã được sắp
- Bước 5: i = i+1;

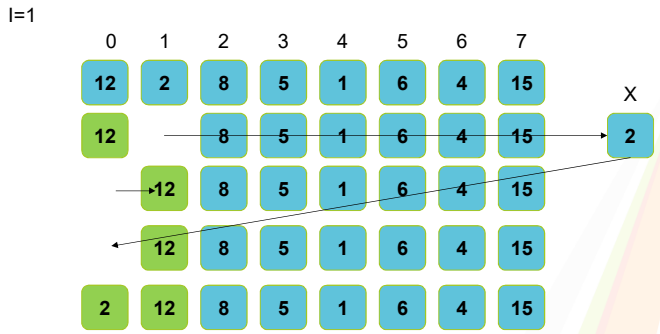
  Nếu i < n : Lặp lại Bước 2

  Ngược lại : Dừng

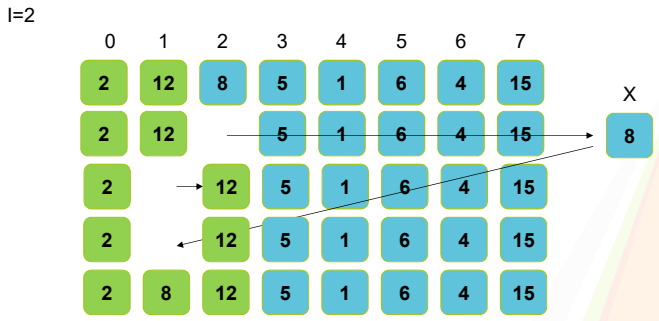
Cài đặt thuật toán

```
void InsertionSort(int a[], int n){
     int pos, i, x;
     for(i=1; i<n; i++){
        x=a[i];
        pos=i-1;
        while(pos>=0 && a[pos]>x){
              a[pos+1]=a[pos];
              pos--;
        }
        a[pos+1]=x;
     }
}
```

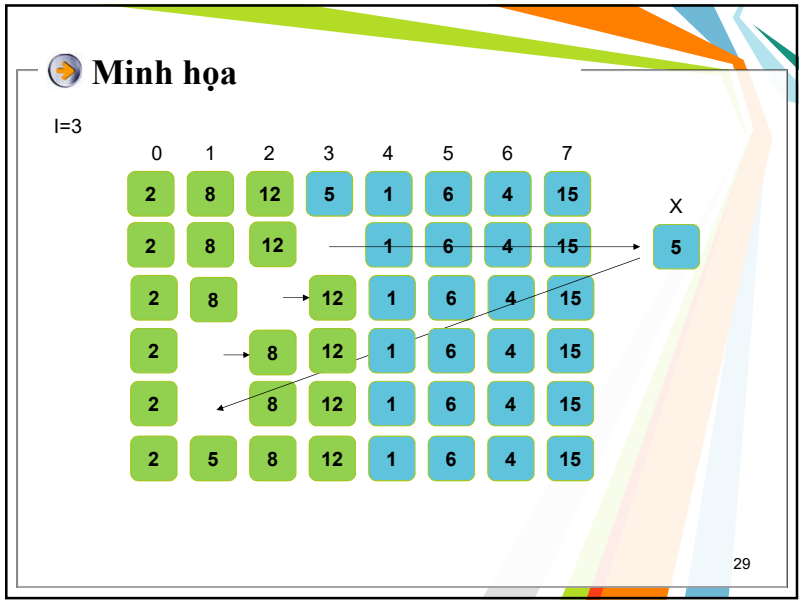

Minh họa



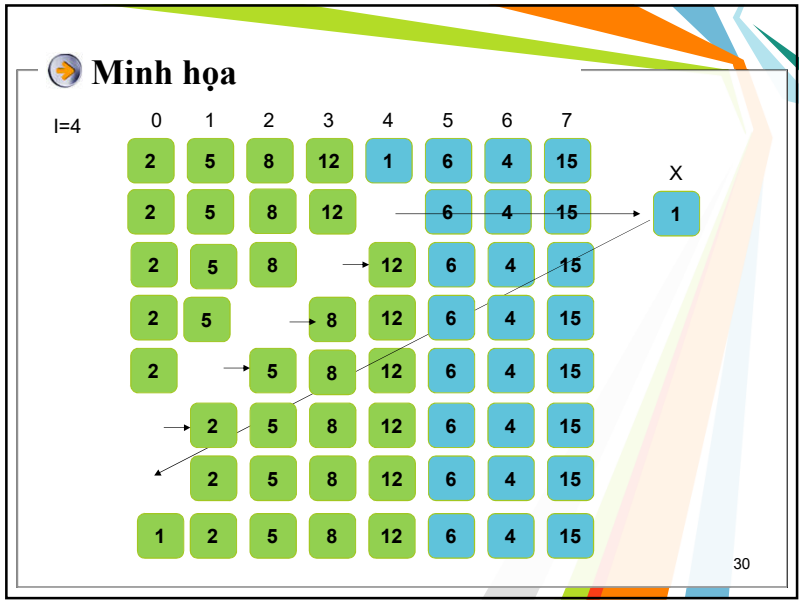
Minh họa



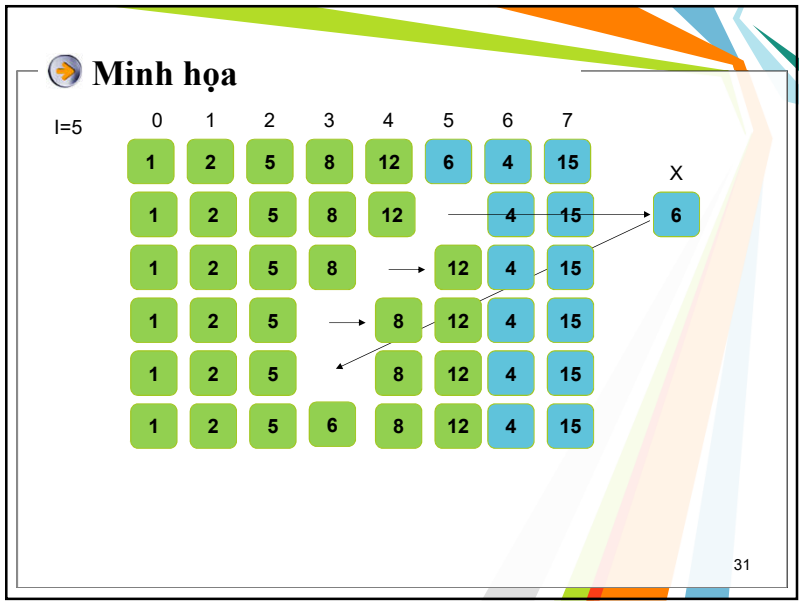
Minh họa



Minh họa

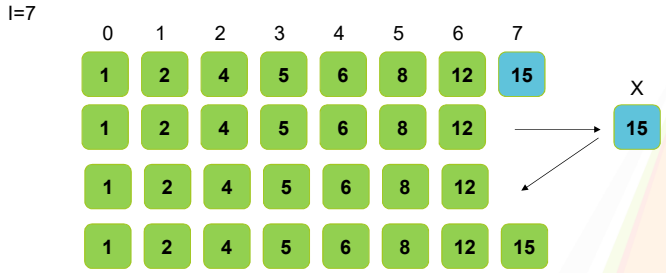


Minh họa



Minh họa

Minh họa



3.5 Nổi bọt - Bubble Sort

Ý tưởng:

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đúng đầu dãy hiện hành, 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.

Các bước thực hiện

- 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]

  Doicho(a[j],a[j-1]);

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

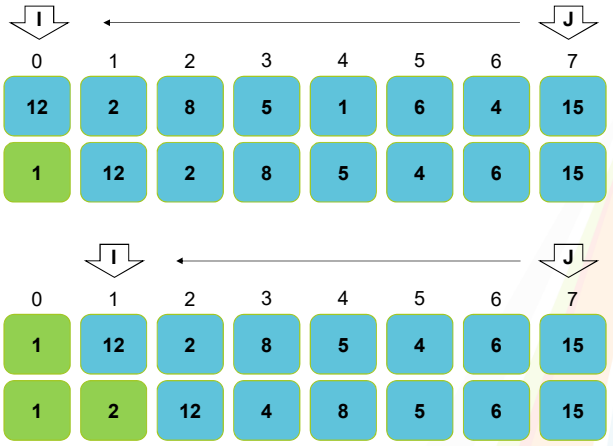
  Nếu i =N-1: Hết dãy. Dừng

  Ngược lại : Lặp lại Bước 2.

