ĐỀ CƯƠNG BÀI GIẢNG
BÀI 3: CÁC PHƯƠNG PHÁP TÌM KIẾM VÀ SẮP XẾP

3.2.5. Sắp xếp nâng cao

3.2.5.1. Sắp xếp phân đoạn

a. Giới thiệu phương pháp

Sắp xếp kiểu phân đoạn (Partition Sort) là một cải tiến của phương pháp sắp xếp đổi chỗ và là một phương pháp sắp xếp khá tốt. Chính vì thế, C.A.R. Hoare người tạo ra phương pháp này đã đặt tên cho nó là sắp xếp nhanh (Quick Sort).

Nguyên tắc của phương pháp này có thể được mô tả như sau:

- Chọn một khoá làm "*chốt*", thông thường để thuận lợi cho việc cài đặt, người ta chọn khóa trung tâm của "*đoạn được xét*" làm chốt.

- Xếp các khoá nhỏ hơn chốt về bên trái chốt (phía đầu dãy), các khoá lớn hơn chốt về bên phải chốt (phía cuối dãy) bằng cách:

Các phần tử trong dãy được so sánh với khoá chốt và sẽ đổi vị trí cho nhau hoặc cho chốt nếu chúng nằm trước chốt mà lại lớn hơn chốt hoặc nằm sau chốt mà lại nhỏ hơn chốt.

- Kết thúc một lượt đổi chỗ, dãy khoá được chia thành hai đoạn: một đoạn gồm các khoá nhỏ hơn chốt, một đoạn gồm các khoá lớn hơn chốt và (có thể) khoá chốt nằm ở giữa hai đoạn, cũng chính là vị trí đúng của nó trong dãy.

Lặp lại quá trình trên đối với từng phân đoạn cho đến khi toàn bộ dãy khoá được sắp xếp. Ở các lượt xử lý tiếp theo, chỉ có một phân đoạn được xử lý, còn phân đoạn còn lại phải được ghi nhớ và xử lý sau.

b. Minh hoạ phương pháp phân đoạn

Sắp xếp dãy khoá X:

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
42	23	74	11	65	58	94	36	99	87

Với quy ước chọn khoá chốt là khóa trung tâm trong dãy đang xét, như vậy với dãy khoá ban đầu thì khoá chốt là $t = X_5 = 65$.

Giả sử cần sắp xếp một đoạn từ X_{left} đến X_{right} trên dãy. Để xác định vị trí các khoá cần đổi chỗ cho nhau, ta dùng hai biến i và j để duyệt từ hai đầu của dãy khóa như sau:

(1) Ban đầu, i = left; j = right;

(lượt sắp đầu tiên left=1, right=10, và $t = X_5 = 65$ với dãy trên)

(2) Nếu $X_i < t$ thì tăng i lên 1 và lặp lại quá trình đó

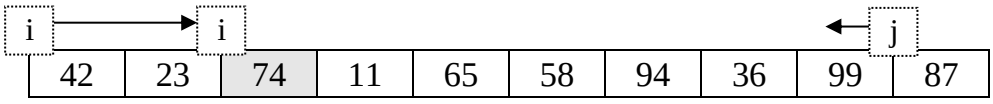
Nếu $X_j > t$ thì j được giảm đi 1 và lặp lại quá trình đó

(3) Đổi chỗ X_i và X_j cho nhau (nếu i<j)

Quay lại bước (2) cho đến khi i > j thì dãy X được chia thành 3 dãy nếu vị trí chốt không bị đổi chỗ, ngược lại dãy X được chia thành 2 dãy và kết thúc một lượt sắp xếp. Từ X_l đến X_j là dãy gồm các phần tử nhỏ hơn chốt, và từ X_i đến X_r là các phần tử lớn hơn chốt.

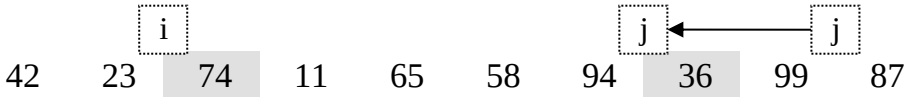
Diễn biến của lượt đầu được mô tả như sau:

Chọn t = X_5 = 65 làm chốt, i=left=1, j=right=10.

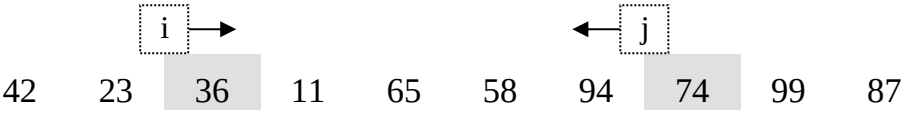


Tăng i cho đến khi gặp khoá không nhỏ hơn chốt

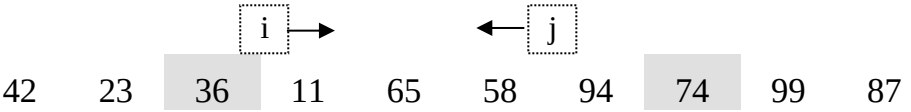
sau đó



Giảm j cho đến khi gặp khóa không lớn hơn chốt

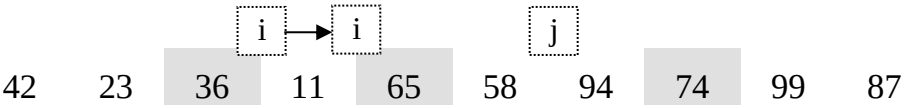


Đổi chỗ X_i và X_j (khi i<j)



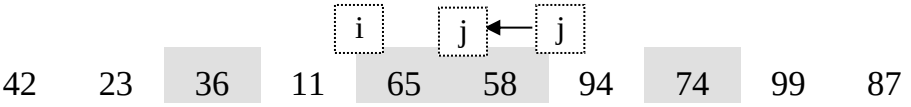
Sau đó i tăng 1 đơn vị và j giảm 1 đơn vị

