CHƯƠNG 2: SẮP XẾP

2.1 TỔNG QUAN

2.1.1 Mục tiêu

Chương này sẽ trình bày một số phương pháp sắp xếp. Với mỗi phương pháp cần nắm vững các phần sau:

- Giải thuật sắp xếp.
- Minh họa việc sắp xếp theo giải thuật.
- Chương trình sắp xếp.
- Đánh giá giải thuật.

2.1.2 Kiến thức cơ bản cần thiết

Các kiến thức cơ bản cần thiết để học chương này bao gồm:

- Cấu trúc dữ liệu kiểu mẩu tin (record) và kiểu mảng (array) của các mẩu tin.
- Kiểu dữ liệu trừu tượng danh sách và thủ tục xen một phần tử vào danh sách (insert).
- Kĩ thuật lập trình và lập trình đệ quy.

2.1.3 Tài liệu tham khảo

A.V. Aho, J.E. Hopcroft, J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley. 1983. (Chapter 8).
Jeffrey H Kingston; *Algorithms and Data Structures*; Addison-Wesley; 1998. (Chapter 9).

Đinh Mạnh Tường. *Cấu trúc dữ liệu & Thuật toán*. Nhà xuất bản khoa học và kĩ thuật. Hà nội-2001. (Chương 9).

Đỗ Xuân Lôi. *Cấu trúc dữ liệu & Giải thuật*. 1995. (Chương 9).

2.1.4 Nội dung cốt lõi

Trong chương này chúng ta sẽ nghiên cứu các vấn đề sau:

- Bài toán sắp xếp.
- Một số giải thuật sắp xếp đơn giản.
- QuickSort
- HeapSort
- BinSort

2.2 BÀI TOÁN SẮP XẾP

2.2.1 Tầm quan trọng của bài toán sắp xếp

Sắp xếp một danh sách các đối tượng theo một thứ tự nào đó là một bài toán thường được vận dụng trong các ứng dụng tin học. Ví dụ ta cần sắp xếp danh sách thí sinh theo tên với thứ tự Alphabet, hoặc sắp xếp danh sách sinh viên theo điểm trung bình với thứ tự từ cao đến thấp. Một ví dụ khác là khi cần tìm kiếm một đối tượng trong một danh sách các đối tượng bằng giải thuật tìm kiếm nhị phân thì danh sách các đối tượng này phải được sắp xếp trước đó.

Tóm lại sắp xếp là một yêu cầu không thể thiếu trong khi thiết kế các phần mềm. Do đó việc nghiên cứu các phương pháp sắp xếp là rất cần thiết để vận dụng trong khi lập trình.

2.2.2 Sắp xếp trong và sắp xếp ngoài

Sắp xếp trong là sự sắp xếp **dữ liệu được tổ chức trong bộ nhớ trong** của máy tính, ở đó ta có thể sử dụng khả năng truy nhập ngẫu nhiên của bộ nhớ và do vậy sự thực hiện rất nhanh.

Sắp xếp ngoài là sự sắp xếp được sử dụng khi số lượng đối tượng cần sắp xếp lớn không thể lưu trữ trong bộ nhớ trong mà phải lưu trữ trên **bộ nhớ ngoài**. Cụ thể là ta sẽ sắp xếp dữ liệu được lưu trữ trong các tập tin.

Chương này tập trung giải quyết vấn đề sắp xếp trong còn sắp xếp ngoài sẽ được nghiên cứu trong chương IV.

2.2.3 Tổ chức dữ liệu và ngôn ngữ cài đặt

Các đối tượng cần được sắp xếp là các mẩu tin gồm một hoặc nhiều trường. Một trong các trường được gọi là khóa (key), kiểu của nó là một kiểu có quan hệ thứ tự (như các kiểu số nguyên, số thực, chuỗi ký tự...).

Danh sách các đối tượng cần sắp xếp sẽ là một mảng của các mẩu tin vừa nói ở trên. Mục đích của việc sắp xếp là tổ chức lại các mẩu tin sao cho các khóa của chúng được sắp thứ tự tương ứng với quy luật sắp xếp.

Để trình bày các ví dụ minh họa chúng ta sẽ dùng PASCAL làm ngôn ngữ thể hiện và sử dụng khai báo sau:

```
CONST N = 10;
TYPE
     KeyType = integer;
     OtherType = real;

     RecordType = Record
                  Key : KeyType;
                  OtherFields :  OtherType;
                  end;
VAR
     a : array[1..N] of RecordType;
```

```
PROCEDURE  Swap(var x,y:RecordType);
VAR  temp : RecordType;
BEGIN
     temp := x;
     x := y;
     y := temp;
END;
```

Cần thấy rằng thủ tục Swap lấy O(1) thời gian vì chỉ thực hiện 3 lệnh gán nối tiếp nhau.