Cài đặt thuật toán

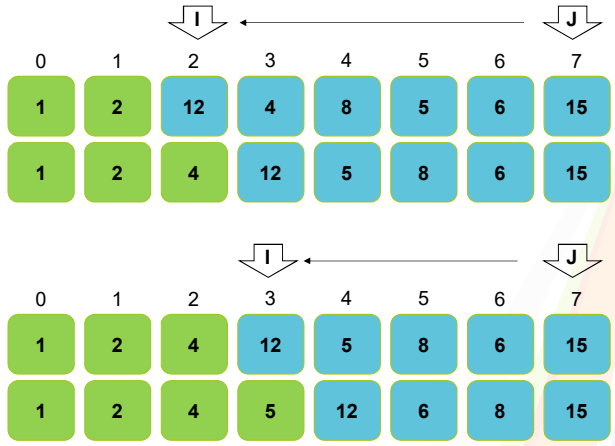
```
void BubbleSort(int a[], int n){
   for (int i=0; i<n-1; i++)
       for (int j=n-1; j>i; j--)
         if(a[j]<a[j-1]){
               int t=a[j];
               a[j]=a[j-1];
               a[j-1]=t;
         }
}
```

Minh họa



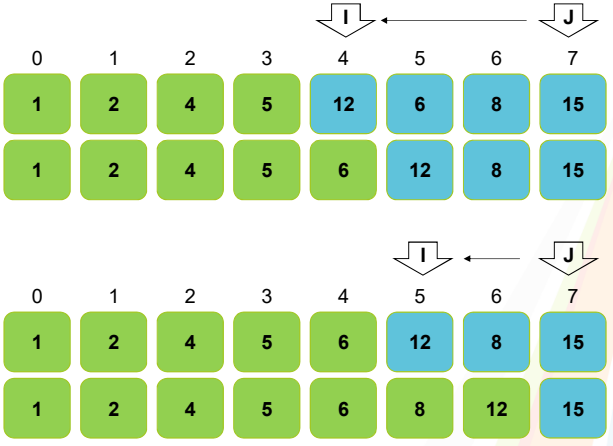
37

Minh họa



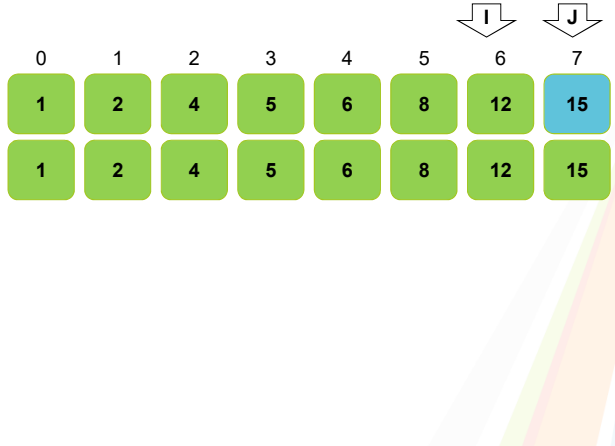
38

Minh họa



39

Minh họa



40

3.6 Quick Sort

Ý tưởng:

- QuickSort chia mảng thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt.
- Những phần tử nhỏ hơn phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn hoặc bằng phần tử chốt được đưa về phía sau và thuộc danh sách con thứ hai.
- Cứ tiếp tục chia các danh sách con như vậy tới khi các danh sách con đều có độ dài bằng 1.

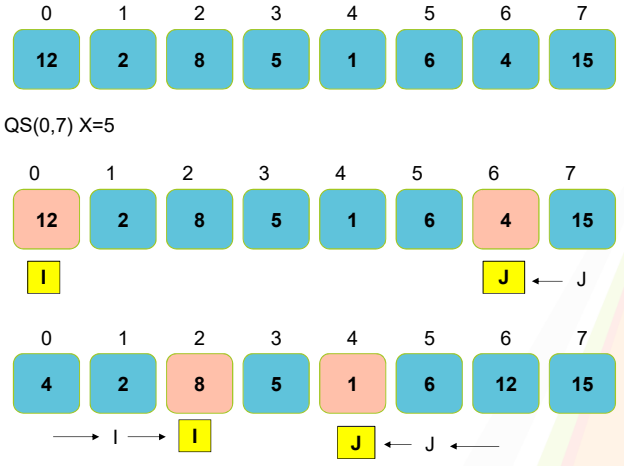
Các bước thực hiện

- ***Bước 1***: *Lấy 1 phần tử bất kì của dãy làm phần tử chốt X (giả sử lấy phần tử đầu tiên hoặc phần tử giữa của dãy,...).*
- ***Bước 2***: *Tiến hành duyệt từ bên trái dãy cho đến khi gặp phần tử lớn hơn hoặc bằng X thì dừng. Đồng thời duyệt từ bên phải dãy cho đến khi gặp phần tử nhỏ hơn hoặc bằng X thì dừng.*
- ***Bước 3***: *Đổi chỗ 2 phần tử tìm được ở bước 2.*
- ***Bước 4***: *Tiếp tục duyệt cho đến khi 2 biến duyệt từ 2 chiều gặp nhau. Khi đó dãy ban đầu đã phân thành 2 phần: phần trái gồm những phần tử nhỏ hơn X, phần phải gồm những phần tử lớn hơn hoặc bằng X.*
- ***Bước 5***: *Lặp lại quá trình phân hoạch đối với phần trái và phần phải mới được tạo ở trên cho đến khi nào dãy được sắp xếp hoàn toàn.*

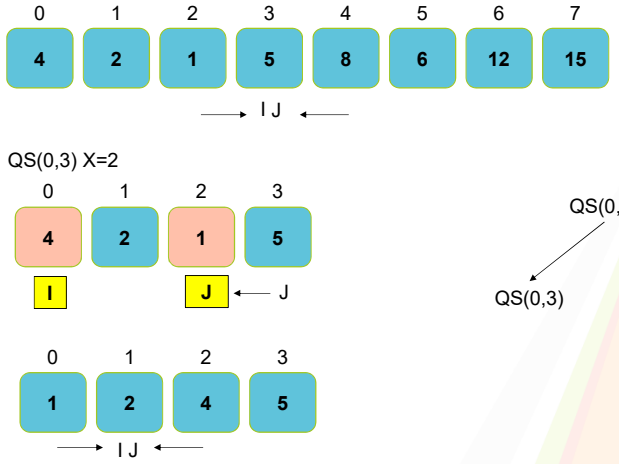
Cài đặt thuật toán Quick Sort

```
void QuickSort(int a[],int left,int right){
   int i, j, x;
   x=a[(left+right)/2];
   i=left; j=right;
   do {
      while(a[i] < x) i++;
      while(a[j] > x) j--;
      if(i<=j){
         int t=a[i]; a[i]=a[j]; a[j]=t;
         i++; j--;
      }
   } while (i<j);
   if(left<j) QuickSort(a, left, j);
   if(i<right) QuickSort(a, i, right);
}
```

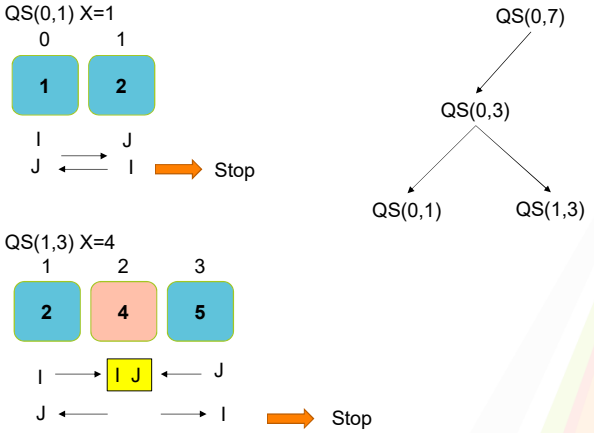

Ví dụ



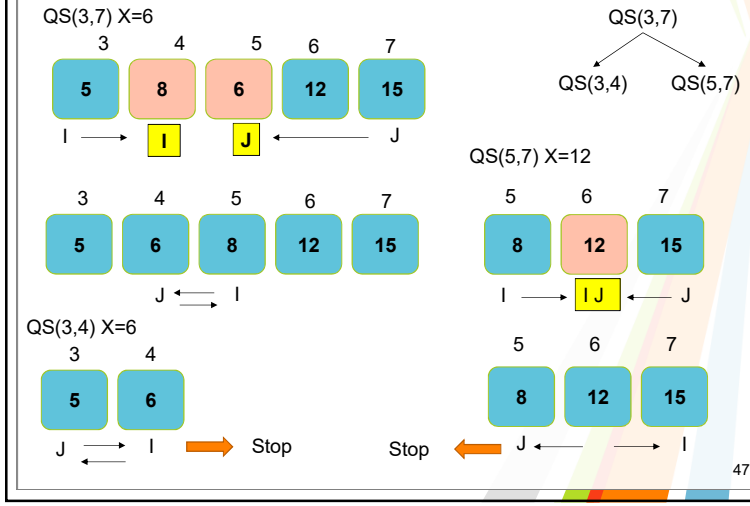
Ví dụ



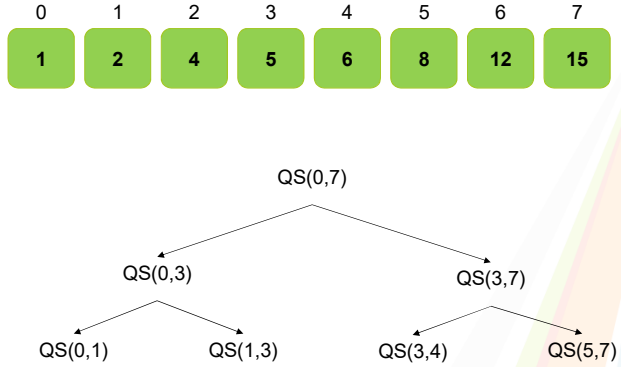
Ví dụ



Ví dụ



Ví dụ



3.7 Sắp xếp theo cây – Heap Sort

Ý tưởng của giải thuật Heap Sort:

Sắp xếp thông qua việc tạo các heap (đống) từ dãy ban đầu, trong đó heap là 1 cây nhị phân hoàn chỉnh có tính chất là khóa ở nút cha bao giờ cũng lớn hơn khóa ở các nút con.

