LỜI NÓI ĐẦU

Giáo trình này sẽ trình bày các cấu trúc dữ liệu cơ bản và thuật giải. Các kiến thức về cấu trúc dữ liệu và thuật giải đóng vai trò quan trọng trong việc đào tạo cử nhân toán – tin học, công nghệ thông tin. Sách này được hình thành trên cơ sở các bài giảng về cấu trúc dữ liệu và thuật giải mà tác giả đã giảng dạy nhiều năm tại khoa Toán – Tin học Đại học Khoa Học Tự Nhiên, Đại học Quốc Gia thành phố Hồ Chí Minh và khoa Công Nghệ Thông Tin các trường bạn. Giá trình được viết chủ yếu để làm tài liệu tham khảo cho sinh viên các ngành Toán – Tin học và Công nghệ Thông tin, nhưng nó cũng rất bổ ích cho các độc giả khác cần có hiểu biết đầy đủ hơn về cấu trúc dữ liệu và thuật giải.

Tác giả mô tả và cài đặt các cấu trúc dữ liệu và thuật giải trong ngôn ngữ C, vì C là ngôn ngữ cơ bản giảng dạy trong trường đại học, được nhiều người biết đến và là ngôn ngữ được sử dụng nhiều để trình bày thuật toán trong các tài liệu khác nhau. Hình ảnh và các đoạn mã có thể lấy từ http://www.math.hcmus.edu.vn/~ptbao/DataStructure/Book/

Nội dung giáo trình cấu trúc dữ liệu và thuật giải được tổ chức gồm 04 chương:

- Chương 1: Tổng quan
  - Khái niệm thuật giải
  - Phân tích thuật giải
  - Trừu tượng hóa dữ liệu
- Chương 2: Tìm kiếm và sắp xếp
  - Tìm kiếm
  - Sắp xếp
- Chương 3: Danh sách liên kết
  - Cấu trúc mảng
  - Danh sách liên kết
  - Danh sách liên kết đơn và những thao tác cơ bản
  - Danh sách liên kết đôi và những thao tác cơ bản
  - Ngăn xếp
  - Hàng đợi
  - Ứng dụng mở rộng của danh sách liên kết
- Chương 4: Cây – Cây nhị phân tìm kiếm
  - Cây tổng quát
  - Cây nhị phân
  - Cây nhị phân tìm kiếm
  - Cây AVL

Mặc dù đã rất cố gắng nhiều trong quá trình biên soạn giáo trình, song không khỏi còn nhiều thiếu sót và hạn chế. Rất mong nhận được sự đóng góp ý kiến quý báu của quý thầy cô, các bạn sinh viên và các bạn đọc để giáo trình ngày một hoàn thiện hơn.

Tác giả
Phạm Thế Bảo.

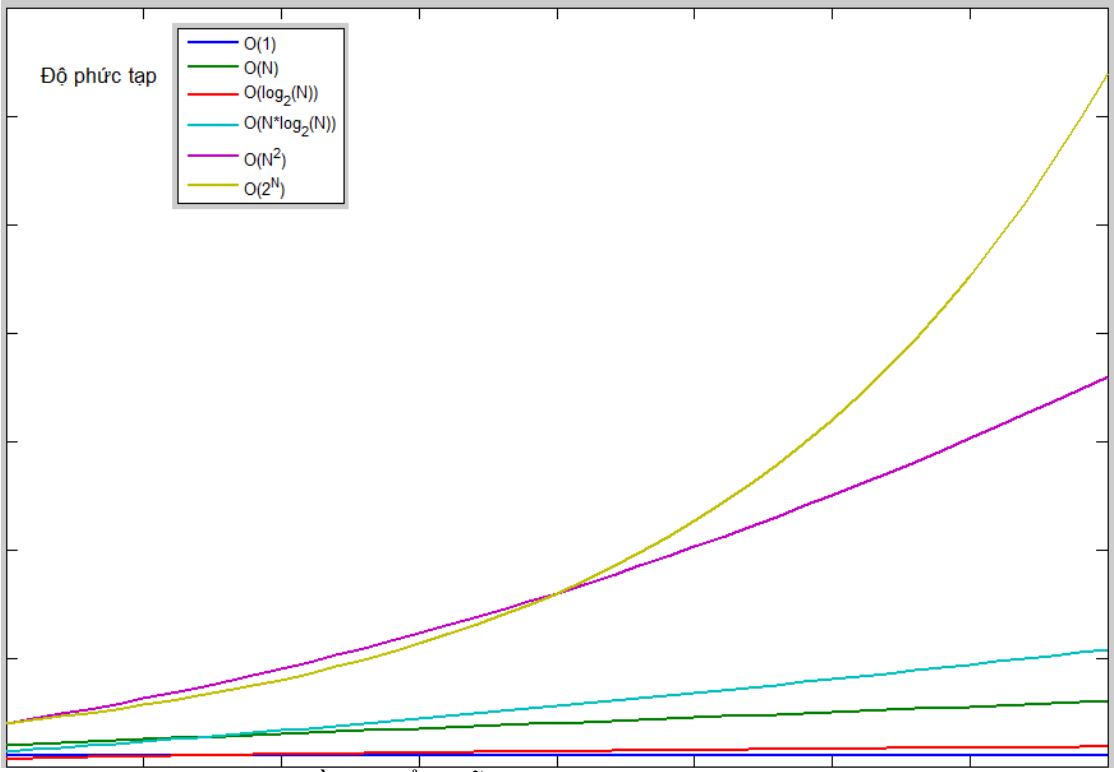
CHƯƠNG 1: TỔNG QUAN

I. Khái niệm thuật giải

II. Phân tích thuật giải

Bảng 1.1 Phân loại độ phức tạp.

Dạng $\mathcal{O}$	Tên Phân loại	
$\mathcal{O}(1)$	Hằng	
$\mathcal{O}\log_2 N$	logarit	
$\mathcal{O}(\sqrt{N})$	Căn thức	
$\mathcal{O}(\sqrt[3]{N})$		
…		
$\mathcal{O}(\sqrt[m]{N})$		
$\mathcal{O}(N)$	Tuyến tính	Đa thức
$\mathcal{O}(N^2)$	Bình phương	
$\mathcal{O}(N^3)$	Bậc ba	
…		
$\mathcal{O}(N^m)$	Đa thức	
$\mathcal{O}(c^N)$ với $c > 1$	Mũ	Độ phức tạp lớn
$\mathcal{O}(N!)$	Giai thừa	



Hình 1.1: Đồ thị biểu diễn độ phức tạp của thuật toán theo N.

III. Trừu tượng hóa dữ liệu

CHƯƠNG 2: TÌM KIẾM VÀ SẮP XẾP

I. Tìm kiếm

II. Sắp xếp

i. Thuật toán Interchange Sort (đổi chỗ trực tiếp)

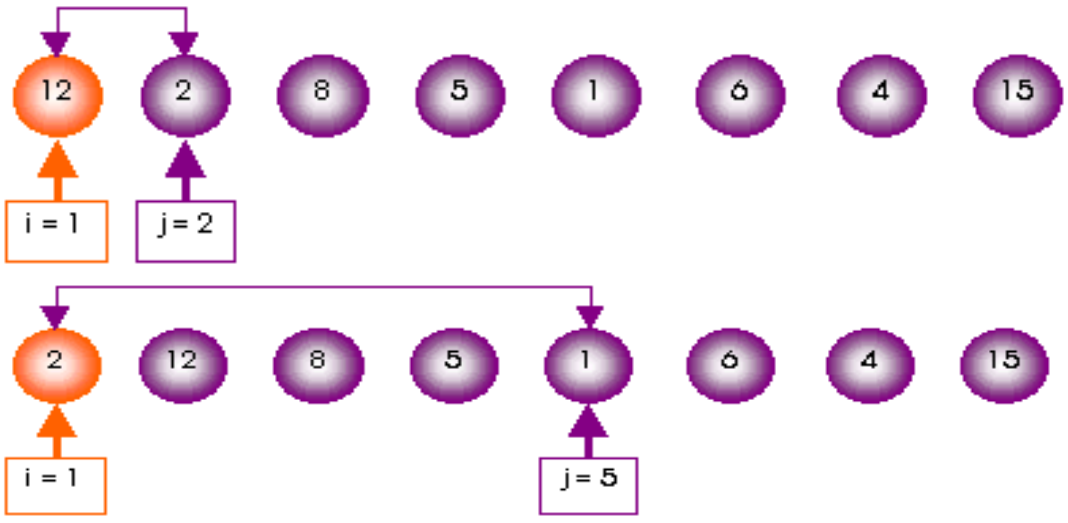
Thuật toán

Thuật toán InterchangeSort(A,n)
Input: Mảng A có n phần tử
Output: Mảng A được sắp xếp
for(i ← 0 to n–2) do
for(j ← i+1 to n–1) do
if(A[i]>A[j]) then A[i] ↔ A[j]
return;

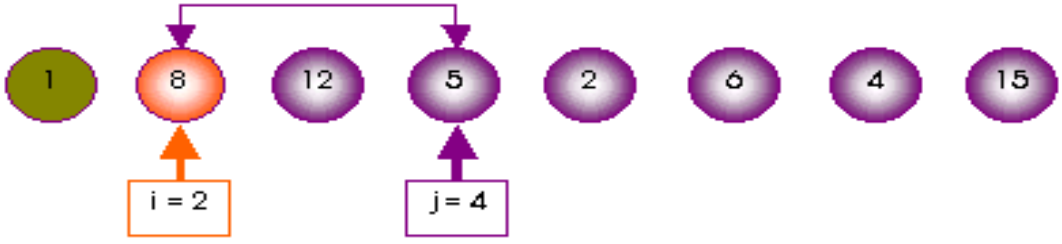
- Ví dụ

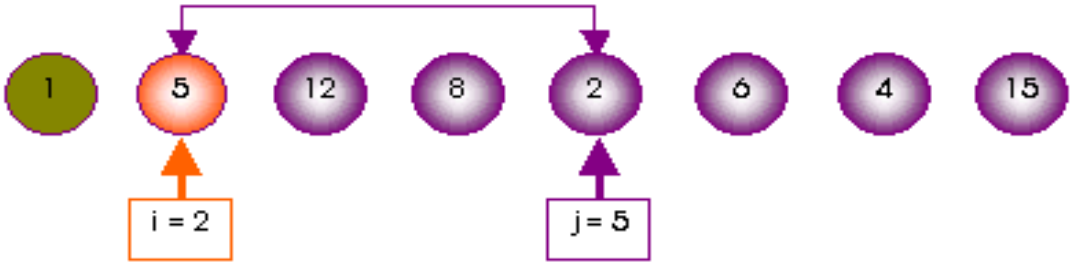
Cho dãy số 12 2 8 5 1 6 4 15

Bước 1:



Bước 2:





Và các bước tiếp theo cho đến bước thứ n–1.