Tiếp tục quá trình

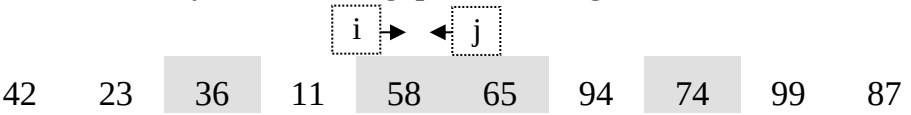


Tăng i cho đến khi gặp khoá không nhỏ hơn chốt

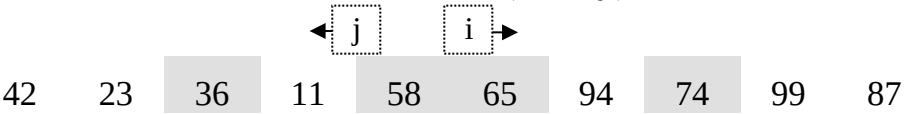
sau đó



Giảm j cho đến khi gặp khóa không lớn hơn chốt



Đổi chỗ X_i và X_j (khi i<j)



Sau đó i tăng 1 đơn vị và j giảm 1 đơn vị

Hình 1. Mô tả phương pháp sắp xếp phân đoạn

Đến đây kết thúc sắp xếp lượt một vì j<i.

Như vậy, sau một lượt duyệt, dãy khoá được chia thành hai phân đoạn, phân đoạn 1 từ X_{left} đến X_j gồm các phần tử nhỏ hơn chốt và phân đoạn 2 từ X_i đến X_{right} gồm các phần tử lớn hơn chốt.

Tiếp tục sắp xếp phân đoạn 1 và phân đoạn 2 cũng theo kỹ thuật như ở lượt đầu. Quá trình phân đoạn kết thúc khi mỗi phân đoạn chỉ còn một phần tử. Đến đây ta hoàn thành việc sắp xếp dãy.

Các phân đoạn được sắp với cùng một kỹ thuật, vì vậy khi cài đặt giải thuật thuật sắp xếp người ta sử dụng kỹ thuật đệ quy.

c. Giải thuật

Dưới đây là giải thuật sắp xếp bằng phương pháp phân đoạn được viết tựa ngôn ngữ C, đây là một giải thuật được thiết kế theo kỹ thuật đệ quy.

```
void QuickSort (X[], left, right)  /*Sắp xếp dãy khoá X với phân đoạn từ X_left đến X_right */
{
    if ( left<right ) //Việc phân đoạn chỉ thực hiện với phân đoạn có 2 phần tử trở lên
    {
        i = left;
        j = right;
        k=(left+right)/2;
        t =X_k ;
        while (i<=j)
        {
            while ( X_i < t )   i = i + 1;
            while ( X_j > t )   j = j - 1;
            if ( i <=j )
            {
                tg = X_i;  X_i  = X_j; X_j = tg;
                i++; j--;
            }
        }
        QuickSort (X, left, j) ;
        QuickSort (X, i, right) ;
    }
}
```

d. Đánh giá phương pháp

Trước hết ta hãy để ý đến một vài chi tiết có ảnh hưởng tới hiệu lực của phương pháp, đồng thời cũng thể hiện rõ đặc điểm của phương pháp này.

a. Vấn đề chọn "chốt":

Trong phần minh họa trên ta chọn chốt là phần tử trung tâm của phân đoạn cần sắp. Tuy nhiên, bạn có thể chọn phần tử bất kỳ trong phân đoạn để làm chốt. Nhưng rõ ràng khi thể hiện giải thuật ta phải định ra một cách thức chọn chốt cụ thể.

Vấn đề là chốt mà ta chọn được có tốt không? Nếu chốt ta chọn rơi vào đúng khoá nhỏ nhất (hoặc lớn nhất) của phân đoạn cần xử lý thì sau mỗi lượt ta chỉ tách ra được một phân đoạn con có kích thước nhỏ hơn trước là 1 phần tử (vì đã bớt đi một phần tử là "chốt") và phân đoạn này tiếp tục được xử lý. Như vậy ta đã quay trở lại phương pháp sắp xếp kiểu nổi bọt đơn giản. Việc chọn chốt như thế này đã dẫn đến tình huống xấu nhất của phương pháp.

Nếu gọi "*trung vị"* (median) của một dãy khoá là khoá sẽ đứng ở giữa dãy đó sau khi dãy đã được sắp xếp, nghĩa là nó lớn hơn một nửa số khoá của dãy và nhỏ hơn số còn lại, thì tốt nhất vẫn là chọn được đúng trung vị làn "chốt". Lúc đó sau mỗi lượt ta sẽ tách ra được hai phân đoạn con có độ dài gần như nhau và phân đoạn xử lý tiếp theo có kích thước chỉ bằng "nửa" phân đoạn đã chứa nó.

Nhưng làm sao có thể chọn được đúng trung vị? Nếu giả thiết: sự xuất hiện của các khoá trong dãy là đồng khả năng thì trung vị có thể là bất kỳ một khoá nào trong dãy. Trong giải thuật trên, ta chọn khoá đứng đầu làm chốt là dựa trên cơ sở này. Nhưng với cách chọn này, nếu dãy khoá có khuynh hướng đã theo thứ tự sắp xếp thì khả năng xấu nhất lại xuất hiện. Tuy nhiên nếu khuynh hướng này hay xuất hiện thì việc chọn khoá đang được đứng ở giữa dãy lại gặp thuận lợi.