Các bước thực hiện

Việc thực hiện giải thuật được chia thành 2 giai đoạn:

- **Giai đoạn 1**: Tạo heap từ dãy ban đầu, khi đó nút gốc của heap là phần tử lớn nhất.
- **Giai đoạn 2**: Sắp xếp dãy dựa trên heap được tạo.
  - *Nút gốc là nút có giá trị lớn nhất, ta chuyển về vị trí cuối cùng của heap.*
  - *Loại phần tử lớn nhất khỏi heap.*
  - *Hiệu chỉnh phần còn lại thành heap.*
  - *Lặp lại quá trình cho tới khi heap chỉ còn 1 nút.*

Hàm hiệu chỉnh heap

```
void Shift(int a[], int l, int r){
   int i=l, j=2*i+1, x=a[i];
   while(j<=r){
       if(j<r && a[j]<a[j+1]) j++;
       if(a[j]<=x)
          return;
       else {
          a[i]=a[j]; a[j]=x;
          i=j; j=2*i+1;
          x=a[i];
       }
   }
}
```

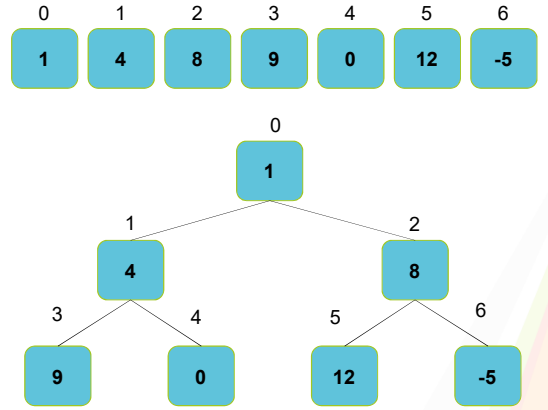

Hàm tạo heap từ dãy ban đầu

```
void CreateHeap(int a[], int n){
   int l=n/2-1;
   while(l>=0){
         Shift(a, l, n-1);
         l=l-1;
   }
}
```

Hàm Heap Sort

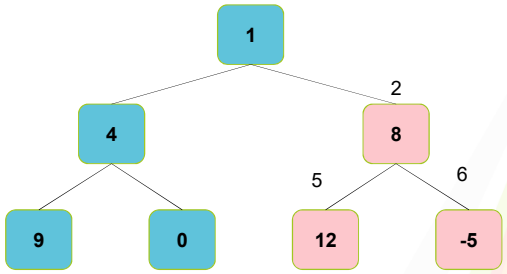
```
void HeapSort(int a[], int n){
   int r;
   CreateHeap(a, n);
   r=n-1;
   while(r>0){
         int t=a[0]; a[0]=a[r]; a[r]=t;
         r--;
         if(r>0) Shift(a, 0, r);
   }
}
```


Minh họa



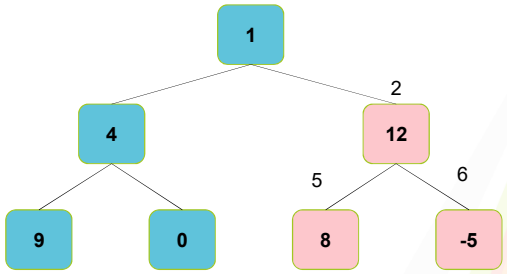
Minh họa

n=7, l=n/2-1=2, r=n-1=6
Shift(a,l,r) ->Shift(a,2,6)
i=l=2, j=2i+1=5



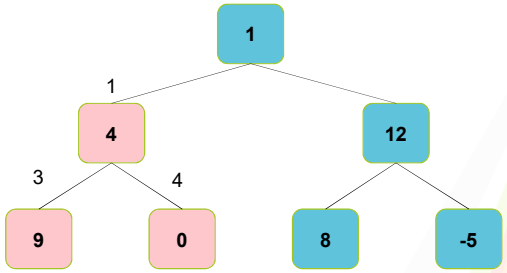
Minh họa

Hoán vị a[2] với a[5]
i=j=5, j=2i+1=11 ***(>r)***



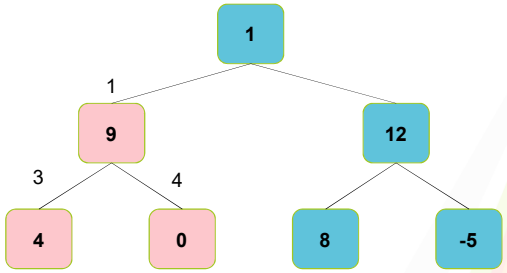
Minh họa

l=l-1=1. Shift(a,1,6),
i=l=1, j=2i+1=3



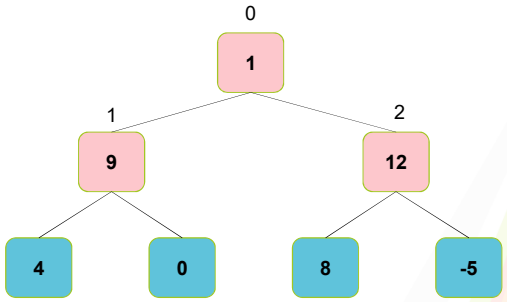
Minh họa

Hoán vị a[3] với a[1]
i=j=4, j=2i+1=9 (>r)



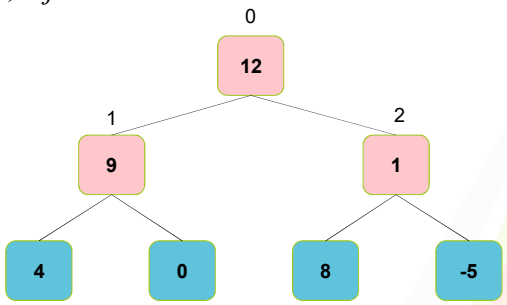
Minh họa

l=l-1=0, Shift(a,0,6)
i=l=0, j=2i+1=1



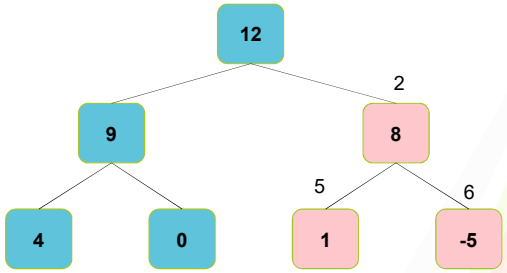
Minh họa

Do a[j]=1<a[j+1]=2 ->j=2
Hoán vị a[0] với a[2]
i=j=2, j=2i+1=5



Minh họa

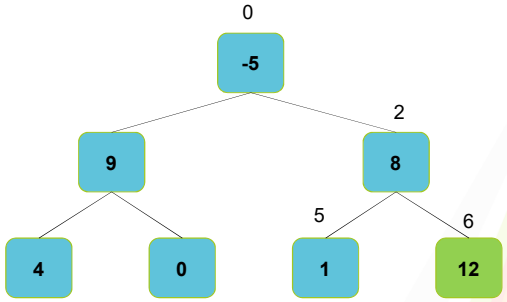
Hoán vị a[2] với a[5]
i=j=5, j=2i+1=11 ***(>r)***



61

Minh họa

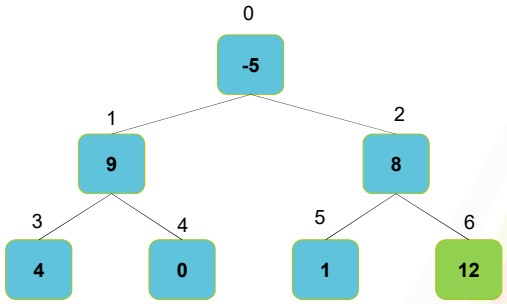
Lấy phần tử lớn nhấy đưa vào cuối, hiệu chỉnh phần còn lại thành heap