ii. Thuật toán Bubble Sort (nổi bọt)
Thuật toán

```
Thuật toán BubbleSort(A,n)                          // nhẹ nổi lên
Input: Mảng A có n phần tử
Output: Mảng A được sắp xếp
for(i ← 0 to n–2) do
        for(j ← n–1 to i+1) do
                if(A[j–1] > A[j]) then A[j–1] ↔ A[j]

return;
```

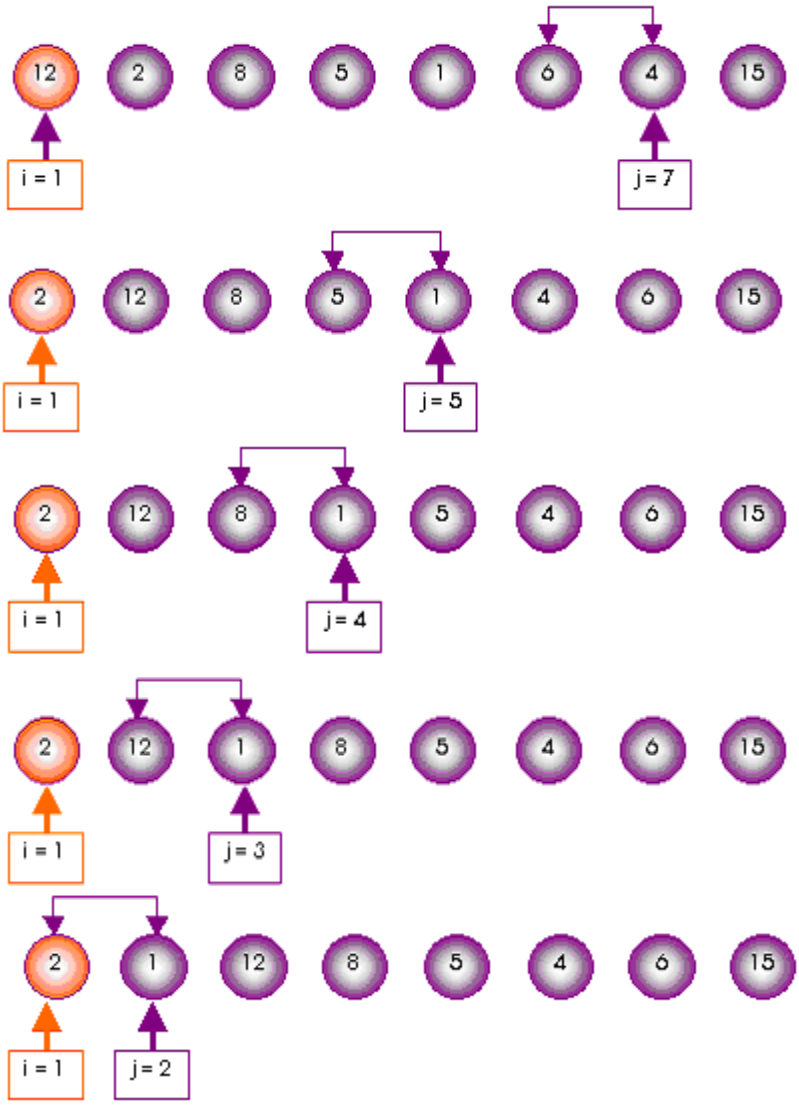
```
Thuật toán BubbleSort1(A,n)                         // nặng chìm xuống
    Input: Mảng A có n phần tử
    Output: Mảng A được sắp xếp
    for(i ← n–1 to 1) do
            for(j ← 0 to i–1) do
                if(A[j] > A[j+1]) then A[j] ↔ A[j+1]

    return;
```

```
Thuật toán BubbleSort2(A,n)                         // kết hợp vừa nặng và nhẹ
    Input: Mảng A có n phần tử
    Output: Mảng A được sắp xếp
    up ← 0; down ← n–1;
    while (up < down) do
            for (j ← up to down–1) do
                    if(A[j] > A[j+1]) then A[j] ↔ A[j+1]
            down ← down –1;
            for(j ← down to up+1) do
                    if(A[j–1] > A[j]) then A[j–1] ↔ A[j]
            up ← up + 1;
    return;
```

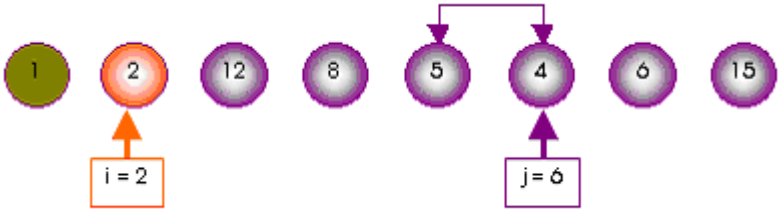
- Ví dụ

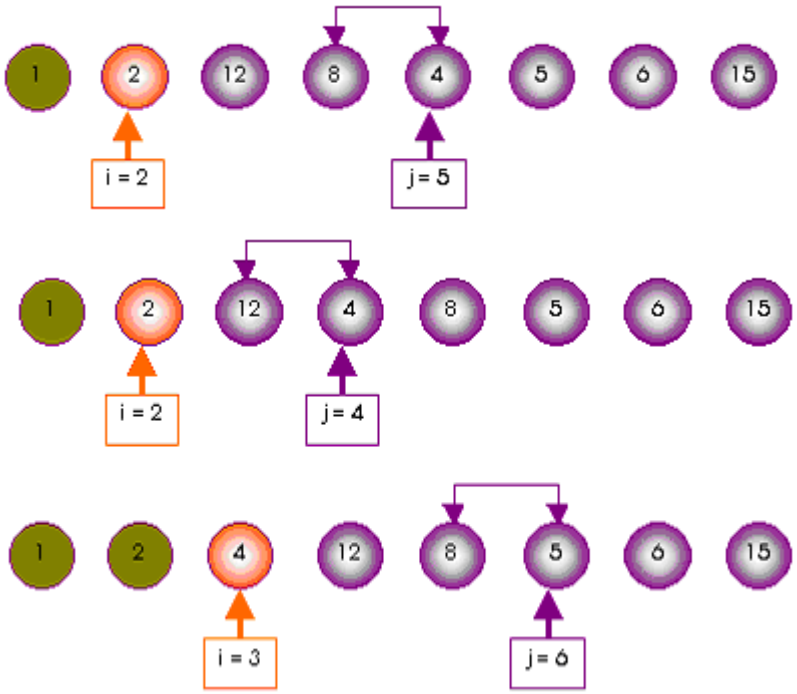
Cho dãy số 12 2 8 5 1 6 4 15

Bước 1:



Bước 2:





Và các bước tiếp theo cho đến bước thứ n–1.