2.3 CÁC PHƯƠNG PHÁP SẮP XẾP ĐƠN GIẢN

Các giải thuật đơn giản thường lấy $O(n^2)$ thời gian để sắp xếp n đối tượng và các giải thuật này thường chỉ dùng để sắp các danh sách có ít đối tượng.

Với mỗi giải thuật chúng ta sẽ nghiên cứu các phần: giải thuật, ví dụ, chương trình và phân tích đánh giá.

2.3.1 Sắp xếp chọn (Selection Sort)

2.3.1.1 Giải thuật

Đây là phương pháp sắp xếp đơn giản nhất được tiến hành như sau:

- Đầu tiên chọn phần tử có khóa nhỏ nhất trong n phần tử từ a[1] đến a[n] và hoán vị nó với phần tử a[1].
- Chọn phần tử có khóa nhỏ nhất trong n-1phần tử từ a[2] đến a[n] và hoán vị nó với a[2].
- Tổng quát ở bước thứ i, chọn phần tử có khoá nhỏ nhất trong n-i+1 phần tử từ a[i] đến a[n] và hoán vị nó với a[i].
- Sau n-1 bước này thì mảng đã được sắp xếp.

Phương pháp này được gọi là phương pháp chọn bởi vì nó lặp lại quá trình chọn phần tử nhỏ nhất trong số các phần tử chưa được sắp.

Ví dụ 2-1: Sắp xếp mảng gồm 10 mẩu tin có khóa là các số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3

Bước 1: Ta chọn được phần tử có khoá nhỏ nhất (bằng 2) trong các phần tử từ a[1] đến a[10] là a[3], hoán đổi a[1] và a[3] cho nhau. Sau bước này thì a[1] có khoá nhỏ nhất là 2.

Bước 2: Ta chọn được phần tử có khoá nhỏ nhất (bằng 2) trong các phần tử từ a[2] đến a[10] là a[4], hoán đổi a[2] và a[4] cho nhau.

Tiếp tục quá trình này và sau 9 bước thì kết thúc.

Bảng sau ghi lại các giá trị khoá tương ứng với từng bước.

Khóa / Bước	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	6	5	2	10	12	9	10	9	3
Bước 2		**2**	5	6	10	12	9	10	9	3
Bước 3			**3**	6	10	12	9	10	9	5
Bước 4				**5**	10	12	9	10	9	6
Bước 5					**6**	12	9	10	9	10
Bước 6						**9**	12	10	9	10
Bước 7							**9**	10	12	10
Bước 8								**10**	12	10
Bước 9									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Hình 2-1: Sắp xếp chọn

2.3.1.2 Chương trình:

```
PROCEDURE SelectionSort;
VAR
     i,j,LowIndex: integer;
     LowKey: KeyType;
BEGIN
{1}  FOR i := 1 TO  n-1 DO BEGIN
{2}       LowIndex := i;
{3}       LowKey := a[i].key;
{4}       FOR  j := i+1 TO  n  DO
{5}            IF a[j].key < LowKey THEN
               BEGIN
{6}                 LowKey := a[j].key;
{7}                 LowIndex := j;
               END;
{8}       Swap(a[i],a[LowIndex]);
     END;
END;
```

2.3.1.3 Đánh giá: Phương pháp sắp xếp chọn lấy $O(n^2)$ để sắp xếp n phần tử.

Trước hết ta có thủ tục Swap lấy một hằng thời gian như đã nói ở mục 2.2.3.

Các lệnh {2}, {3} đều lấy O(1) thời gian. Vòng lặp for {4} – {7} thực hiện n-i lần, vì j chạy từ i+1 đến n, mỗi lần lấy O(1), nên lấy O(n-i) thời gian. Do đó thời gian tổng cộng là:

$$T(n) = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2}$$ tức là $O(n^2)$.

2.3.2 Sắp xếp xen (Insertion Sort)

2.3.2.1 Giải thuật

Trước hết ta xem phần tử a[1] là một dãy đã có thứ tự.