62

Minh họa

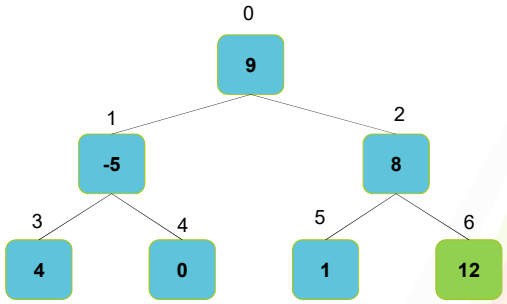
r=5, shift(a,0,5)
i=l=0, j=2i+1=1



63

Minh họa

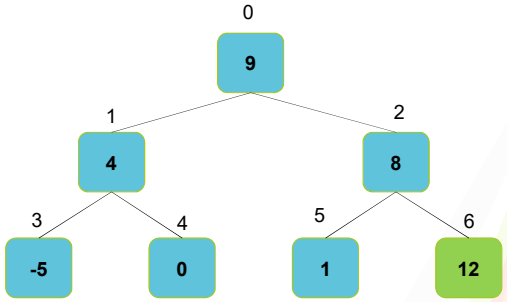
Hoán vị a[0] với a[1]
i=j=1, j=2i+1=3



64

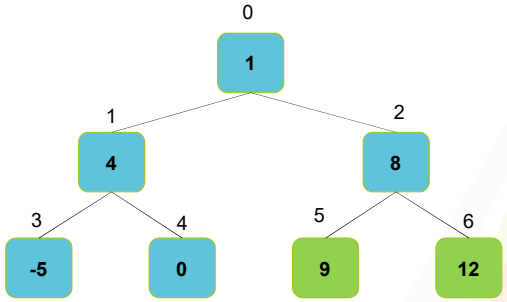
Minh họa

Hoán vị a[1] với a[3]
i=j=3, j=2i+1=7 ***(>r)***

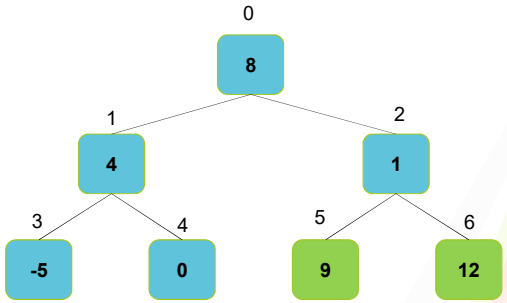


Minh họa

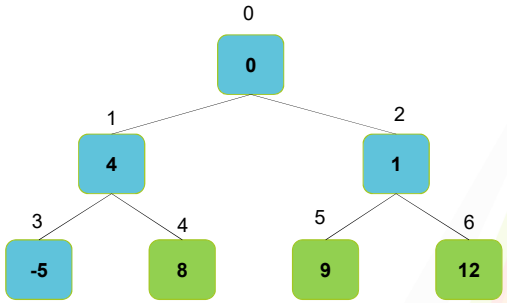
Lấy phần tử lớn nhấy đưa vào cuối, hiệu chỉnh phần còn lại thành heap



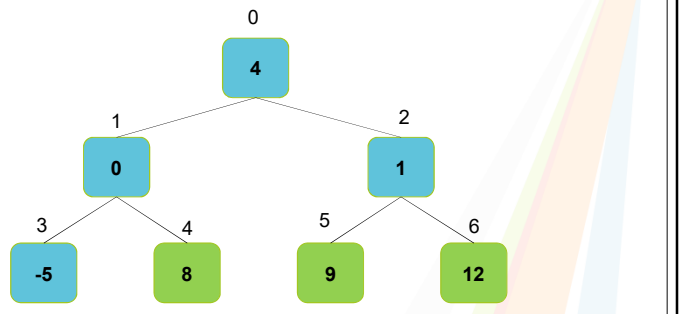
Minh họa



Minh họa

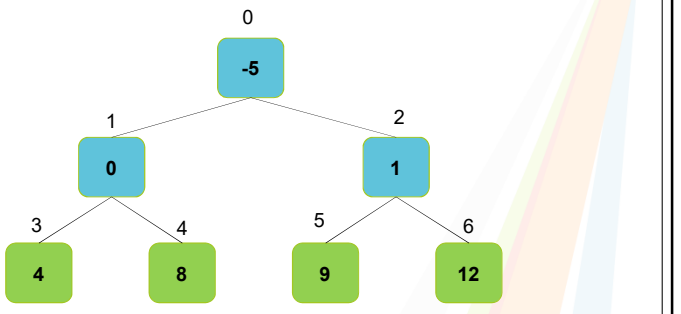


Minh họa



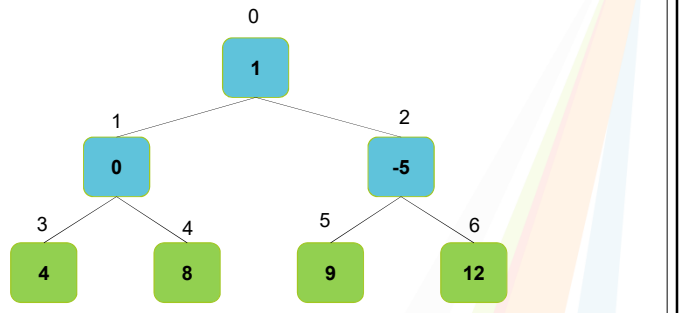
69

Minh họa



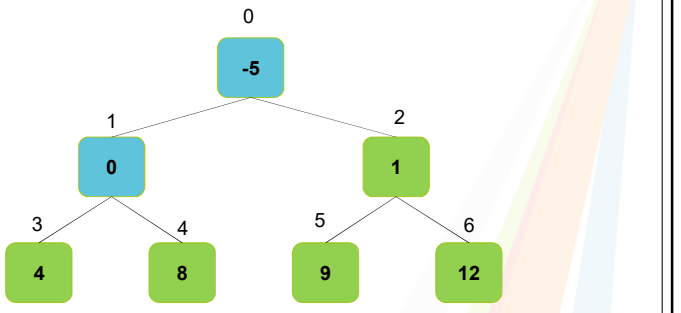
70

Minh họa



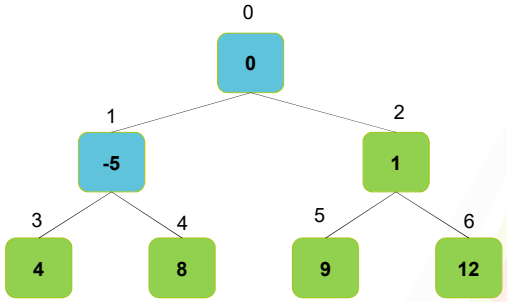
71

Minh họa

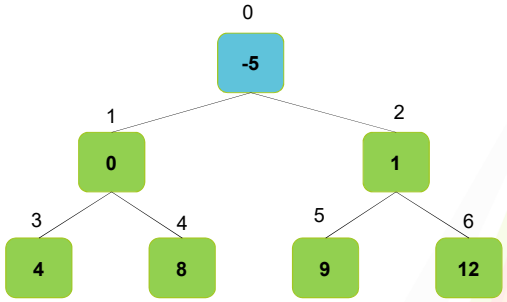


72

Minh họa

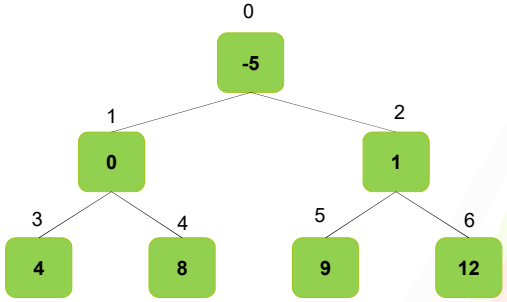


Minh họa



Minh họa

Ta được kết quả sau cùng:



3.8 Shell Sort

Ý Tưởng:

- *Phân hoạch dãy thành các dãy con.*
- *Sắp xếp các dãy con theo phương pháp chèn trực tiếp.*
- *Dùng phương pháp chèn trực tiếp sắp xếp lại cả dãy.*

Cách thực hiện

• Phân chia dãy ban đầu thành những dãy con gồm các phần tử ở cách nhau h vị trí;

• Dãy ban đầu : a_1, a_2, ..., a_n được xem như sự xen kẽ của các dãy con sau :

- Dãy con thứ nhất: a[1], a[h+1], a[2h+1] , ...
- Dãy con thứ hai: a[2], a[h+2], a[2h+2] ...
- ....
- Dãy con thứ h: a[h], a[2h], a[3h], ...