iii. Shaker Sort

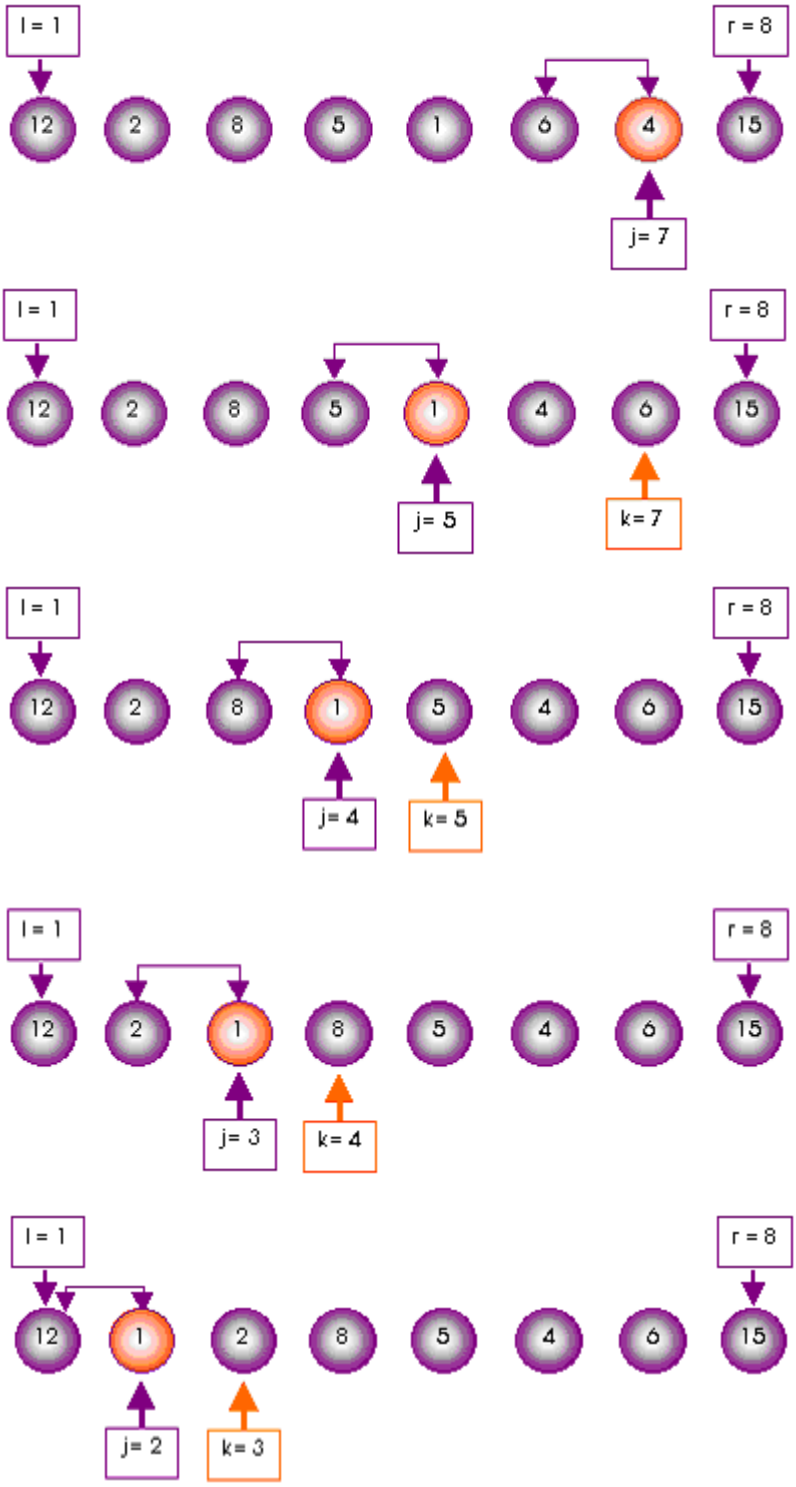
Thuật toán

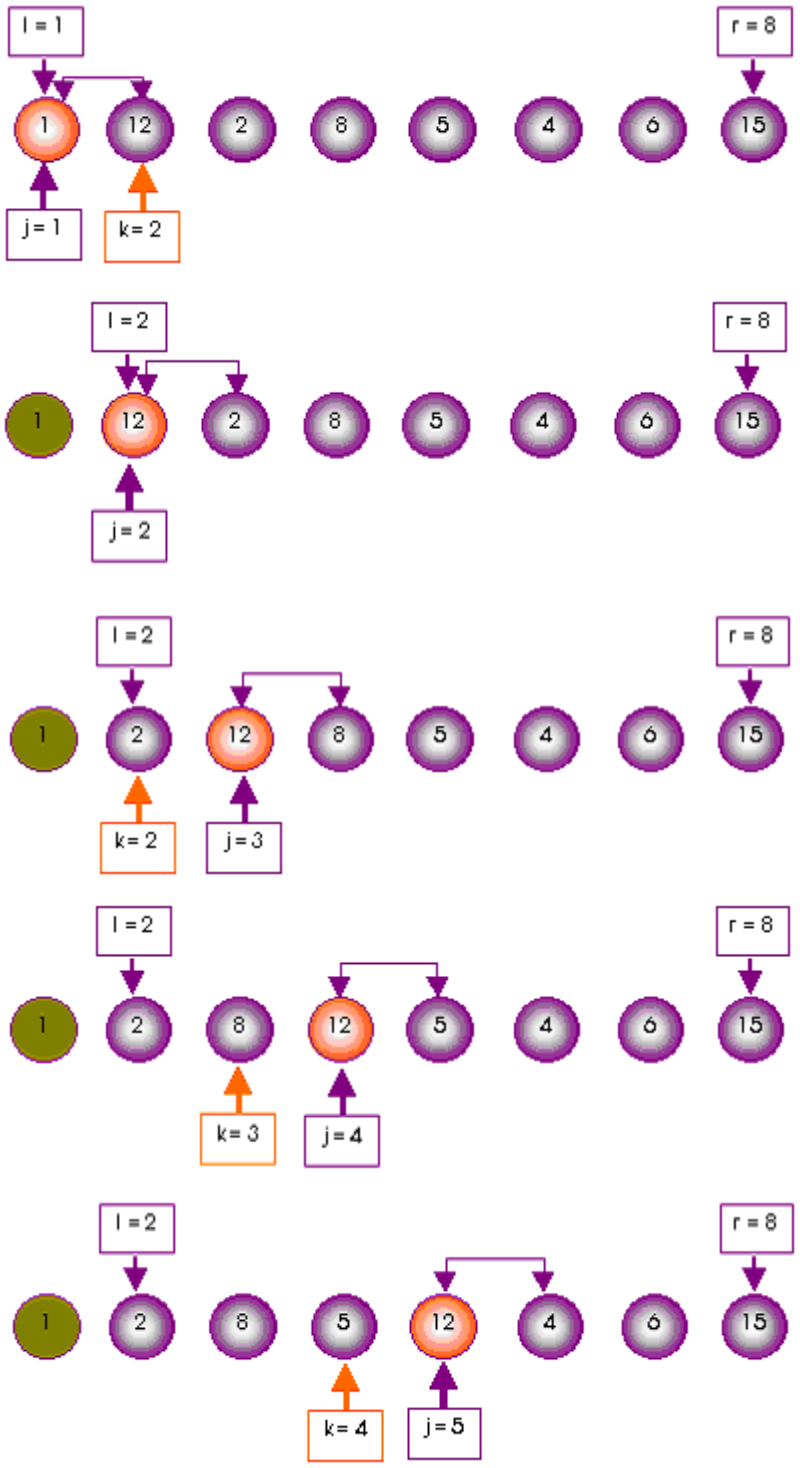
```
Thuật toán ShakerSort(A,n)
    Input: Mảng A có n phần tử
    Output: Mảng A được sắp xếp
    up ← 0; down ← n–1;hv ← 0;
    while (up < down) do
                for (j ← up to down–1) do
                        if(A[j] > A[j+1]) then
                                A[j] ↔ A[j+1];
                                hv ← j;
                down ← hv;
                for(j ← down to up+1) do
                        if(A[j–1] > A[j]) then
                                A[j–1] ↔ A[j];
                                hv ← j;
                up ← hv;
    return;
```

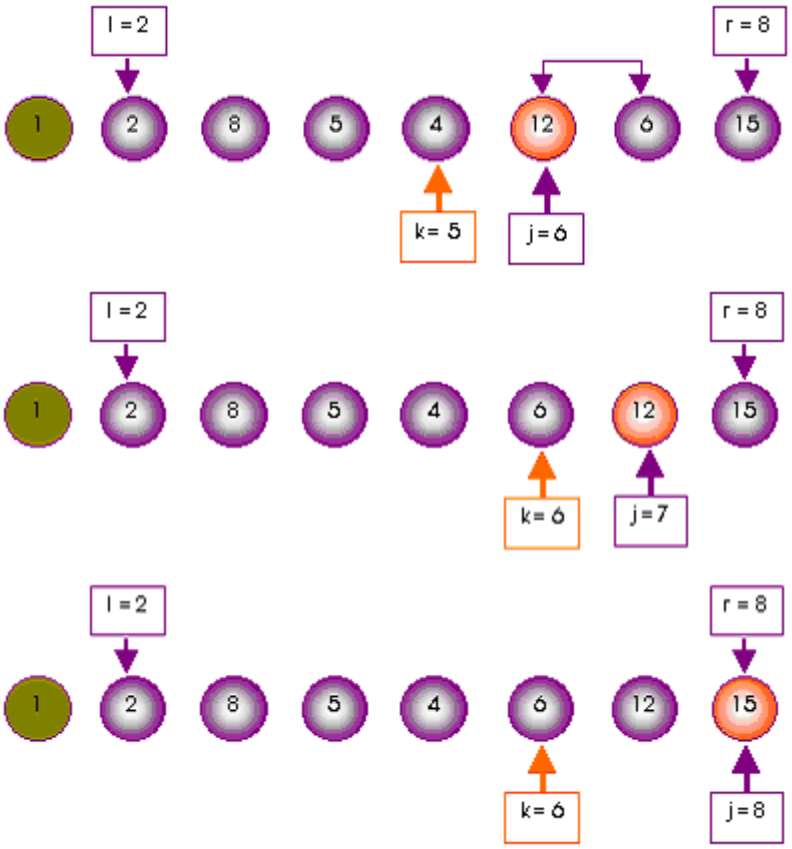
- Ví dụ

Cho dãy số 12 2 8 5 1 6 4 15

Bước 1: nhẹ nổi sau đó đến nặng chìm


l = 1
r = 8
12
2
8
5
1
6
4
15
j= 7
l = 1
r = 8
12
2
8
5
1
4
6
15
j= 5
k= 7
l = 1
r = 8
12
2
8
1
5
4
6
15
j= 4
k= 5
l = 1
r = 8
12
2
1
8
5
4
6
15
j= 3
k= 4
l = 1
r = 8
12
1
2
8
5
4
6
15
j= 2
k= 3


l = 1
r = 8
1
12
2
8
5
4
6
15
j = 1
k = 2
l = 2
r = 8
1
12
2
8
5
4
6
15
j = 2
l = 2
r = 8
1
2
12
8
5
4
6
15
k = 2
j = 3
l = 2
r = 8
1
2
8
12
5
4
6
15
k = 3
j = 4
l = 2
r = 8
1
2
8
5
12
4
6
15
k = 4
j = 5



Bước 2: thay vì bắt đầu tại vị trí có giá trị là 12 (vị trí r = 7) thì chúng ta sẽ bắt đầu tại vị trí r = 6 để cho phần từ nhẹ nổi lên. Rồi sau đó cho phần tử nặng chìm xuống.

Các bước cứ tiếp tục cho đến khi l = r thì dừng.

iv. Thuật toán Insertion Sort (chèn trực tiếp)

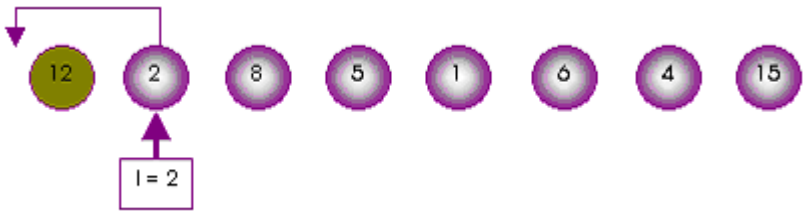
Thuật toán

```
Thuật toán InsertionSort(A,n)
    Input: Mảng A có n phần tử
    Output: Mảng A có thứ tự
    for(i ← 1 to n–1) do
                x ← A[i];
                j ← i–1;
                while (x < A[j] && j ≥ 0) do
                        A[j+1] ← A[j];
                        j ← j–1;
                A[j+1] ← x;
    return;
```

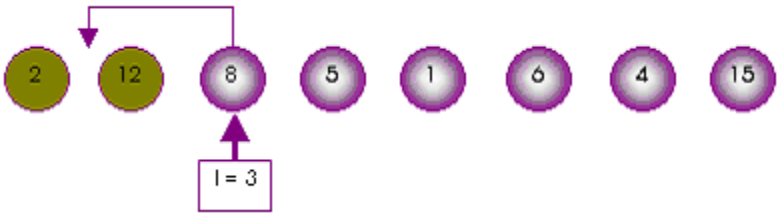
- Ví dụ

Cho dãy số 12 2 8 5 1 6 4 15

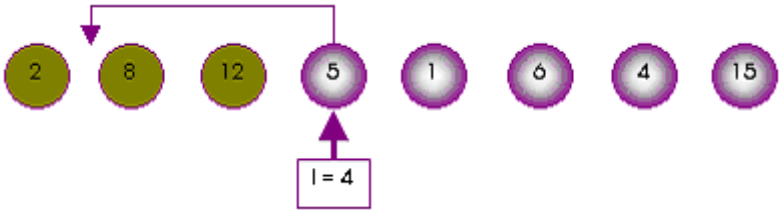
Bước 1:



Bước 2:



Bước 3:



Và các bước tiếp theo cho đến n.

v. Thuật toán Binary Insertion Sort (chèn nhị phân)
Thuật toán

```
Thuật toán BinaryInsertionSort(A,n)
    Input: Mảng A có n phần tử
    Output: Mảng A có thứ tự
    for(i ← 1 to n–1) do
                x ← A[i];
                k ← BinarySearchForPosition(A,i,x);
                for(j ←i to k+1) do
                        A[j] ← A[j–1];
                A[k] ← x;
    return;
```

vi. Thuật toán Shell Sort
Thuật toán

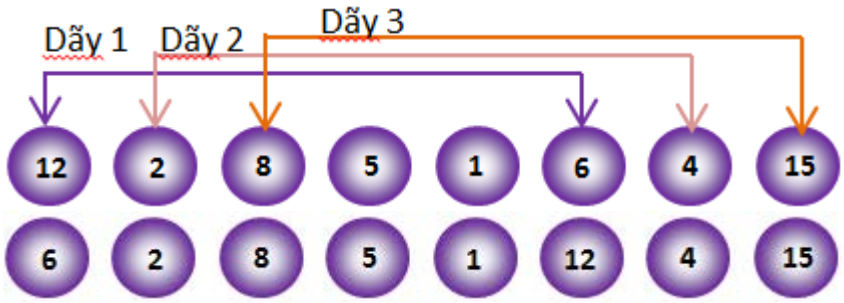
```
Thuật toán ShellSort(A,n)
    Input: Mảng A có n phần tử
    Output: Mảng A có thứ tự
    Chọn h[0], h[1], …, h[k–1] = 1;
    for(i ← 0 to k–1) do
                Phân hoạch mảng A thành t dãy có chiều dài h[i]
```

```
                    for(j ← 0 to t–1) do
                            InsertionSort(dãy con đã phân hoạch);
    return;
```

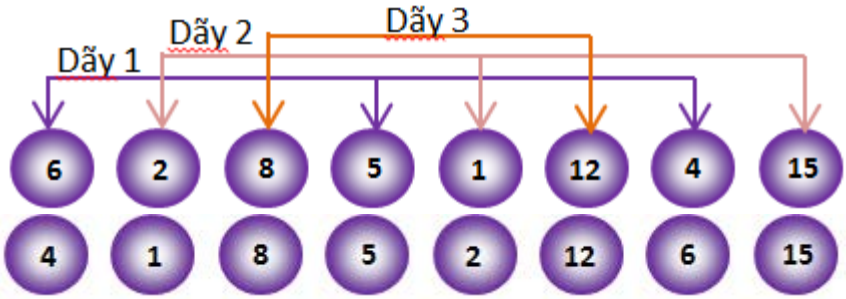
- Ví dụ

Cho dãy số 12 2 8 5 1 6 4 15
Xét các bước nhảy h = {5, 3, 1}

Bước 1: phân hoạch mảng A thành 03 dãy con có bước nhảy 5, sắp xếp từng dãy con này.