Để dung hoà với cách chọn như trên, đồng thời cũng để kết hợp với một đề nghị sau này của Hoare là: "Chọn trung vị của một dãy khoá nhỏ hơn, thuộc dãy khoá cho, làm chốt", R.C.Singleton đã đưa ra một cách chọn là:

Chọn X[q] là "chốt" với X[q] là trung vị của ba khoá X[1eft], X[(left+right)/2] và X[right] trong đó left và right là chỉ số của khoá đầu và khoá cuối của phân đoạn. Các kiểu chọn khoá chốt còn được nhiều tác giả khác nữa đưa ra và cũng có nhiều kết quả đáng chú ý.

b. Vấn đề phối hợp với cách sắp xếp khác

Khi kích thước của các phân đoạn đã khá nhỏ, việc tiếp tục phân đoạn nữa theo QUICK-SORT thực ra sẽ không có lợi. Lúc đó sử dụng một số phương pháp sắp xếp đơn giản lại tiện hơn. Vì vậy, sắp xếp NHANH thường không tiến hành triệt để mà dừng lại ở lúc cần thiết để gọi tới một phương pháp sắp xếp đơn giản, giao cho nó tiếp tục thực hiện sắp xếp với các phân đoạn nhỏ còn lại. Kunth (1974) có nêu: 9 có thể coi là kích thước giới hạn của phân đoạn để sau đó QUICK-SORT gọi tới phương pháp sắp xếp đơn giản.

Ngoài ra việc chọn phân đoạn nào để xử lý tiếp theo cũng là một vấn đề cần được xem xét tới.

Bây giờ ta sẽ đánh giá giải thuật QUICK-SORT.

Gọi T(n) là thời gian thực hiện giải thuật ứng với một bảng n khoá, P(n) là thời gian để phân đoạn một bảng n khoá thành hai bảng con. Ta có thể viết:

$$T(n) = P(n) + T(j\text{-}left) + T(right\text{-}j)$$

Chú ý rằng P(n) = Cn với C là hằng số.

Trường hợp xấu nhất xảy ra khi bảng khoá vốn đã có thứ tự sắp xếp: sau khi phân đoạn một trong hai bảng con là rỗng (j = left hoặc j = right)

Giả sử j = left, ta có:

$$T_x(n) = P(n) + T_x(0) + T_x(n-1)$$
$$= Cn + T_x(n-1)$$
$$= Cn + C(n-1) + T_x(n-2)$$
$$\dots$$
$$= \sum_{k=1}^{n} C_k + T_x(0)$$
$$= C.\frac{n(n+1)}{2} = O(n^2)$$

Trường hợp này QUICK-SORT không hơn gì các phương pháp đã nêu trước đây.

Trường hợp tốt nhất xảy ra khi bảng luôn luôn được chia đôi, nghĩa là j=(left+right)/2. Lúc đó:

$$T_1(n) = P(n)b + 2T_1(n/2)$$
$$= Cn + 2T_1(n/2)$$
$$= Cn + 2C(n/2) + 4T_1(n/4) = 2Cn_n + 2^2T_1(n/4)$$
$$= Cn + 2C(n/2) + 4C(n/4) + 8T_1(n/8) = 3C_n + 2^3T_1(n/8)$$
$$\dots$$
$$= (\log_2 n)\, Cn + 2^{long}2^mT_1(1)$$
$$= O(n\log_2 n)$$

Việc xác định giá trị trung bình $T_{tb}(n)$ không còn đơn giản như hai trường hợp trên, nên ta sẽ không xét chi tiết. Kết quả mà ta cần ghi nhận là: người ta đã chứng minh được:

$$T_{tb}(n) = O(n\log_2 n)$$

Như vậy rõ ràng là khi n khá lớn hơn QUICK-SORT đã tỏ ra có hiệu lực hơn hẳn ba phương pháp đã nêu. Tuy nhiên, việc sử dụng kỹ thuật đệ qui sẽ tiêu tốn rất nhiều bộ nhớ khi thực hiện giải thuật trên máy tính.

Sau đây ta xem xét một phương pháp sắp xếp với thời gian cũng rất tốt đó là phương pháp vun đống.

3.2.5.2. Sắp xếp vun đống

a. Giới thiệu phương pháp

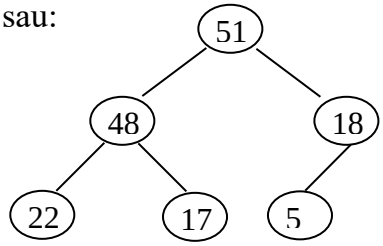
Trước tiên, ta xem xét khái niệm đống (Heap)

Đống là một cây nhị phân hoàn chỉnh mà mỗi nút được gán một giá trị khoá sao cho khoá ở nút cha bao giờ cũng lớn hơn khóa ở các nút con.

Đống được lưu trữ kế tiếp trên máy (bởi mảng một chiều)

Như vậy đối với đống, nếu nút cha ở vị trí thứ i thì 2 nút con (nếu có) sẽ ở vị trí thứ 2*i và 2*i+1.

Ví dụ: đống là cây T có dạng như sau:



Hình 2: Đống

b. Nguyên tắc sắp xếp:

Sắp xếp kiểu vun đống (Heap Sort) được chia thành hai giai đoạn:

- Giai đoạn tạo đống: Cây nhị phân biểu diễn dãy khoá ban đầu được biến đổi để tạo thành đống.

- Giai đoạn sắp xếp: ở giai đoạn này, ta tiến hành nhiều lượt hai phép xử lý sau:

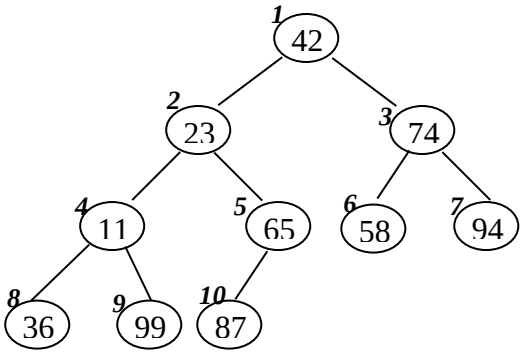
  + Đưa khóa trội về đúng vị trí, bằng cách thực hiện hoán vị các khoá.