• Tiến hành sắp xếp các phần tử trong cùng dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối ;

• Giảm khoảng cách h để tạo thành các dãy con mới ;

• Dừng khi h=1

Các bước tiến hành

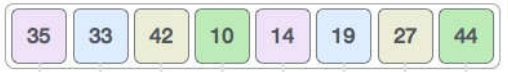
- Bước 1: Khởi tạo giá trị h theo công thức Knuth như sau: h=h*3+1 (h là Khoảng - interval với giá trị ban đâu là 1)
- Bước 2: Chia list thành các sublist nhỏ hơn tương ứng với h
- Bước 3: Sắp xếp các sublist này bởi sử dụng sắp xếp chèn (Insertion Sort)
- Bước 4: Lặp lại cho tới khi list đã được sắp xếp

Cài đặt Shell Sort

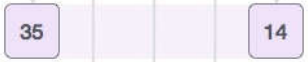
```
void ShellSort(int a[], int n){
   int  i, j, x, h=1;
   while (h <= n/3)
       h=h*3+1;
   while (h > 0){
       for (i = h; i<n; i++){
           x=a[i];
           j=i-h;
           while ((x<a[j]) && (j>=0)){
              a[j+h]=a[j];
              j=j-h;
           }
           a[j+h]=x;
       }
       h=(h-1)/3;
   }
}
```


Ví dụ

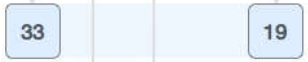
h=1*3+1=4:



i=4, j=0

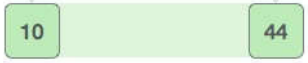


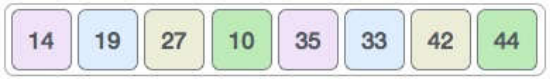
i=5, j=1



i=6, j=2

i=7, j=3



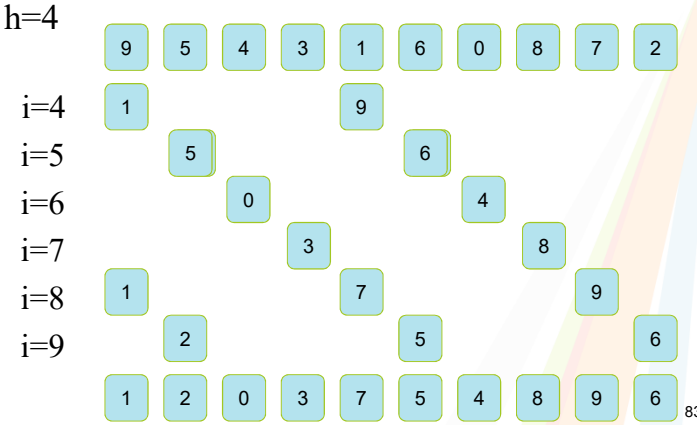


Ví dụ

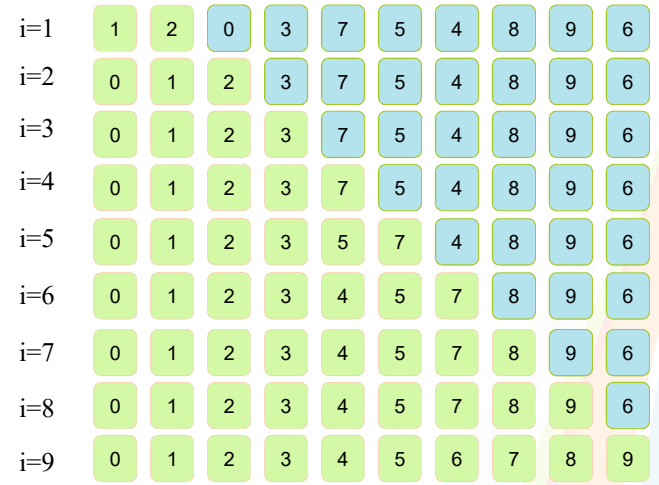
h=(h-1)/3=(4-1)/3=1

Ví dụ

h=4



h=1



3.9 Trộn trực tiếp – Merge Sort

Ý tưởng:

Ý tưởng của giải thuật là trộn 2 dãy đã được sắp xếp thành 1 dãy mới cũng được sắp xếp.

Đầu tiên ta coi mỗi phần tử của dãy là 1 danh sách con gồm 1 phần tử đã được sắp. Tiếp theo tiến hành trộn từng cặp 2 dãy con 1 phần tử kề nhau để tạo thành các dãy con 2 phần tử được sắp. Các dãy con 2 phần tử được sắp này lại được trộn với nhau tạo thành dãy con 4 phần tử được sắp. Quá trình tiếp tục đến khi chỉ còn 1 dãy con duy nhất được sắp, đó chính là dãy ban đầu.

Các bước thực hiện

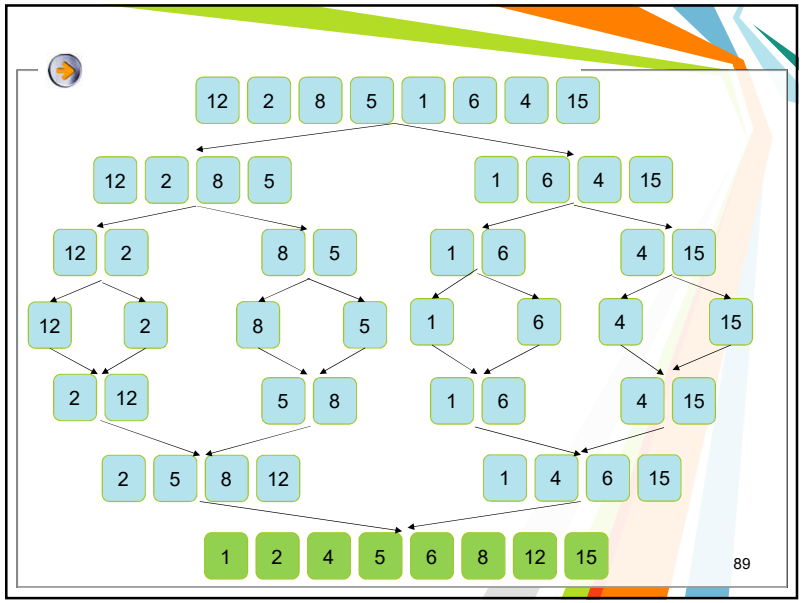
- Bước 1: Nếu chỉ có một phần tử trong list thì list này được xem như là đã được sắp xếp. Trả về list hay giá trị nào đó.
- Bước 2: Chia list một cách đệ qui thành hai nửa cho tới khi không thể chia được nữa.
- Bước 3: Kết hợp các list nhỏ hơn (đã qua sắp xếp) thành list mới (cũng đã được sắp xếp).

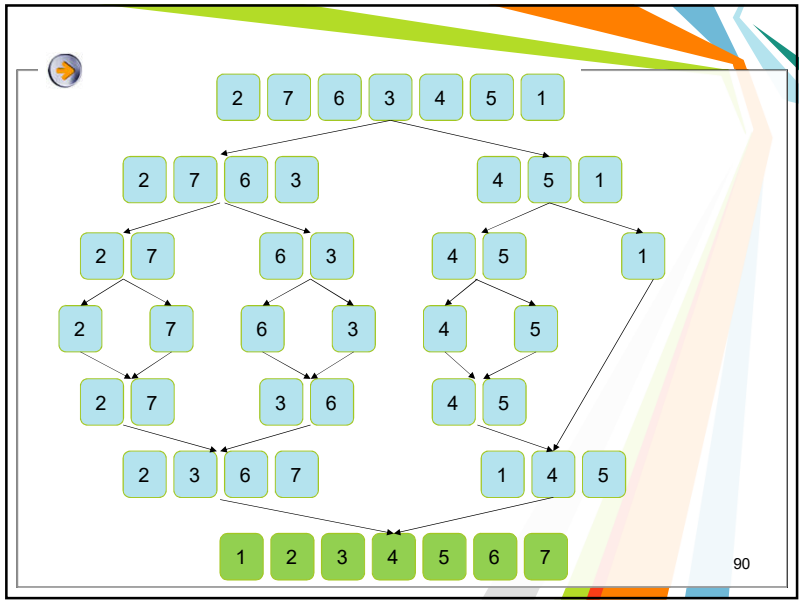
Cài đặt – Merge Sort

```
void merge(int a[], int first, int mid, int last){
    int temp[last], first1, last1, first2, last2, i=first;
    first1=first; last1=mid;
    first2=mid+1; last2=last;
    while((first1<=last1) && (first2<=last2))
        if (a[first1]<a[first2])
            temp[i++]=a[first1++];
        else
            temp[i++]=a[first2++];
    while(first1<=last1)
        temp[i++]=a[first1++];
    while(first2 <= last2)
        temp[i++]=a[first2++];
    for(i=first; i<= last; i++)
       a[i]=temp[i];
}
```


Cài đặt – Merge Sort

```
void mergeSort(int a[], int first, int last){
    if(first < last){
        int mid = int((first+last)/2);
        mergeSort(a, first, mid);
        mergeSort(a, mid+1, last);
        merge(a, first, mid, last);
    }
}
```





Trộn tự nhiên – Natural Merge Sort

Đường chạy (run):

Một đường chạy của dãy số a là một dãy con không giảm của cực đại của a. Nghĩa là, đường chạy r = (ai, ai+1, ., aj) phải thỏa điều kiện:

- $0 \le i \le j < n$, với n là số phần tử của dãy a
- $a_k \le a_{k+1}$ $\forall k, i \le k \le j$

Ví dụ:

Dãy 12, 2, 8, 5, 1, 6, 4, 15 có thể coi như gồm 5 đường chạy (12); (2, 8); (5); (1, 6); (4, 15).