Bước 2: phân hoạch mảng A thành 03 dãy con có bước nhảy 3, sắp xếp từng dãy con này.



Bước 3: sắp xếp mảng có các phần tử: 4 1 8 5 2 12 6 15 bằng chèn trực tiếp để có mảng sắp xếp hoàn toàn.

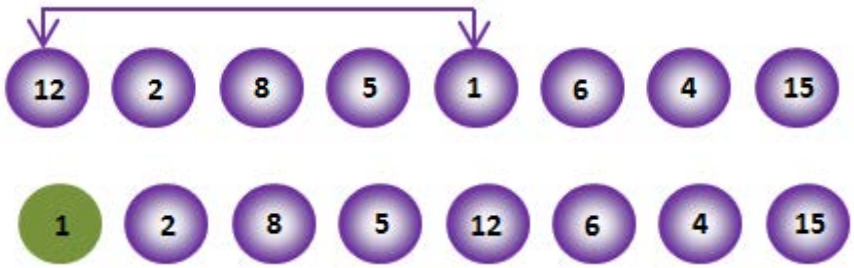
vii. Thuật toán Selection Sort (chọn trực tiếp)
Thuật toán

```
Thuật toán SelectionSort(A,n)
    Input: Mảng A có n phần tử
    Output: Mảng A có thứ tự
    for(i ← 0 to n–2) do
                k ← i;
                for(j ← i+1 to n–1) do
                        if( A[k] > A[j] ) then k← j;
                A[k] ↔ A[i];
    return;
```

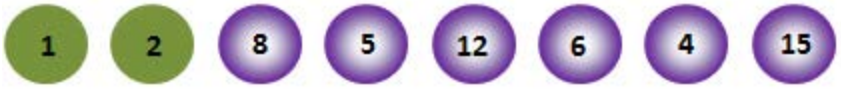
- Ví dụ

Cho dãy số 12 2 8 5 1 6 4 15

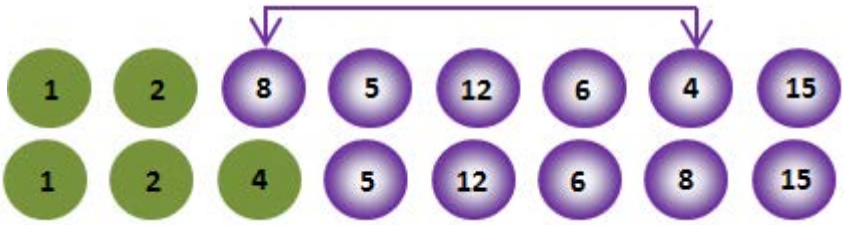
Bước 1: chọn được 1 là phần tử bé nhất, hoán đổi 1 và 12.



Bước 2: chọn được 2 là phần tử bé nhất trong dữ liệu còn lại.



Bước 3: chọn được 4 là phần tử bé nhất trong dữ liệu còn lại, hoán đổi 4 và 8.



Các bước tiếp theo cho đến cuối chúng ta được mảng sắp xếp toàn bộ.

viii. Thuật toán Heap Sort

Thuật toán

```
Thuật toán HeapSort(A,n)
        Input: Mảng A có n phần tử
        Output: Mảng A có thứ tự
        CreateHeap(A,n);
        for(k ← n–1 to 1) do
                A[0] ↔ A[k];
                InsertHeap(A,0,k–1);
        return;
```

```
Thuật toán CreateHeap(A,n)
        Input: Mảng A có n phần tử
        Output: Mảng A là heap max
        for(k ← (n+1)/2 – 1 to 0) do
                InsertHeap(A,k,n–1);
```

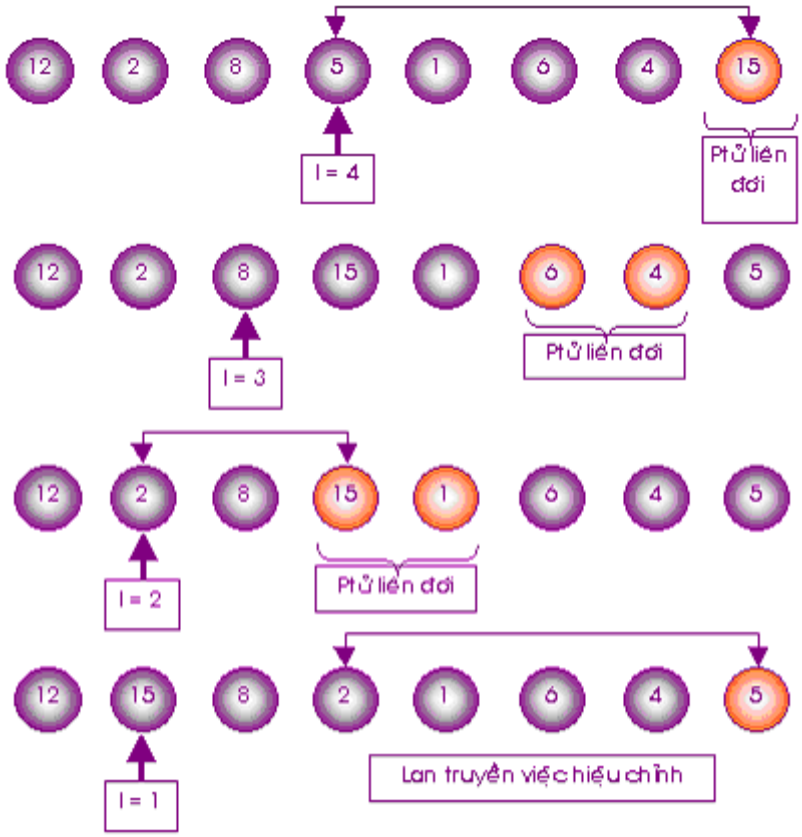
```
        return;
```

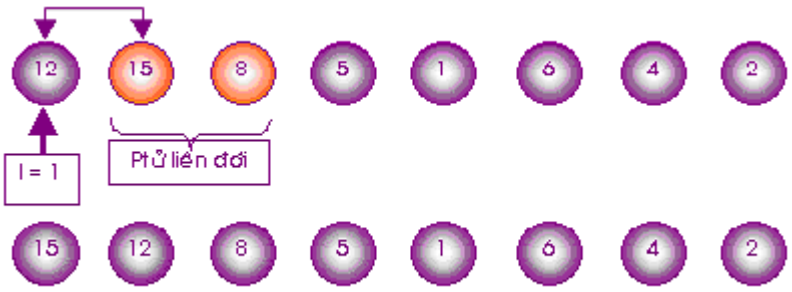
```
Thuật toán InsertHeap(A,l,r)
        Input: Mảng A chứa heap   a_{left+1}  a_{left+2}  ...  a_{right}
        Output: Mảng A chứa heap  a_{left}  a_{left+1}  ...  a_{right}
        p ← 2*left;
        if(p > right) then return;
        if(p < right) then
                if(A[p] < A[p+1]) then p ← p+1;
        if(A[left] < A[p])
                A[left] ↔ A[p];
                InsertHeap(A,p,right);
    return;
```

- Ví dụ

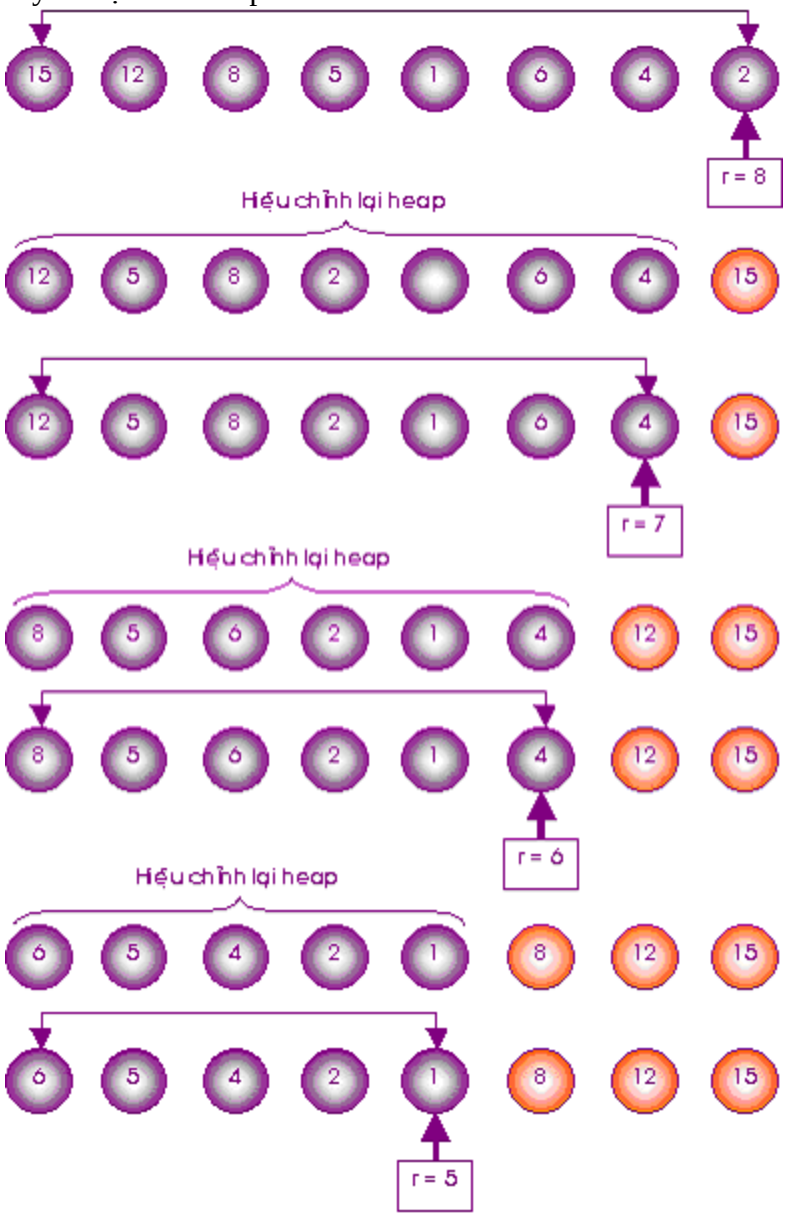
Cho dãy số 12 2 8 5 1 6 4 15

Giai đoạn 1: hiệu chỉnh dãy ban đầu thành heap max





Giai đoạn 2: Sắp xếp dãy số dựa trên heap



Các bước làm tiếp tục cho r giảm dần thì chúng ta sẽ có mảng được sắp xếp.