  + Vun lại cây gồm các khoá còn lại (sau khi đã loại khoá trội) thành đống.

Ví dụ mô tả: Giả sử với dãy khoá X

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
42	23	74	11	65	58	94	36	99	87

Ta có cây nhị phân biểu diễn dãy khoá ban đầu như sau:

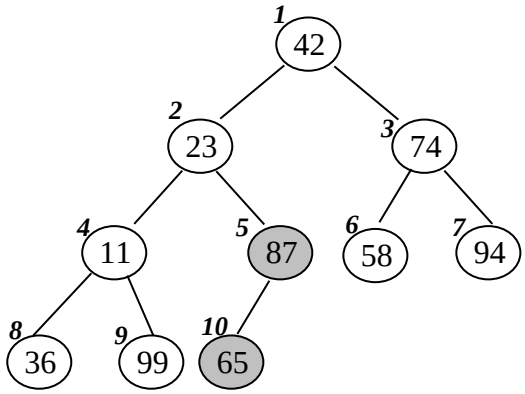


Hình 3: Cây nhị phân biểu diễn dãy khóa

Các bạn lưu ý, cây nhị phân trên đây chỉ là mô hình giúp ta hình dung được quá trình sắp xếp theo phương pháp vun đống. Còn quá trình sắp xếp thực sự vẫn là việc hoán đổi vị trí trực tiếp các phần tử mảng.

Với cây này để nó trở thành đống ta làm như sau:

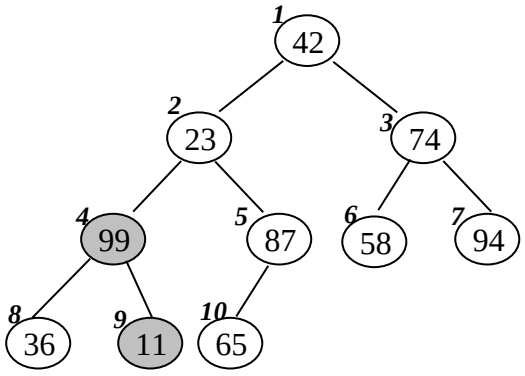
+ Cây con có nút gốc là 65 (vị trí thứ 5 trở thành đống) – hình 4



Hình 4: Xử lý nút khóa X_5

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
42	23	74	11	87	58	94	36	99	65

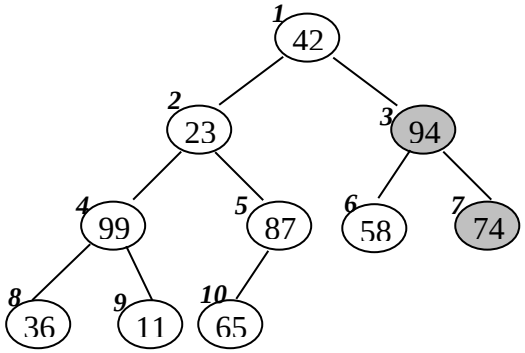
+ Cây con có nút gốc 11 (nút thứ 4) trở thành đống (hình 5)



Hình 5: Xử lý nút khóa X_4

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
42	23	74	99	87	58	94	36	11	65

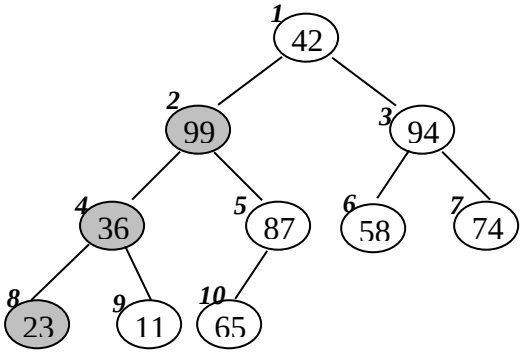
+ Cây con có nút gốc 74 (nút thứ 3) trở thành đống – hình 6



Hình 6: Xử lý nút khóa X_3

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
42	23	94	99	87	58	74	36	11	65

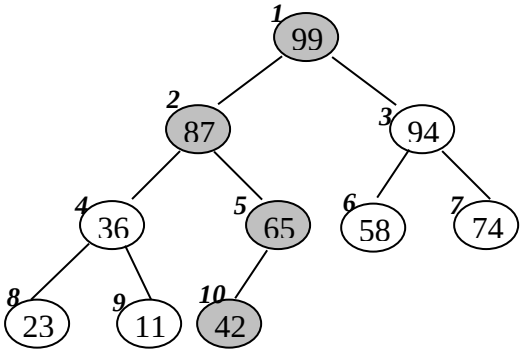
+ Cây con có nút gốc 23 trở thành đống (nút thứ 2) – hình 7



Hình 7: Xử lý nút khóa X_2

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
42	99	94	36	87	58	74	23	11	65

+ Cuối cùng, cây con có nút gốc 42 trở thành đống (nút thứ 1) – hình 8

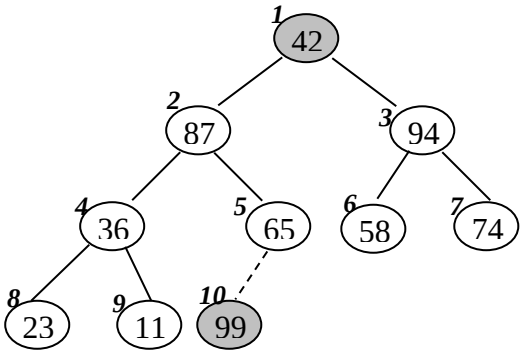


Hình 8: Xử lý nút khóa X_1 - Đống đầu tiên

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
99	87	94	36	65	58	74	23	11	42