Trộn tự nhiên – Giải thuật

Bước 1 : // Chuẩn bị

r = 0; // r dùng để đếm số đường chạy

Bước 2 :

Tách dãy a1, a2, ..., an thành 2 dãy b, c theo nguyên tắc luân phiên từng đường chạy:

Bước 2.1 :

Phân phối cho b một đường chạy; r = r+1;

Nếu a còn phần tử chưa phân phối

Phân phối cho c một đường chạy; r = r+1;

Bước 2.2 :

Nếu a còn phần tử: quay lại bước 2.1;

Trộn tự nhiên – Giải thuật

Bước 3 :

Trộn từng cặp đường chạy của 2 dãy b, c vào a.

Bước 4 :

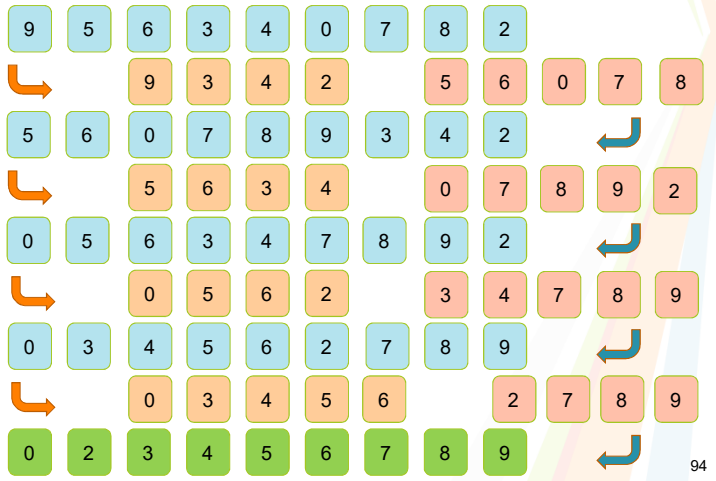
Nếu r <= 2 thì trở lại bước 2;

Ngược lại: Dừng;

Cài đặt:

Xem như bài tập

Trộn tự nhiên – Ví dụ



3.10 Sắp xếp theo cơ số - Radix Sort

- Radix Sort là một thuật toán tiếp cận theo một hướng hoàn toàn khác.
- Nếu như trong các thuật toán khác, cơ sở để sắp xếp luôn là việc so sánh giá trị của 2 phần tử thì Radix Sort lại dựa trên nguyên tắc phân loại thư của bưu điện. Vì lý do đó Radix Sort còn có tên là Postman's Sort.
- Radix Sort không hề quan tâm đến việc so sánh giá trị của phần tử mà bản thân việc phân loại và trình tự phân loại sẽ tạo ra thứ tự cho các phần tử.

3.10 Sắp xếp theo cơ số - Radix Sort

- Mô phỏng lại qui trình trên, để sắp xếp dãy a_1, a_2, ..., a_n, giải thuật Radix Sort thực hiện như sau:
  - Trước tiên, ta có thể giả sử mỗi phần tử a_i trong dãy a_1, a_2, ..., a_n là một số nguyên có tối đa m chữ số.
  - Ta phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm, … tương tự việc phân loại thư theo tỉnh thành, quận huyện, phường xã, ….

Thuật toán Radix Sort

- Bước 1 :// k cho biết chữ số dùng để phân loại hiện hành
  - d = 0; // d = 0: hàng đơn vị; d = 1: hàng chục; …
- Bước 2 : //Tạo các lô chứa các loại phần tử khác nhau
  - Khởi tạo 10 lô B_0, B_1, …, B_9 rỗng;
- Bước 3 :
  - For i = 1 .. n do
    - Đặt a_i vào lô B_t với t: chữ số thứ d của a_i;
- Bước 4 :
  - Nối B_0, B_1, …, B_9 lại (theo đúng trình tự) thành a.
- Bước 5 :
  - d = d+1;Nếu d < m thì trở lại bước 2. Ngược lại: Dừng
  - (m là số chữ số tối đa)

Cài đặt Radix Sort

```
void RadixSort(int a[], int n){
        const int MAX=100, m=5;
        int digit, num, h=10, B[10][MAX], Len[10];
        for(int d=0; d<m; d++){
            for(int i=0; i<10; i++) Len[i]=0;
            for(int i=0; i<n; i++){
               digit=(a[i]%h)/(h/10);
               B[digit][Len[digit]++]=a[i];
            }
            num=0;
            for(int i=0; i<10; i++)
                for(int j=0; j<Len[i]; j++)
                    a[num++]=B[i][j];
            h*=10;
        }
}
```


Ví dụ - Radix Sort

12	0701										
11	1725										
10	0999										
9	9170										
8	3252										
7	4518										
6	7009										
5	1424										
4	0428										
3	1239										
2	8425										
1	7013										
CS	A	0	1	2	3	4	5	6	7	8	9

Ví dụ - Radix Sort

12											
11											
10											
9											
8											
7											
6											
5											
4											
3											0999
2							1725			4518	7009
1		9170	0701	3252	7013	1424	8425			0428	1239
CS	A	0	1	2	3	4	5	6	7	8	9

Ví dụ - Radix Sort

12	0999										
11	7009										
10	1239										
9	4518										
8	0428										
7	1725										
6	8425										
5	1424										
4	7013										
3	3252										
2	0701										
1	9170										
CS	A	0	1	2	3	4	5	6	7	8	9

101

Ví dụ - Radix Sort

12											
11											
10											
9											
8											
7											
6											
5											
4				0428							
3				1725							
2		7009	4518	8425							
1		0701	7013	1424	1239		3252		9170		0999
CS	A	0	1	2	3	4	5	6	7	8	9

102

Ví dụ - Radix Sort

12	0999										
11	9170										
10	3252										
9	1239										
8	0428										
7	1725										
6	8425										
5	1424										
4	4518										
3	7013										
2	7009										
1	0701										
CS	A	0	1	2	3	4	5	6	7	8	9

103

Ví dụ - Radix Sort

12											
11											
10											
9											
8											
7											
6											
5											
4											
3						0428					
2		7013		3252		8425			1725		
1		7009	9170	1239		1424	4518		0701		0999
CS	A	0	1	2	3	4	5	6	7	8	9

104

Ví dụ - Radix Sort

12	0999										
11	1725										
10	0701										
9	4518										
8	0428										
7	8425										
6	1424										
5	3252										
4	1239										
3	9170										
2	7013										
1	7009										
CS	A	0	1	2	3	4	5	6	7	8	9

105

Ví dụ - Radix Sort

12											
11											
10											
9											
8											
7											
6											
5											
4											
3		0999	1725								
2		0701	1424						7013		
1		0428	1239		3252	4518			7009	8425	9170
CS	A	0	1	2	3	4	5	6	7	8	9

106

Ví dụ - Radix Sort

12	9170										
11	8425										
10	7013										
9	7009										
8	4518										
7	3252										
6	1725										
5	1424										
4	1239										
3	0999										
2	0701										
1	0428										
CS	A	0	1	2	3	4	5	6	7	8	9

107

CHƯƠNG IV
DANH SÁCH LIÊN KẾT

108

Nội dung chính

1. Tổng quan

Biến tĩnh:

- Là biến được khai báo tường minh, có tên gọi, tồn tại trong phạm vi khai báo.
- Biến tĩnh có kích thước không đổi => không tận dụng hiệu quả bộ nhớ.

Giới thiệu tổng quan

Biến động:

- Là biến không được khai báo tường minh, không có tên gọi, có thể cấp khi cần, giải phóng khi sử dụng xong.
- Biến động linh động về kích thước => tận dụng hiệu quả bộ nhớ.
- Để thao tác với biến động, ta lưu địa chỉ của nó bằng biến con trỏ => truy xuất biến động thông qua biến con trỏ.

Giới thiệu tổng quan

Cấu trúc tự trỏ:

Là kiểu cấu trúc mà bên trong nó có 1 thành phần con trỏ trỏ tới 1 biến dạng cấu trúc đang được định nghĩa.

Ví Dụ:

```
struct Node{
    int Data;
    Node *Next;// Next tự trỏ đến cấu trúc chứa nó
};
```

Giới thiệu tổng quan

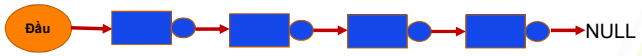
Cấu trúc dữ liệu kiểu danh sách:

Là một kiểu dữ liệu có dạng tuyến tính bao gồm các phần tử có cùng kiểu dữ liệu:

- ***Mảng***: *Liên kết ngầm.*
- ***Danh sách liên kết***: *Liên kết tường minh.*

2 Danh sách liên kết đơn (Linked list)

- Danh sách liên kết có thể được biểu diễn như là một chuỗi các nút (node). Mỗi nút sẽ trỏ tới nút kế tiếp



- Mỗi phần tử trong danh sách liên kết đơn là một cấu trúc có hai thành phần:
  - ***Thành phần dữ liệu***: Lưu trữ thông tin về bản thân phần tử;
  - ***Thành phần liên kết***: Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.