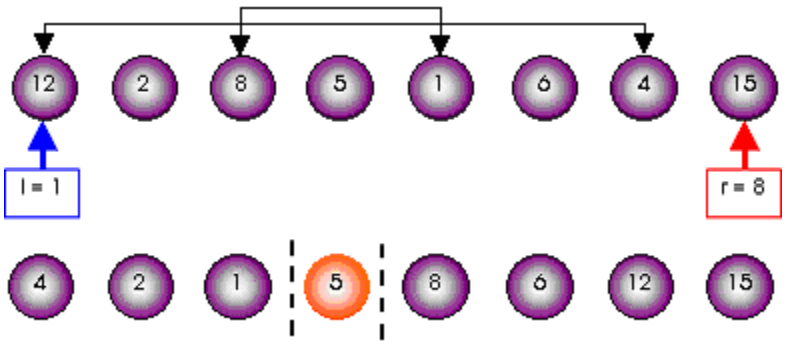
ix. Thuật toán Quick Sort

Thuật toán

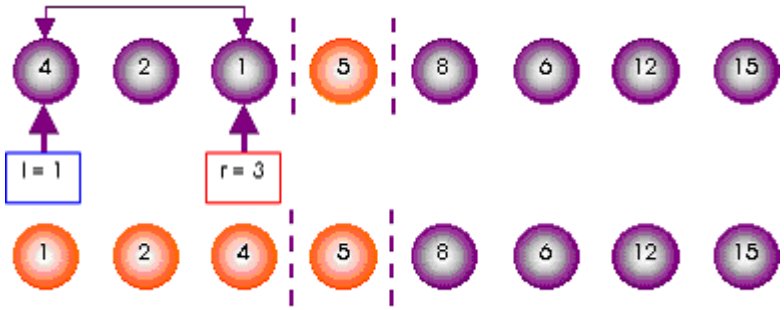
Thuật toán QuickSort(A,left,right)
Input: Mảng A có các phần tử ở vị trí tử left đến right.
Output: Mảng A được sắp xếp từ left đến right.
Chọn phần tử X;
i←left; j←right;
while (i < j) do
while (A[i] < X) do i←i + 1;
while (A[j] > X) do j←j – 1;
if (i < j) then
A[i]↔A[j];
i←i + 1;
j←j – 1;
if (j > left) then QuickSort(A,left,j);
if(right > i) then QuickSort(A,i+1,right);
return;

- Ví dụ

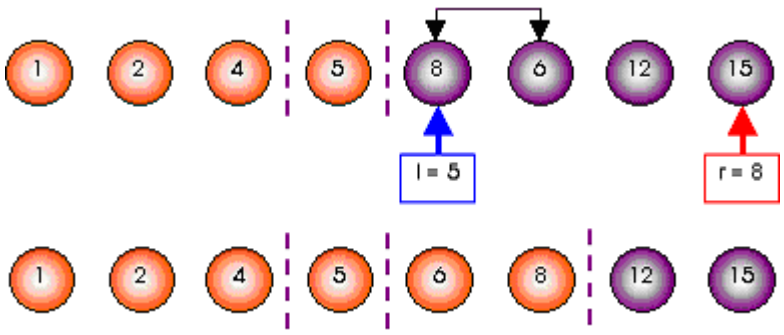
Cho dãy số 12 2 8 5 1 6 4 15
Bước 1:



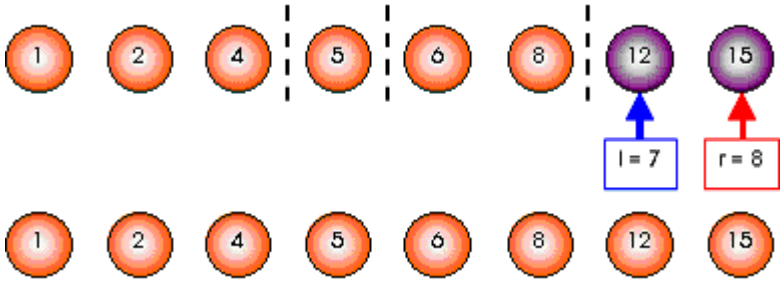
Bước 2:



Bước 3:



Bước 4:



x. Thuật toán Merge Sort

Thuật toán

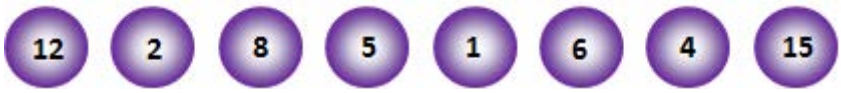
Thuật toán MergeSort(A,left,right)
Input: Mảng A có các phần tử ở vị trí tử left đến right.
Output: Mảng A được sắp xếp từ left đến right.
if(left < right) then
mid ← (left+right)/2 ;
MergeSort(A,left,mid);
MergeSort(A,mid+1,right)
Trộn A[left,mid] và A[mid+1,right] thành A[let,right];
return;

- Ví dụ

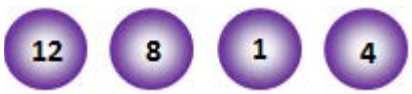
Cho dãy số 12 2 8 5 1 6 4 15

❖ Cách chia thứ nhất: tuần tự từng phần tử đưa và 02 mảng con.

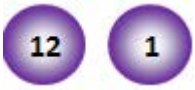
Bước 1: phân chia mảng A



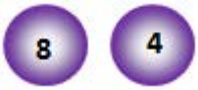
thành 02 mảng con B và C



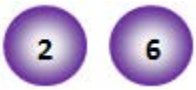
Bước 2: phân chia B thành B1



và B2.



C thành C1



và C2.

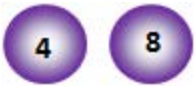
Bước 3: phân chia:

- ✓ B1 thành B11 và B12;
- ✓ B2 thành B21 và B22;
- ✓ C1 thành C11 và C12;
- ✓ C2 thành C21 và C22;

Bước 4: trộn:

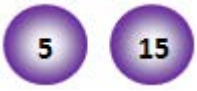
- ✓ B11 và B12 thành B1

- ✓ B21 và B22 thành B2



- ✓ C11 và C12 thành C1

- ✓ C21 và C22 thành C2



Bước 5: trộn:

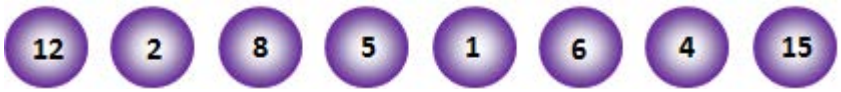
- ✓ B1 và B2 thành B

- ✓ C1 và C2 thành C

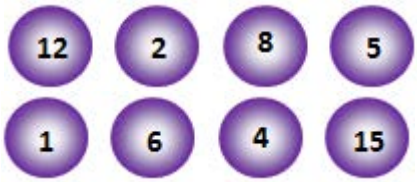
Bước 6: trộn B và C thành mảng A được sắp xếp hoàn toàn.

❖ Cách chia thứ hai: chia một nửa đầu tiên của A vào B và nửa còn lại vào C.

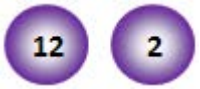
Bước 1: phân chia mảng A



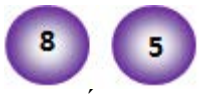
thành 02 mảng con B và C.



Bước 2: phân chia B thành B1



và B2.



Quá trình phân chia thành các mảng có một phần tử; rồi trộn lại giống như cách chia thứ nhất để có mảng sắp xếp toàn bộ.

xi. Phương pháp Radix Sort (sắp xếp theo cơ số)

Thuật toán

```
Thuật toán RadixSort(A,n,k)
    Input: Mảng A có n phần tử, có tối đa k chữ số.
    Output: Mảng A được sắp xếp.
    Khởi tạo 10 ngăn chứa B[0..9] rỗng;
    for (t ← 0 to k–1 ) do
        for (j ← 0 to n–1) do
            Thêm A[j] vào B[Digit(A[j],t)];
        for (h ← 0 to 9) do
            Lấy ngược các phần tử từ B[h] đưa vào A
    return;
```

```
Thuật toán Digit(n,k)
    Input: số nguyên n và k.
    Output: giá trị tại vị trí k của số n.
    value ← 1;
    for (i ← 0 to k–1 ) do
        value ←  value * 10;
```

```
return (n / value)% 10;
```

- Ví dụ

Bước 1: sàng theo hàng đơn vị và lấy từ các sọt ra mảng.

0	123										
1	23										
2	120										
3	421										
4	78										
5	79										
6	69										
7	218				23					218	69
8	452	120	421	452	123					78	79
CS	A	0	1	2	3	4	5	6	7	8	9

0	120										
1	421										
2	452										
3	123										
4	23										
5	78										
6	218										
7	79										
8	69										
CS	A	0	1	2	3	4	5	6	7	8	9

Bước 2: sàng theo hàng chục và lấy từ các sọt ra mảng.

0	120										
1	421										
2	452										
3	123										
4	23										
5	78			23							
6	218			123							
7	79			421					79		
8	69		218	120			452	69	78		
CS	A	0	1	2	3	4	5	6	7	8	9

0	218										
1	120										

2	421										
3	123										
4	23										
5	452										
6	69										
7	78										
8	79										
CS	A	0	1	2	3	4	5	6	7	8	9

Bước 2: sàng theo hàng trăm và lấy từ các sọt ra mảng.