Đến đây quá trình tạo đống đầu tiên kết thúc, ta thấy khóa trội được chuyển về về vị trí đầu tiên của mảng (nút gốc của cây-đống)

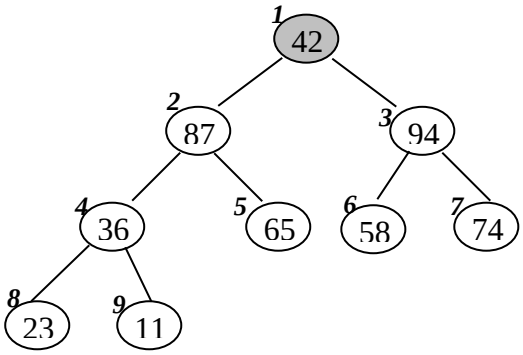
Sang giai đoạn 2, khoá trội 99 được chuyển đến vị trí cuối cùng bằng cách hoán vị cho khoá 42, sau khi hoán vị, nút ứng với khóa trội 99 coi như được loại khỏi cây – hình 9.



Hình 9: Hoán đổi nút đầu với nút cuối

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
42	87	94	36	65	58	74	23	11	**99**

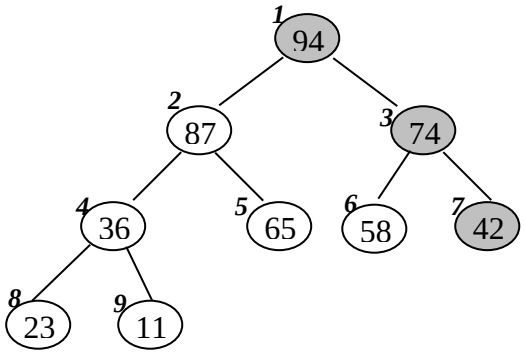
Cây còn lại được vun lại thành đống. Nó có dạng như hình 10 sau đây:



Hình 10: Cây sau khi loại nút cuối

Nhận thấy rằng, cây này chưa phải là một đống, nhưng tất cả các cây con của cây này đều là đống. Vì vậy, ở bước này và tất cả các bước còn lại ta chỉ cần xét nút gốc (nút thứ 1) của cây.

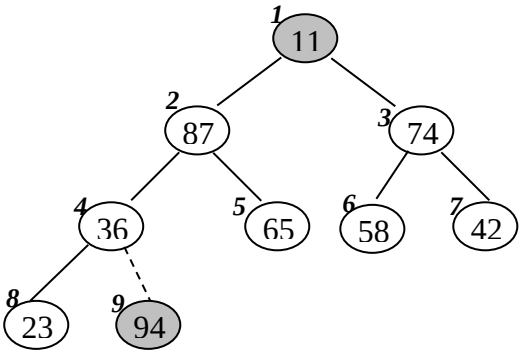
Ta có đống thứ 2 như sau – hình 11



Hình 11: Đống thứ 2

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
94	87	74	36	65	58	42	23	11	**99**

Lại đổi vị trí nút đầu tiên và nút cuối cùng – hình 12



Hình 12: Đổi vị trí nút đầu và nút cuối

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
11	87	74	36	65	58	42	23	**94**	**99**

Sau đó một lượt xử lý tiếp theo được thực hiện tương tự và cứ như thế cho đến khi toàn bộ dãy khóa được sắp xếp. Bạn đọc có thể tự mình mô tả quá trình sắp xếp trong các bước tiếp theo và mô tả việc sắp xếp dãy khóa theo thứ tự giảm dần.

c. Giải thuật

Nhận thấy, có thể coi một nút lá là một cây con đã thoả mãn tính chất của đống, có nghĩa một nút lá được coi là một đống. Như vậy, bước tạo đống hay vun đống được quy về một phép xử lý chung là: chuyển một cây thành đống mà cây con trái và cây con phải của gốc đã là đống rồi. Ta xây dựng riêng một thủ tục vun đống là thủ tục Hoan_vi.

Mặt khác, đối với cây nhị phân hoàn chỉnh có n nút, chỉ các nút ứng với các vị trí từ 1 đến (n/2) mới có thể là nút cha của các nút khác. Nên khi tạo đống thủ tục Hoan_vi chỉ

cần áp dụng với các cây con có gốc ở vị trí (n/2), (n/2) -1, …, 1. Còn khi vun đống thì luôn áp dụng với cây có gốc ở vị trí 1.

Giải thuật sắp xếp dãy n khoá X_1, X_2, …, X_n được biểu diễn bởi 3 thủ tục tựa ngôn ngữ lập trình C sau đây.

```
void Hoan_vi(X[],k,r)
{
    if (X[k] khác lá và có giá trị nhỏ hơn 2 con)
    {   + Chọn con có giá trị lớn hơn, giả sử là X[j];
        + Đổi chỗ X[k], và X[j] ;
        + call Hoan_vi(X,j,r);
    }
}
```

Thủ tục này thực hiện biến đổi cây có gốc là X[k] thành đống, với r là vị trí của khoá cuối cùng trong dãy được xét. Thủ tục này cũng có thể viết dưới dạng một giải thuật lặp để cải thiện thời gian sắp xếp. Bạn đọc dựa vào minh hoạ trên để "làm mịn" thủ tục này.

```
void Tao_dong_dau_tien(X[], n)
{
    for (i = n/2; i>=1; i--)
        Hoan_vi(X, i, n);
}
```