Cài đặt và khởi tạo danh sách

- **Cài đặt danh sách**

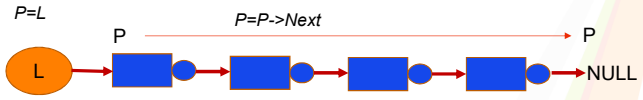
```
typedef  int item;
struct Node{
  item Data;  //thành phần dữ liệu
  Node *Next; //thành phần liên kết
};
typedef  Node *List;
```

- **Khởi tạo danh sách rỗng**

```
void InitList(List &L) {
       L=NULL;
}
```


Duyệt danh sách

- **Duyệt danh sách**
  - Tạo 1 con trỏ P trỏ vào đầu danh sách.
  - Duyệt theo danh sách bằng cách gán P=P->Next cho đến khi P=NULL



Duyệt danh sách

- Hiển thị danh sách

```
void PrintList(List L){
    Node *P = L;
    cout <<"\n[ " ;
    while(P!=NULL){//Duyệt từ đầu đến cuối d/s
         cout<<P->Data <<", ";//In thành phần dữ liệu
         P = P->Next;
    }
    cout<<" ] ";
}
```


Duyệt danh sách

- Tính độ dài danh sách

```
int ListLen (List L) {
    Node *P=L;
    int i=0; //biến đếm
    while (P!=NULL) { //duyệt đến cuối d/s
        i++; //đếm số Node
        P=P->Next; //cho P trỏ đến Node tiếp theo
    }
    return i; //trả về số Node của L
}
```


Duyệt danh sách

- Tìm vị trí phần tử có giá trị x trong danh sách

```
int Position(List L, item x) {
    Node *P=L; int i=1;
    while (P!=NULL && P->Data!=x) {
        P=P->Next;
        i++;
    }
    if (P!=NULL)
         return i; //trả về vị tri tìm thấy
    else
         return 0; //trả về 0 nếu không tìm thấy
}
```


Tạo Node

- Tạo 1 Node chứa dữ liệu x

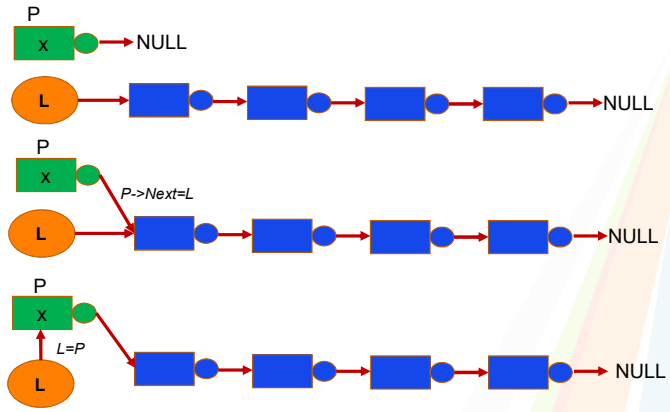
```
Node *MakeNode (item x) {
    Node *P=new Node; //Cấp phát vùng nhớ cho P
    P->Next=NULL; //Cho Next trỏ đến NULL
    P->Data=x; //Ghi dữ liệu vào Data
    return P;
}
```

Tạo Node

➢ **Thêm 1 nút vào đầu danh sách**

```
void InsFirst (List &L, item x) {
     Node *P=MakeNode(x);//tạo Node P chứa dữ liệu x
     P->Next=L;//cho P trỏ đến Node mà L đang trỏ
     L=P;//cho L trỏ về P
}
```


Thêm 1 nút vào đầu danh sách

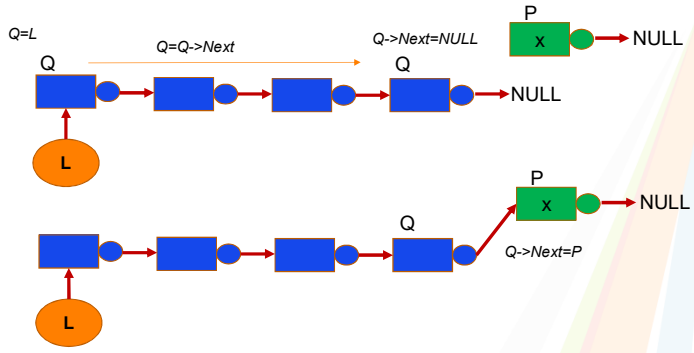


Các thao tác thêm vào danh sách

➢ **Thêm 1 nút vào vị trí cuối danh sách**

```
void InsLast(List &L, item x){
    Node *P = MakeNode(x);//tạo 1 Node P
    if (L!=NULL) { //trường hợp d/s không rỗng
        Node *Q=L; //tạo Q để duyệt d/s
        while (Q->Next!=NULL)
            Q=Q->Next; //đến node cuối
        Q->Next=P; //cho node cuối trỏ về P
     }
     else //trường hợp d/s rỗng
        L=P;
}
```


Thêm 1 nút vào cuối danh sách

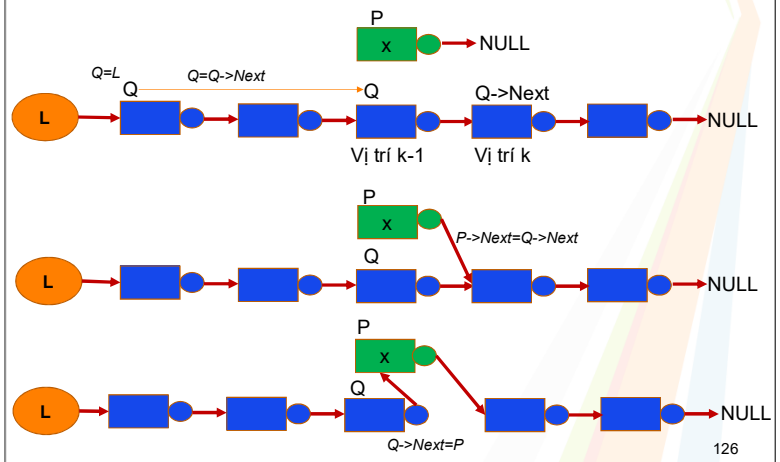


Các thao tác thêm vào danh sách

➢ **Thêm 1 nút vào vị trí k trong danh sách**

```
int InsPos(List &L, item x, int k) {
    Node *P, *Q=L; int i=1;
    P = MakeNode(x); //tao 1 Node P
    if (k<=1) InsFirst(L, x);//chèn vào vị trí đầu tiên
    else if (k>ListLen(L)) InsLast(L, x); //the7m cuối
    else {
          while (i!=k-1) {
              i++; Q=Q->Next;
          }//đến node k-1
          P->Next=Q->Next; Q->Next=P;
    }
    return 1;
}
```


Thêm 1 nút vào vị trí k

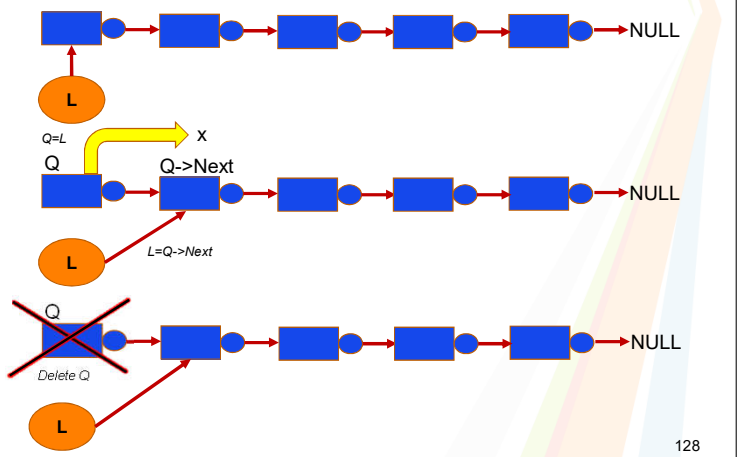


Các thao tác xóa trên danh sách

➢ **Xóa nút ở đầu danh sách**

```
int DelFirst (List &L, item  &x) {
    if (L==NULL)  return 0; //Xóa thất bại
    Node *Q=L; //Cho Q trỏ đến Node đầu trong d/s
    x=L->Data; //Lấy dữ liệu để dùng (nếu cần)
    L=L->Next; //Cho L trỏ đến Node thứ 2 trong d/s
    delete Q; //Giải phóng vùng nhớ
    return 1; //Xóa thành công
}
```