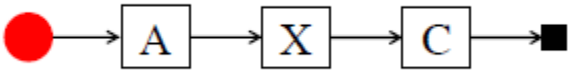
0	218										
1	120										
2	421										
3	123										
4	23										
5	452	79									
6	69	78									
7	78	69	123			452					
8	79	23	120	218		421					
CS	A	0	1	2	3	4	5	6	7	8	9

0	23										
1	69										
2	78										
3	79										
4	120										
5	123										
6	218										
7	421										
8	452										
CS	A	0	1	2	3	4	5	6	7	8	9

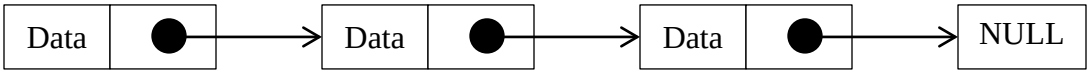
CHƯƠNG 3: DANH SÁCH LIÊN KẾT

I. Cấu trúc mảng

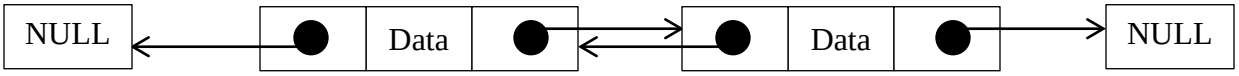
II. Danh sách liên kết



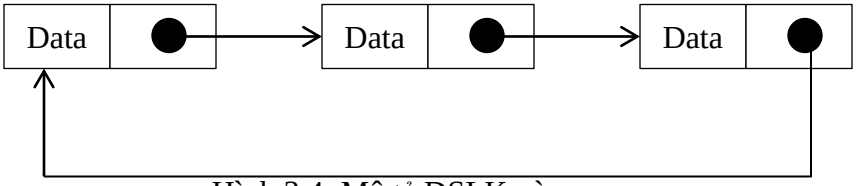
Hình 3.1: Mô tả trực quan danh sách liên kết.



Hình 3.2: Mô tả DSLK đơn.



Hình 3.3: Mô tả DSLK đôi.



Hình 3.4: Mô tả DSLK vòng.

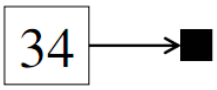
III. Danh sách liên kết đơn và những thao tác cơ bản

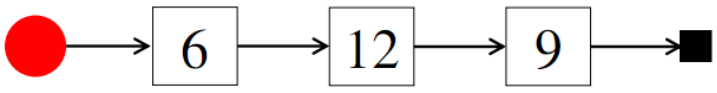
Khai báo cài đặt trong C dạng cấu trúc:

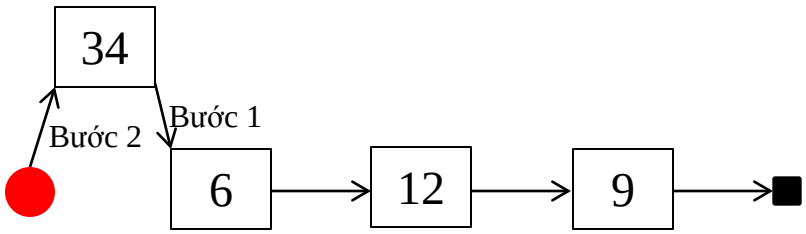
```
typedef struct tagNode{
        DataType Data;              // khai báo dữ liệu
        struct tagNode *Next;
};
typedef tagNode *Node;
typedef struct tagList{
        Node Head;    //phần tử đầu tiên.
        Node Tail;    // phần tử cuối, có thể không cần dùng
}LinkedList;
```

Khai báo cài đặt trong C dạng hướng đối tượng:

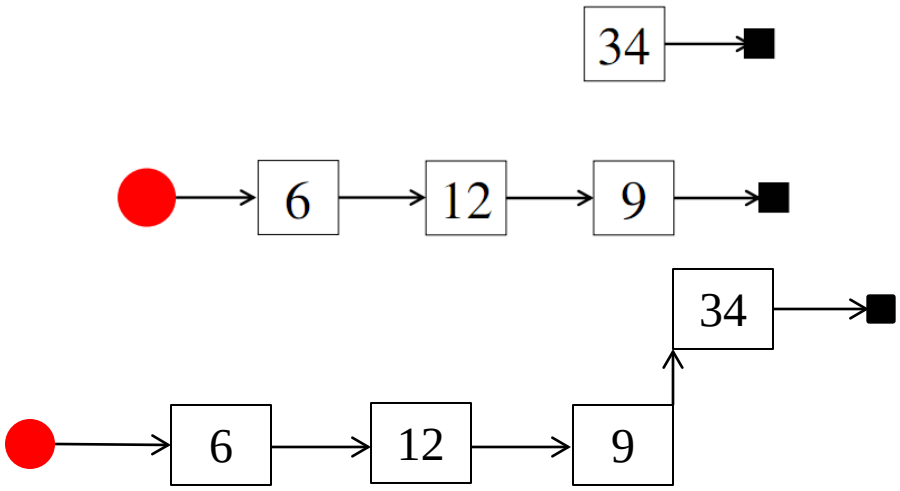
```
class Node{
friend class LinkedList;
        private:
                DataType data;
                Node *Next;
        public:
                Node(DataType d=giá trị đặc biệt ) {
        data=d;Next=NULL;}
                ~Node(){if(Next)delete Next;}

};
typedef Node *LinkedNode;

class LinkedList{
        protected:
                LinkedNode Head,Tail;
        public:
                LinkedList(){Head=Tail=NULL;}
                ~LinkedList(){if(Head)delete Head;}
                void insertNode(DataType);
                int searchNode(DataType);
                void deleteNode(DataType);
                void addTail(DataType);
                void printList();
                void addHead(DataType);
                void addAfter(DataType,DataType);
        private:
                void addTail(LinkedNode);
                void addHead(LinkedNode);
                LinkedNode createNode(DataType);
                void insertNode(LinkedNode);
                int searchNode(LinkedNode);
                int deleteNode(DataType);
                int addAfter(LinkedNode,DataType);
};
```



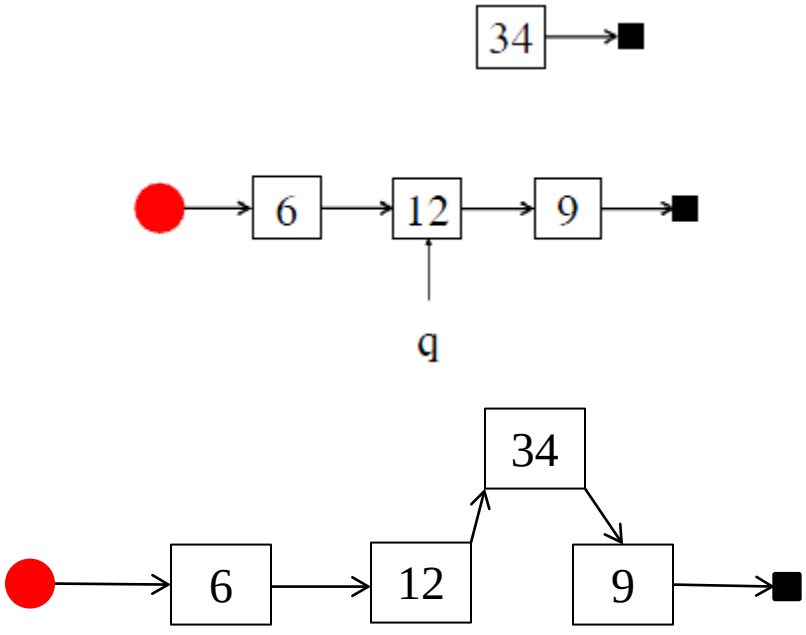




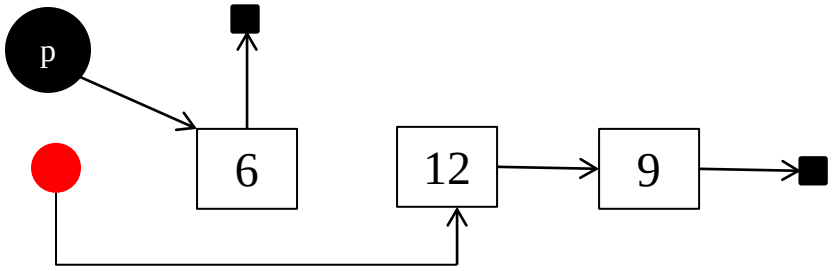
Hình 3.5. Ví dụ thêm một phần tử vào đầu DSLK đơn.



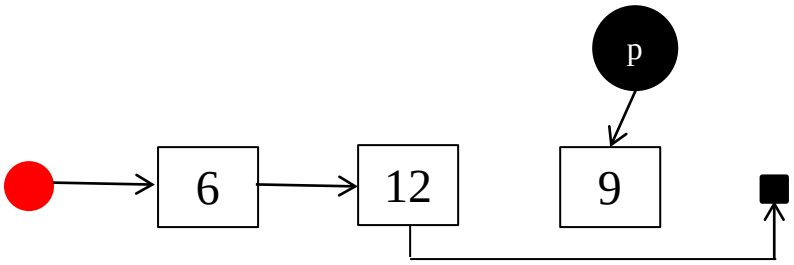
Hình 3.6. Ví dụ thêm một phần tử vào cuối DSLK đơn.



Hình 3.7. Ví dụ thêm một phần tử vào sau phần tử q trong DSLK đơn.



Hình 3.8. Ví dụ hủy một phần tử ở đầu DSLK đơn.



Hình 3.9. Ví dụ hủy một phần tử ở cuối DSLK đơn.

IV. Danh sách liên kết đôi và những thao tác cơ bản

Khai báo cài đặt trong C dạng cấu trúc:

Hai con trỏ trỏ đến hai phần tử trước và sau (trường hợp 1).	Hai con trỏ trỏ đến hai phần tử bên trái và bên phải (trường hợp 2).

typedef struct tagNode{
DataType Data;
tagNode *Next;
tagNode *Prev;
};
	typedef struct tagNode{
	DataType Data;
	tagNode *Left;
	tagNode *Right;
	};