Thủ tục này thực hiện biến đổi dãy ban đầu (n khoá) thành dãy biểu diễn đống đầu tiên. Quá trình biến đổi đống đầu tiên được thực hiện từ dưới lên, các đống tiếp theo sẽ được thực hiện từ trên xuống (bạn đọc có thể xem lại phần minh họa)

```
void Heap_sort(X[], n)
{
    Tao_dong_dau_tien(X,n);
    for (i=n; i>=2; i--)
    {   Đổi chỗ (X[1], X[i]);
        Hoan_vi(X,1,i-1);
    }
}
```

Thủ tục này thực hiện việc sắp dãy khoá theo chiều tăng dần, sử dụng hai thủ tục bên trên. Tuy nhiên, thủ tục Hoan_vi được viết dưới dạng đệ qui, vì thế làm chậm quá trình sắp

xếp và tốn bộ nhớ. Trường hợp này ta nên khử đệ qui, nghĩa là sử dụng giải thuật lặp, việc này khá đơn giản, bạn đọc tự nghiên cứu, xem như một bài tập.

d. Phân tích đánh giá

Đối với HEAP_SORT ta chú ý tới việc đánh giá trong trường hợp xấu nhất. Như ta đã biết: một cây nhị phân hoàn chỉnh có n nút thì chiều cao của cây đó là $[\log_2(n+1)]$. Khi tạo đống cũng như khi vun lại đống trong giai đoạn sắp xếp, trường hợp xấu nhất thì số lượng phép so sánh cũng chỉ tỷ lệ với chiều cao của cây. Do đó có thể suy ra, trong trường hợp xấu nhất, cấp độ lớn của thời gian thực hiện HEAP_SORT chỉ là $O(n\log_2 n)$. Việc đánh giá thời gian thực hiện trung bình phức tạp hơn, ta không xét tới mà chỉ ghi nhận một kết quả đã được chứng minh là: cấp độ lớn của thời gian thực hiện trung bình giải thuật HEAP_SORT cũng là $O(n\log_2 n)$.

Có thể nhận xét thêm là: QUICK-SORT còn phải dùng thêm không gian nhớ cho stack, để bảo lưu thông tin về các phân đoạn sẽ được xử lý tiếp theo (vì thực hiện đệ quy). Còn HEAP_SORT thì ngoài một nút nhớ phụ, để thực hiện đổi chỗ, nó không cần thêm gì nữa.

3.2.5.3. Sắp xếp trộn

Bây giờ ta xét tới một phương pháp sắp xếp mới, khá đặc biệt ở chỗ nó dựa trên một phép toán rất đơn giản là phép hòa nhập.

a. Phép trộn hai dãy đã sắp xếp

Một tư tưởng hết sức đơn giản là trộn hai dãy đã được sắp xếp thành một dãy được sắp xếp.

Giả sử cho hai dãy được sắp

X: 12 25 28 và

Y: 3 9 15 32 39

Khi đó ta sẽ trộn hai dãy X và Y thành dãy Z cũng được sắp tăng như sau:

Z: 3 9 12 15 25 28 32 39

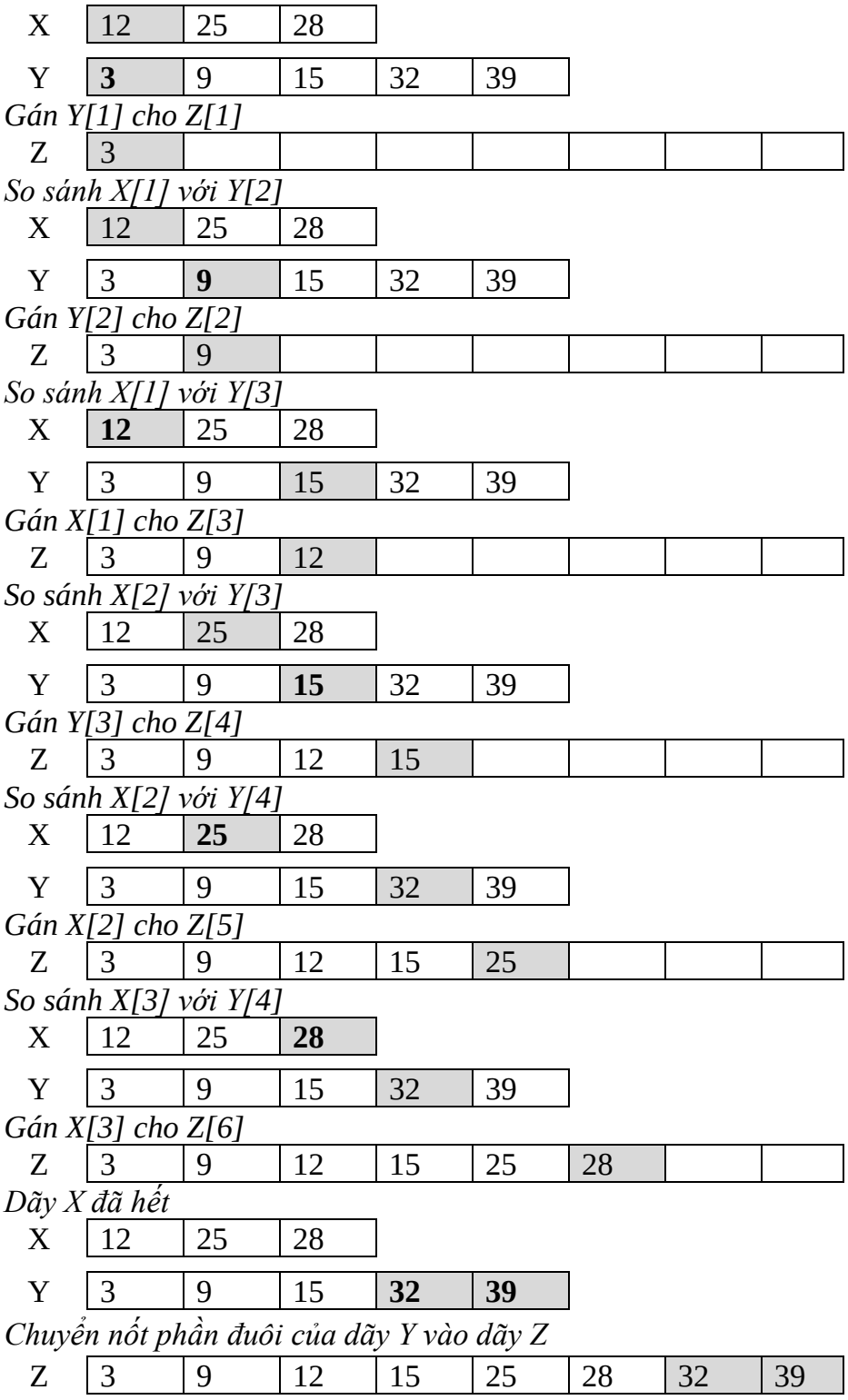
Có thể mô tả cách trộn đơn giản như sau:

+ So sánh hai khóa nhỏ nhất của hai dãy X và Y, chọn khóa nhỏ hơn đặt vào vị trí thích hợp trong dãy Z, rồi loại khóa được chọn khỏi dãy chứa nó.

+ Quá trình như vậy cứ tiếp tục cho đến khi một trong hai dãy X hoặc Y đã hết, khi đó chuyển nốt phần đuôi của dãy còn lại vào dãy Z là xong.

Hình vẽ sau minh họa cách làm trên

So sánh X[1] với Y[1]



Hình 13: Trộn dãy X và dãy Y sắp tăng, thành dãy Z sắp tăng

Bây giờ ta xét tới giải thuật MERGING, các dãy X, Y, Z được lưu trữ bởi các mảng, với m, n lần lượt là độ dài của X và Y. Việc so sánh X[i] và Y[j] sẽ xác định phần tử của dãy nào được gán cho Z[k].

```
void  MERGING(X[], m, Y[], n, Z)
{
```

```
//1. Khởi tạo các chỉ số
        i=1; j=1; k=1;
//2. Chuyển các phần phần tử từ dãy X, Y vào dãy Z
        while (i<=m && j<=n)
        {
                if (X[i]<Y[j])
                {
                        Z[k]=X[i]; i++; k++;
                }
                else
                {
                        Z[k]=Y[j]; j++; k++;
                }
        }
//3. Một trong hai mạch đã hết, chuyển đuôi của dãy còn lại vào Z
        while (i<=m)
        {
                Z[k] = X[i]; i++; k++;
        }
        while (j<=n)
        {
                Z[k] = Y[j]; j++; k++;
        }
}
```

b. Sắp xếp trộn.

1. Giới thiệu phương pháp

Với ý tưởng trộn hai dãy đã sắp xếp thành một dãy đã sắp xếp người ta đã phát triển thành một phương pháp sắp xếp mới mà ta quen gọi là phương pháp trộn.

Giả sử cần sắp xếp dãy khóa X với độ dài n: X_1, X_2, ..., X_n. Có thể thấy rằng mỗi khóa X_i trong dãy có thể xem như một dãy con được sắp với độ dài là 1(sau đây ta gọi là một vệt).

Nếu hòa nhập hai vệt như vậy ta sẽ được một vệt mới với độ dài là 2. Lại hòa nhập hai vệt độ dài 2 ta được một vệt độ dài 4, tiếp tục hòa nhập như thế, cuối cùng ta sẽ được một vệt có chiều dài n, như thế dãy ban đầu được sắp xếp hoàn toàn.

**** Ta mô tả phương pháp vừa nêu trong ví dụ sau:***

Cho dãy khóa: 42 23 74 11 65 58 94 36 99 87

Coi dãy khóa gồm 10 mạch có độ dài 1

[42] [23] [74] [11] [65] [58] [94] [36] [99] [87]

Hòa nhập các cặp vệt độ dài 1 kề nhau ta được

[23 42] [11 74] [58 65] [36 94] [87 99]

Hòa nhập các cặp vệt độ dài 2 kề nhau ta được

[11 23 42 74] [36 58 65 94] [87 99]

Ta giữ nguyên vệt lẻ cặp: [87 99]

Hòa nhập các cặp vệt độ dài 4 kề nhau ta được

[11 23 36 42 58 65 74 94] [87 99]

Ta giữ nguyên vệt lẻ cặp: [87 99]

Cuối cùng, hòa nhập hai vệt ta có dãy được sắp

[11 23 36 42 58 65 74 87 94 99]

Qua ví dụ trên bạn đọc đã hiểu được phương pháp sắp trộn, sau đây ta đi thiết kế giải thuật cho phương pháp này.

2. Giải thuật

Giải thuật gồm ba công đoạn

+ Thủ tục trộn hai vệt thành một vệt

+ Thủ tục trộn các cặp vệt độ dài *l* (có thể một vệt có chiều dài nhỏ hơn *l*).