Xóa nút đầu trên danh sách

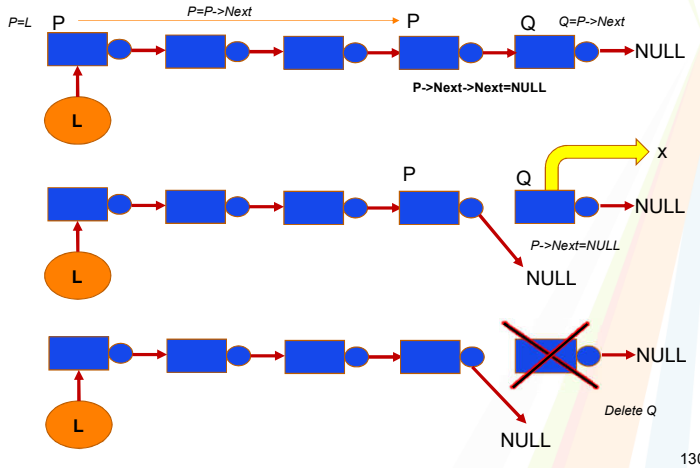


Các thao tác xóa trên danh sách

➢ **Xóa nút cuối danh sách**

```
int DelLast (List &L, item &x) {
    if (L==NULL)  return 0;//Xóa thất bại
    if (L->Next==NULL) {
       DelFirst(L, x); return 1;//Xóa thành công
    }
    Node *P=L, *Q;
    while (P->Next->Next!=NULL) P=P->Next;
    Q=P->Next; x=Q->Data; P->Next= NULL;
    delete Q; //Giải phóng vùng nhớ
    return 1; //Xóa thành công
}
```


Xóa nút cuối trên danh sách

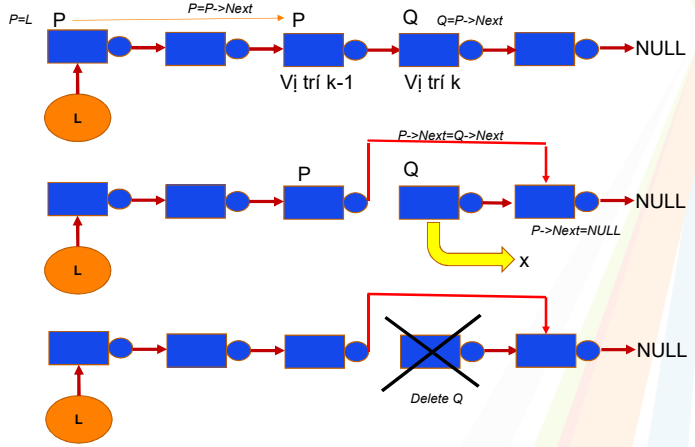


Các thao tác xóa trên danh sách

➢ **Xóa nút ở vị trí k**

```
int DelPos (List &L, item &x, int k) {
    if (k<1 || k>ListLen(L)) return 0; //Xóa thất bại
    if (k==1) DelFirst(L,x);
    else {
          Node *P=L, *Q;  int i=1;
          while (i!=k-1) { P=P->Next; i++; }
          Q=P->Next; P->Next=Q->Next;
      x=Q->Data; delete Q;
    }
    return 1; //Xóa thành công
}
```


Xóa nút tại vị trí k



Các thao tác xóa trên danh sách

➢ **Xóa nút có giá trị x trên danh sách**

```
int DelEqual(List &L, item x) {
   if (L==NULL) return 0;
   while (Position(L, x))
          DelPos(L, x, Position(L, x));
         //trong khi vẫn tìm thấy x thì tiếp tục xóa
   return 1;
}
```

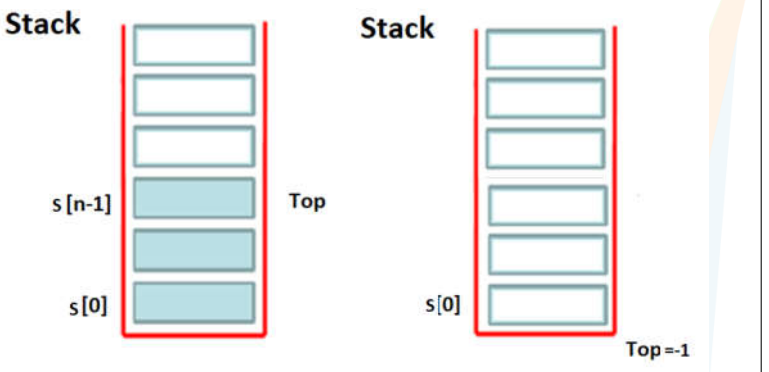

3. Ngăn xếp

- **Stack** (ngăn xếp): Là một danh sách mà việc thêm vào hoặc lấy ra một phần tử theo nguyên tắc **LIFO** (Last In First Out – Vào sau ra trước)
- **Các thao tác chính trên ngăn xếp:**
  - Tạo stack (Create).
  - Thêm đối tượng vào stack (Push).
  - Lấy đối tượng từ stack (Pop).
  - Kiểm tra stack rỗng hay không? (IsEmpty)
  - Kiểm tra stack đã đầy chưa? (IsFull)

Cài đặt stack bằng mảng một chiều

- Để cài đặt ngăn xếp bằng mảng, ta sử dụng mảng một chiều S để biểu biễn ngăn xếp.
- Thiết lập phần tử đầu tiên của mảng S[0] làm đáy ngăn xếp.Các phần tử tiếp theo được đưa vào ngăn xếp sẽ lần lượt được lưu tại các vị trí S[1], S[2],…
- Nếu hiện tại ngăn xếp có n phần tử thì S]n-1] sẽ là phần tử mới nhất được đưa vào ngăn xếp (đỉnh).
- Để lưu trữ đỉnh hiên tại của ngăn xếp, ta sử dụng một biến top lưu chỉ số của đỉnh (ở đây top=n-1)
- Khi ngăn xếp chưa có phần tử nào ta quy ước top = -1

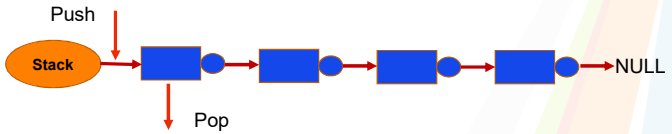
Cài đặt stack bằng mảng một chiều



Cài đặt stack bằng danh sách đơn

Theo tính chất của danh sách đơn, việc bổ sung và loại bỏ 1 phần tử thực hiện đơn giản và nhanh nhất khi phần tử đó nằm đầu danh sách. Do vậy, ta sẽ chọn cách:

- *Bổ sung 1 phần tử mới vào stack ta thêm nó vào đầu danh sách.*
- *Lấy 1 phần tử ra khỏi ngăn xếp ta lấy giá trị nút đầu tiên và loại nó ra khỏi danh sách.*



Một số ứng dụng của ngăn xếp

- Đảo ngược xâu kí tự.
- Tính giá trị biểu thức dạng hậu tố (postfix).
- Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix).
- Khử đệ quy lui
- …

4. Hàng đợi

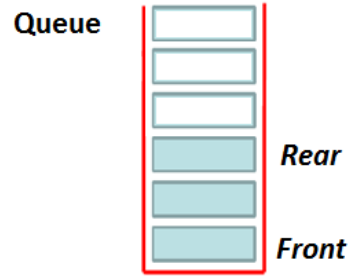
- **Queue** (hàng đợi): Là danh sách làm việc theo cơ chế **FIFO** (First In First Out), nghĩa là thêm hay lấy ra 1 đối tượng thực hiện theo cơ chế "vào trước ra trước".
- **Các thao tác chính:**
  - Tạo 1 hàng đợi (Create).
  - Thêm đối tượng vào cuối hàng đợi (Put).
  - Lấy đối tượng từ đầu hàng đợi (Get).
  - Kiểm tra hàng đợi có rỗng hay Không? (IsEmpty)
  - Kiểm tra hang đợi đầy chưa? (IsFull)
  - Dồn hang đợi (Pack)

Cài đặt Queue bằng mảng 1 chiều

- Sử dụng mảng 1 chiều với 2 con trỏ Front và Rear

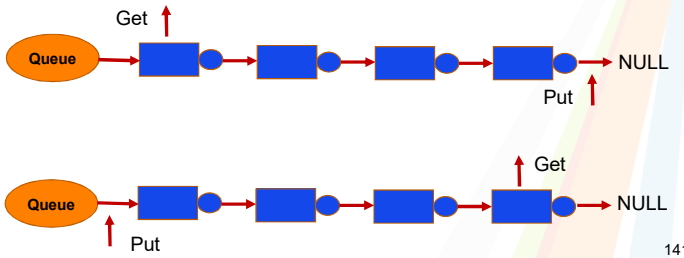
  Front: Chỉ số đầu

  Rear: Chỉ số cuối



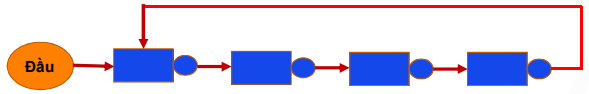
Cài đặt Queue bằng danh sách đơn

- *Để cài đặt hàng đợi bằng danh sách liên kết, ta sẽ sử dụng 1 danh sách liên kết đơn.*
- *Các thao tác thêm vào và lấy ra sẽ thực hiện bằng cách thêm vào cuối và lấy ở đầu của danh sách (hay thêm vào đầu lấy ở cuối danh sách)*



5. Một số dạng danh sách khác

- **Danh sách liên kết vòng (Circular Linked List):** *Nút cuối liên kết với nút đầu tiên*



- **Danh sách liên kết kép (Doubly Linked List):** Mỗi nút sẽ liên kết với nút trước và nút sau nó