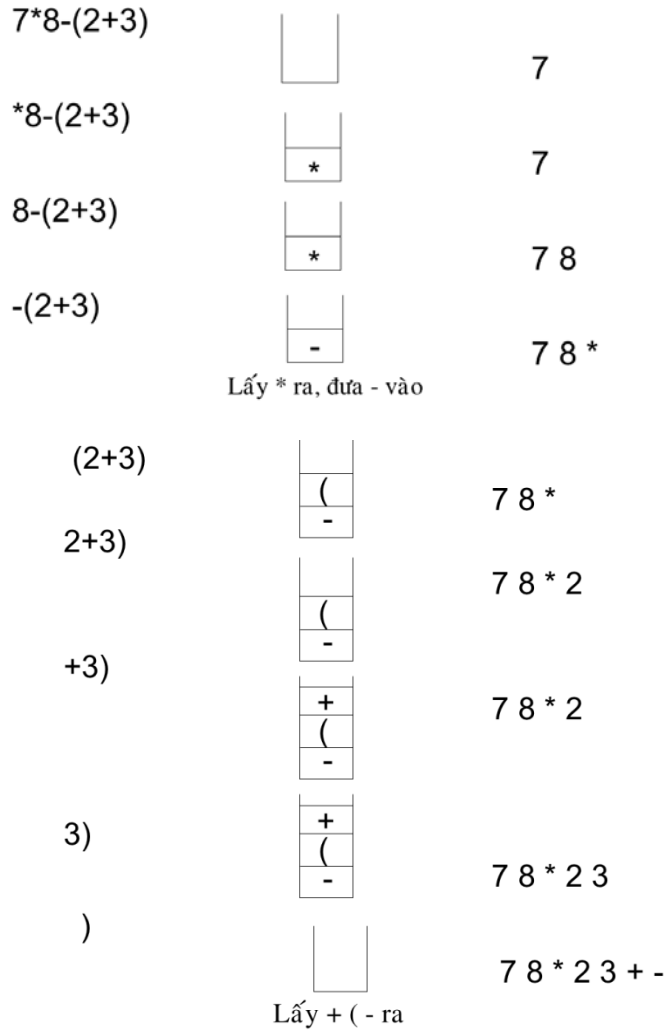
V. Ngăn xếp

VI. Hàng đợi

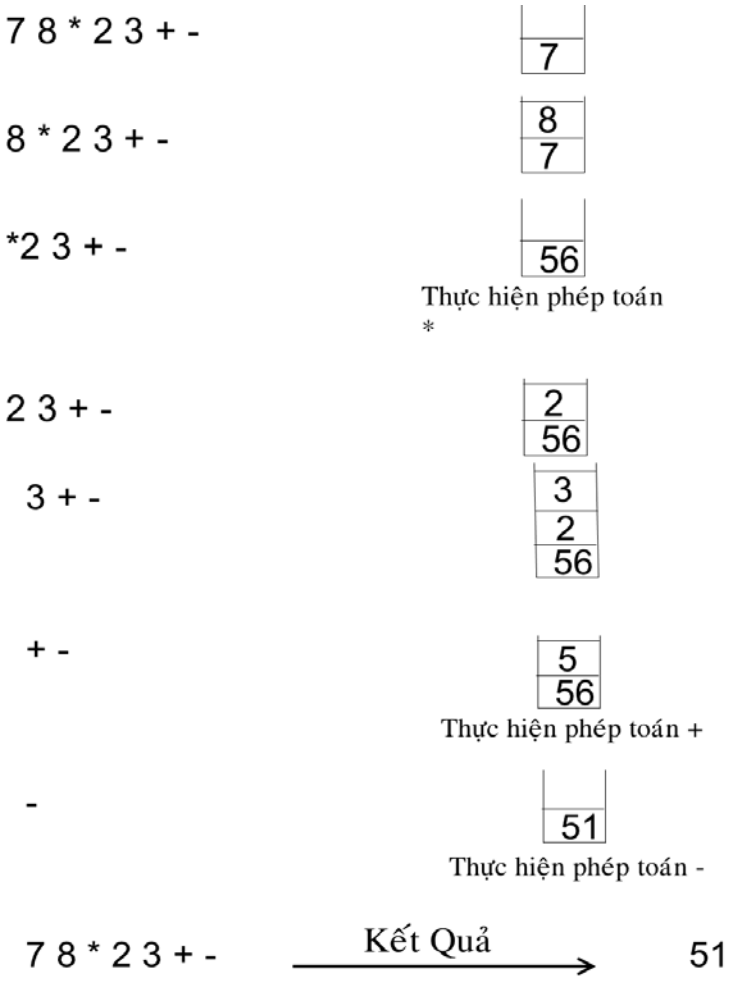
VII. Ứng dụng mở rộng của danh sách liên kết

Ví dụ:

Bước 1: tìm biểu thức Ba Lan ngược của biểu thức ”7*8– (2+3)”

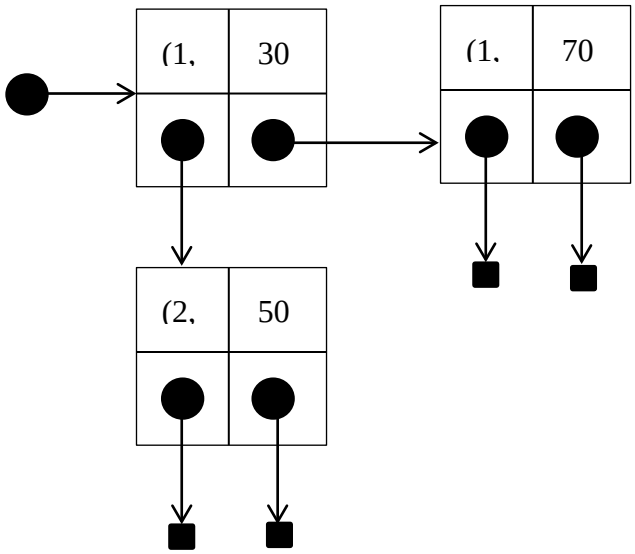


Bước 2: tính giá trị của biểu thức Ba Lan ngược

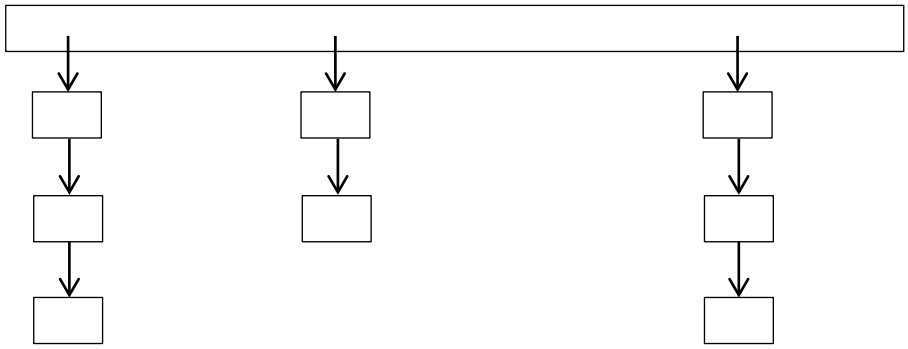


(vị trí hàng, vị trí cột)	Giá trị
Con trỏ đến node kế tiếp trên hàng	Con trỏ đến node kế tiếp trên cột

Hình 3.10. Cấu trúc một node.



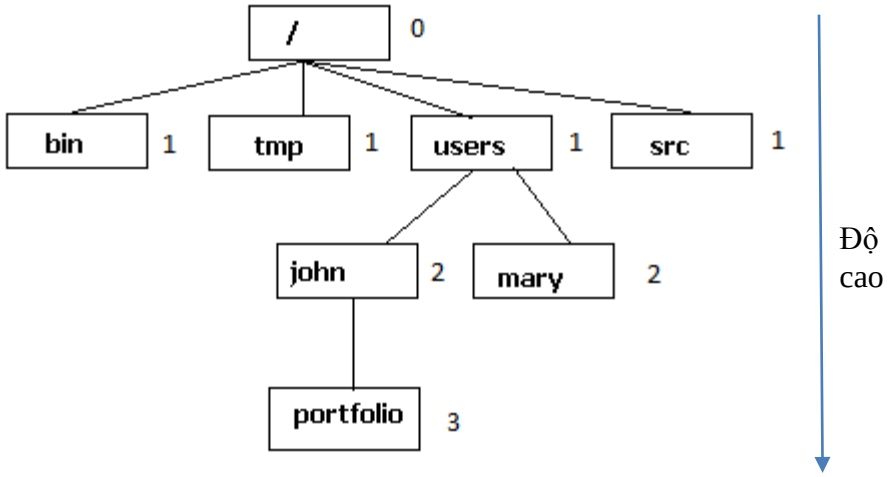
Hình 3.11. Ví dụ minh họa sử dụng DSLK như một cấu trúc mảng hai chiều.



Hình 3.12. Mảng các danh sách liên kết.

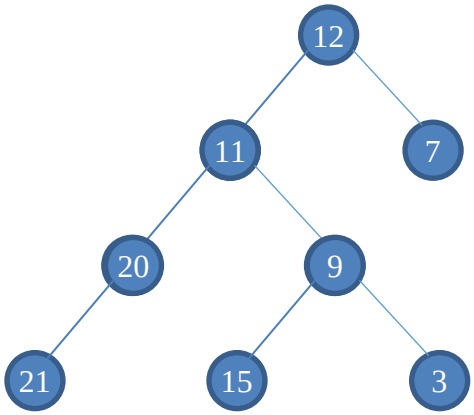
CHƯƠNG 4:
CÂY – CÂY NHỊ PHÂN TÌM KIẾM

I. Cây tổng quát



Hình 4.2. Mức và chiều cao của cây.

II. Cây nhị phân



Hình 4.3. Cây nhị phân.

Khai báo cài đặt trong C dạng cấu trúc:

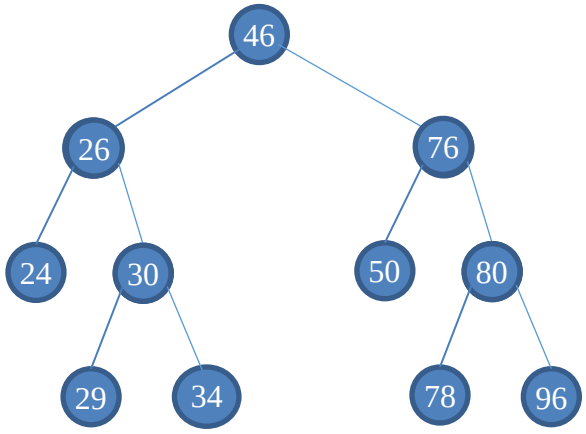
```
typedef struct tagNode{
        DataType Data;              // khai báo dữ liệu
        struct tagNode *Left;
        struct tagNode *Right;
};
```

```
typedef tagNode *Node;

typedef struct tagBinaryTree{
        Node Root;     //phần tử đầu tiên.
}BinaryTree;
```

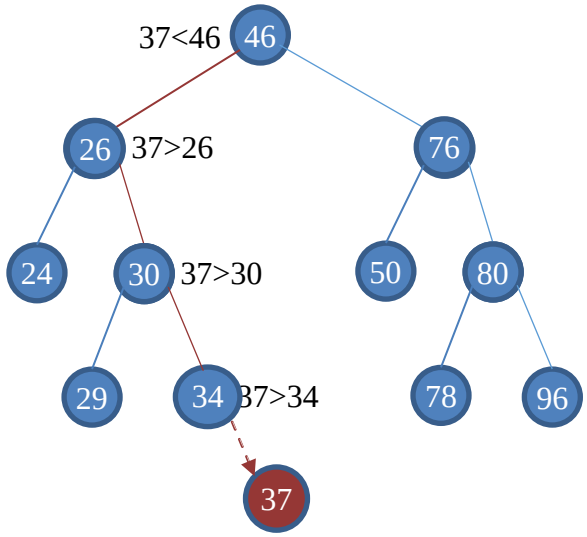
Khai báo cài đặt trong C dạng hướng đối tượng:

```
class tagNode{
        friend class BinaryTree;
        private:
                DataType        Data;
                tagNode                 *Left,*Right;
        public:
                tagNode(Data d=giá trị mặc định)
                        {Left=Right=NULL;Data = d}
                ~tagNode(){if(Left)delete Left;if(Right)delete Right;}
    };
typedef tagNode *Node;

class BinaryTree{
    private:
        Node Root;
    public:
        BinaryTree(){Root=NULL;}
        ~BinaryTree(){if(Root)delete Root;}
        // Các phương thức thêm, xoá, tìm kiếm,
        // đếm, …
};
```