+ Thủ tục sắp xếp trộn

Ta cải tiến thủ tục MERGING thành thủ tục trộn hai vệt kề nhau trên dãy khóa X, vệt sau khi trộn nằm trên dãy Z.

```
void MERGE (X[], bt1, w1, bt2, w2, Z[])
{
    //bt1, bt2: là vị trí biên trái của hai vệt, w1, w2 là độ dài của hai vệt
        i=bt1; j=bt2; bp1=bt1+w1-1; bp2=bt2+w2-1; k=bt1;
    //bp1, bp2 là biên phải của hai vệt, k là biên trái của vệt mới trên dãy Z
    while (i<=bp1 && j<=bp2)
    {
        if (X[i]<X[j])
        {
            Z[k] = X[i]; i++; k++;
        }
        else
```

```
        {
            Z[k] = X[j]; j++; k++;
        }
    }
    while (i<=bp1)
    {
        Z[k] = X[i]; i++; k++;
    }
    while (j<=bp2)
    {
        Z[k] = X[j]; j++; k++;
    }
}
```

**** Thủ tục trộn một lần các cặp vệt.***

Với dãy khóa X độ dài n: X_1, X_2, ..., X_n, các dãy con trong X là các vệt có độ dài *l*, trừ dãy con cuối cùng có thể có độ dài nhỏ hơn *l*.

Nếu *cv* là số cặp vệt có độ dài *l* thì cv = n/(2**l)*

Đặt s = 2**l**cv thì s là số các phần tử thuộc các cặp vệt, và r = n-s là số các phần tử còn lại.

```
void MERGR_PASS (X[], n, l, Z[])
{
    //Z là dãy có độ dài n, chứa các phần tử của X sau khi hòa nhập
    //1. Khởi tạo các giá trị ban đầu
        cv = n/(2*l);
        s = 2*l*cv;
        r = n – s;
    //2. Hòa nhập từng cặp mạch
        for (j=1; j<=cv; j++)
        {
            b1 = l + (2*j -2)*l ; //biên trái của mạch thứ nhất
            MERGE(X, b1, l, b1+l, Z);
        }
    //3. Chỉ còn một vệt
        if (r<=l)
            for (j=1; j<=r; j++)
```

```
                Z[s+j] = X[s+j];
    //4. Còn hai vệt nhưng một vệt có độ dài nhỏ hơn l
            else MERGE(X, s+1, l, s+l+1, r-l, Z);
}
```

Thủ tục MERGE_PASS được gọi trong thủ tục thực hiện sắp xếp kiểu hòa nhập hai đường trực tiếp sau đây:

```
void MERGE_SORT (X[], n)
{
    //1. Khởi tạo số phần tử trong một vệt
        l =1;
    //2. Sắp xếp trộn
        while (l<n)
        {
        //hòa nhập và chuyển các phần tử vào dãy Z
            MERGE_PASS(X, n, l, Z);
        //hòa nhập và chuyển các phần tử trở lại dãy X
            MERGE_PASS(X, n, 2*l, Z);
            l = l*4;
        }
}
```

Sau khi sắp xếp xong các phần tử của dãy khóa X vẫn ở chỗ cũ.

3. Phân tích đánh giá

Trong phương pháp này ta thấy, số lượng phép toán chuyển chỗ thường nhiều hơn số lượng phép toán so sánh. Chẳng hạn, với thủ tục *MERGE*, trong câu lệnh *while*, ứng với một phép so sánh có một phép đổi chỗ. Nhưng nếu một vệt nào đó hết trước, thì phần đuôi của vệt còn lại được chuyển chỗ mà không tương ứng với một phép so sánh nào. Vì vậy, với phương pháp này ta chọn phép toán tích cực là phép toán chuyển chỗ để đánh giá thời gian thực hiện của giải thuật.

Nhận thấy rằng, ở bất kỳ lượt hòa nhập nào (MERGE_PASS) thì toàn bộ các khóa cũng được chuyển sang dãy mới (từ X sang Z, hoặc từ Z sang X). Như vậy chi phí thời gian cho một lượt hòa nhập là O(n). Ngoài ra số lượt gọi thủ tục MERGE_PASS trong thủ tục MERGE_SORT là $[\log_2 n]$, vì ở lượt 1 kích thước của vệt là $l=1=2^0$. Ở lượt i kích thước của vệt sẽ là 2^{i-1}, mà sau lượt cuối cùng thì vệt đã có kích thước là n.

Vậy sắp xếp kiểu hòa nhập hai đường trực tiếp có thời gian thực hiện là $O(n\log_2 n)$. Tuy nhiên, chi phí về không gian nhớ khá lớn, nó đòi hỏi 2n phần tử nhớ, gấp đôi so với các phương pháp khác. Do đó người ta thường sử dụng phương pháp này khi sắp xếp ngoài, sắp xếp dữ liệu trong tệp tin.

***Kết luận chung:**

Ta nhận thấy, với cùng một mục đích sắp xếp mà có rất nhiều phương pháp và kỹ thuật giải quyết khác nhau. Cấu trúc dữ liệu dược lựa chọn để mô tả đối tượng sắp xếp đã ảnh hưởng rất lớn tới việc lựa chọn giải thuật sắp xếp thích hợp. Các giải thuật sắp xếp đơn giản thể hiện ba kỹ thuật cơ sở của sắp xếp (dựa vào phép so sánh các giá trị khoá). Tuy nhiên, cấp độ lớn của thời gian thực hiện chúng là (n^2), do đó chỉ nên sử dụng chúng khi n nhỏ. Các giải thuật cải tiến như Quick_Sort, Heap_Sort đã đạt được thời gian thực hiện tốt hơn là $O(n\log_2 n)$, thường được sử dụng khi n lớn. Nếu dãy khoá cần sắp xếp vốn có khuynh hướng hầu như đã được sắp xếp sẵn rồi thì Quick Sort lại không nên dùng. Nhưng nếu ban đầu dãy có khuynh hướng ít nhiều có thứ tự ngược với thứ tự sắp xếp thì Heap Sort lại tỏ ra thuận lợi.

Việc khẳng định phương pháp sắp xếp này tốt hơn các phương pháp khác chỉ là tương đối, vì thế việc chọn một phương pháp sắp xếp thích hợp thường tuỳ thuộc vào từng yêu cầu, từng điều kiện cụ thể của bài toán.