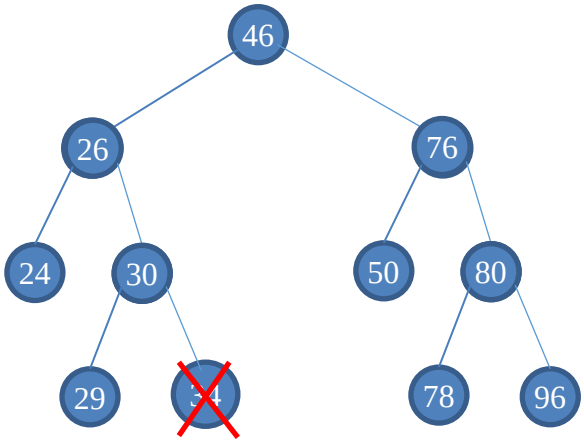
III. Cây nhị phân tìm kiếm



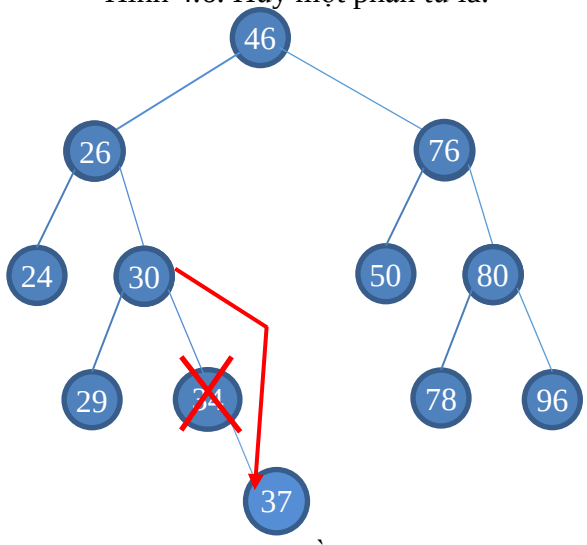
Hình 4.4. Cây nhị phân tìm kiếm.



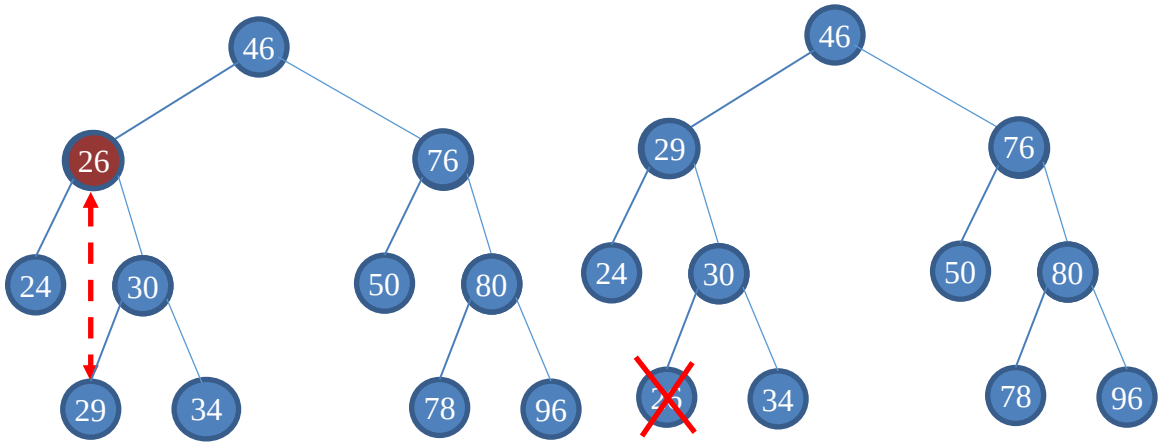
Hình 4.5. Thêm phần tử “37” vào cây NPTK.



Hình 4.6. Hủy một phần tử lá.

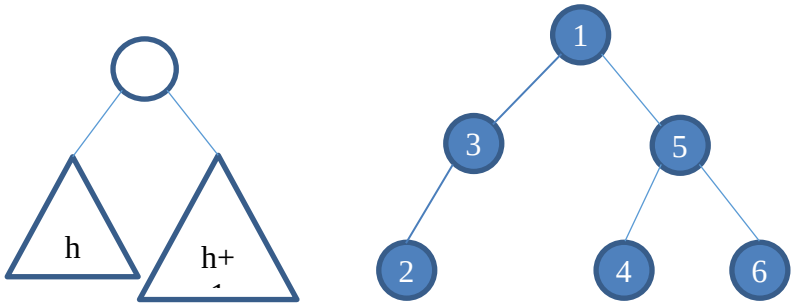


Hình 4.7. Hủy phần có một con.



Hình 4.8. Hủy phần tử có 02 con.

IV. Cây AVL

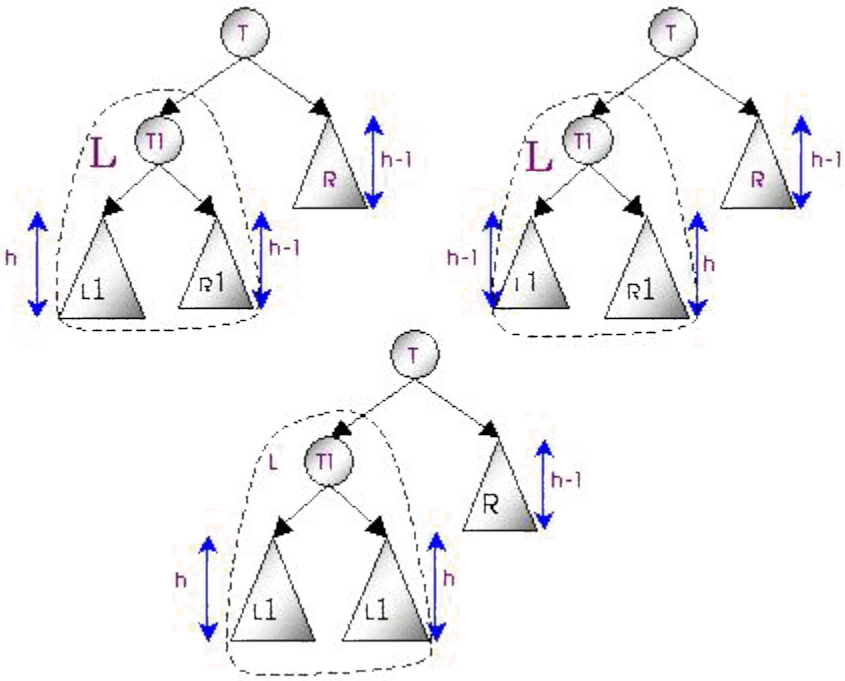


Hình 4.11. Cây AVL.

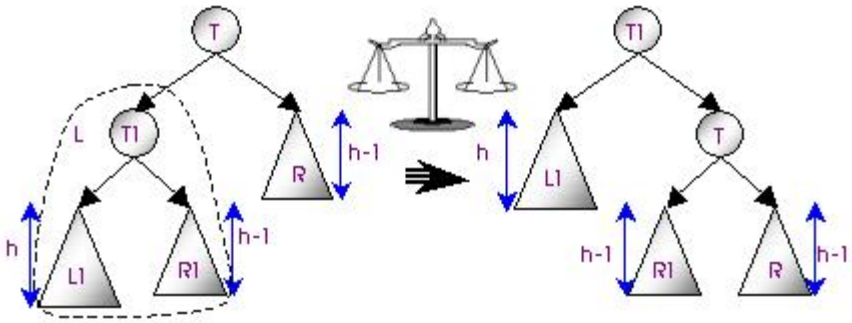
```
#define LH -1          //Cây con trái cao hơn
#define EH  0          //Hai cây con bằng nhau
#define RH  1          //Cây con phải cao hơn
```

Khai báo cài đặt trong C dạng cấu trúc:

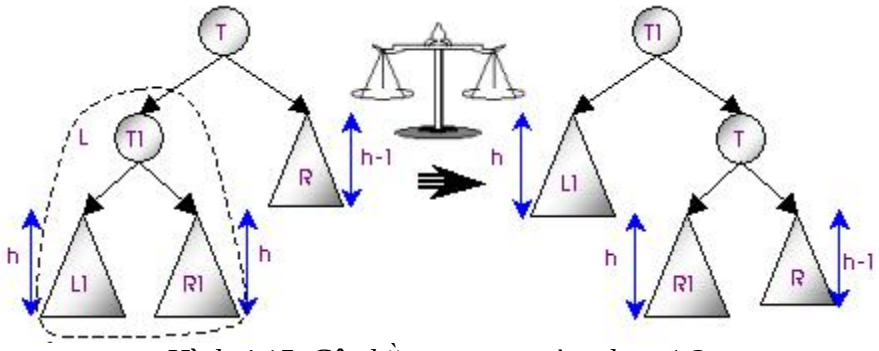
```
typedef struct tagAVLNode{
        char balFactor;             //Chỉ số cân bằng
        DataType Data;              // khai báo dữ liệu
        struct tagAVLNode *Left;
        struct tagAVLNode *Right;
};

typedef tagAVLNode *AVLNode;

typedef struct tagAVLTree{
        AVLNode Root;       //phần tử đầu tiên.
}AVLTree;
```



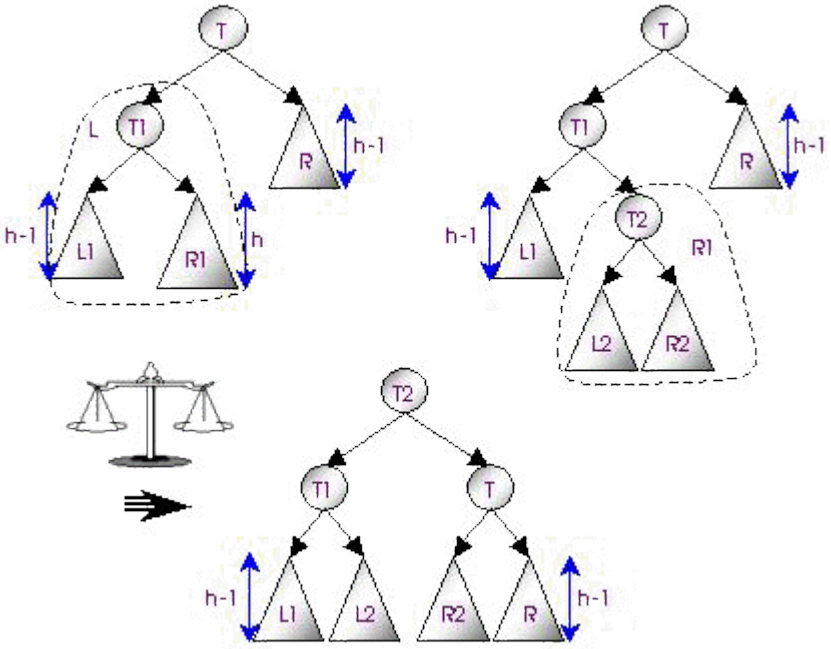
Hình 4.12. Ba trường hợp lệch về bên trái.



Hình 4.14. Cân bằng trong trường hợp 1.1.



Hình 4.15. Cân bằng trong trường hợp 1.2.



Hình 4.16. Cân bằng trong trường hợp 1.3.