- Bước 1, xen phần tử a[2] vào danh sách đã có thứ tự a[1] sao cho a[1], a[2] là một danh sách có thứ tự.
- Bước 2, xen phần tử a[3] vào danh sách đã có thứ tự a[1], a[2] sao cho a[1], a[2], a[3] là một danh sách có thứ tự.
- Tổng quát, bước i, xen phần tử a[i+1] vào danh sách đã có thứ tự a[1],a[2],..a[i] sao cho a[1], a[2],.. a[i+1] là một danh sách có thứ tự.
- Phần tử đang xét a[j] sẽ được xen vào vị trí thích hợp trong danh sách các phần tử đã được sắp trước đó a[1],a[2],..a[j-1] bằng cách so sánh khoá của a[j] với khoá của a[j-1] đứng ngay trước nó. Nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán đổi a[j-1] và a[j] cho nhau và tiếp tục so sánh khoá của a[j-1] (lúc này a[j-1] chứa nội dung của a[j]) với khoá của a[j-2] đứng ngay trước nó...

Ví dụ 2-2: Sắp xếp mảng gồm 10 mẩu tin đã cho trong ví dụ 2-1.

Bước 1: Xen a[2] vào dãy chỉ có một phần tử a[1] ta được dãy hai phần tử a[1]..a[2] có thứ tự. Việc xen này thực ra không phải làm gì cả vì hai phần tử a[1], a[2] có khoá tương ứng là 5 và 6 đã có thứ tự.

Bước 2: Xen a[3] vào dãy a[1]..a[2] ta được dãy ba phần tử a[1]..a[3] có thứ tự. Việc xen này được thực hiện bằng cách : so sánh khoá của a[3] với khoá của a[2], do khoá của a[3] nhỏ hơn khoá của a[2] (2<6) nên hoán đổi a[3] và a[2] cho nhau. Lại so sánh khoá của a[2] với khoá của a[1], do khoá của a[2] nhỏ hơn khoá của a[1] (2<5) nên hoán đổi a[2] và a[1] cho nhau.

Tiếp tục quá trình này và sau 9 bước thì kết thúc.

Bảng sau ghi lại các giá trị khoá tương ứng với từng bước.

Khóa / Bước	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	A[8]	a[9]	a[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**5**	**6**								
Bước 2	**2**	**5**	**6**							
Bước 3	**2**	**2**	**5**	**6**						
Bước 4	**2**	**2**	**5**	**6**	**10**					
Bước 5	**2**	**2**	**5**	**6**	**10**	**12**				
Bước 6	**2**	**2**	**5**	**6**	**9**	**10**	**12**			
Bước 7	**2**	**2**	**5**	**6**	**9**	**10**	**10**	**12**		
Bước 8	**2**	**2**	**5**	**6**	**9**	**9**	**10**	**10**	**12**	
Bước 9	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Hình 2-2: Sắp xếp xen

2.3.2.2 Chương trình

```
PROCEDURE InsertionSort;
VAR
     i,j: integer;
```

```
BEGIN
{1}  FOR i := 2 TO n DO BEGIN
{2}       J := i;
{3}       WHILE (j>1) AND (a[j].key < a[j-1].key) DO BEGIN
{4}            swap(a[j], a[j-1]);
{5}            j := j-1;
          END;
     END;
END;
```

2.3.2.3 Đánh giá: Phương pháp sắp xếp xen lấy $O(n^2)$ để sắp xếp n phần tử.

Ta thấy các lệnh {4} và {5} đều lấy O(1). Vòng lặp {3} chạy nhiều nhất i-1 lần, mỗi lần tốn O(1) nên {3} lấy i-1 thời gian. Lệnh {2} và {3} là hai lệnh nối tiếp nhau, lệnh {2} lấy O(1) nên cả hai lệnh này lấy i-1.

Vòng lặp {1} có i chạy từ 2 đến n nên nếu gọi T(n) là thời gian để sắp n phần tử thì ta có

$$T(n) = \sum_{i=2}^{n}(i-1) = \frac{n(n-1)}{2}$$ tức là $O(n^2)$.

2.3.3 Sắp xếp nổi bọt (Bubble Sort)

2.3.3.1 Giải thuật

Chúng ta tưởng tượng rằng các mẩu tin được lưu trong một mảng dọc, qua quá trình sắp, mẩu tin nào có khóa "nhẹ" sẽ được nổi lên trên. Chúng ta duyệt toàn mảng, từ dưới lên trên. Nếu hai phần tử ở cạnh nhau mà không đúng thứ tự tức là nếu phần tử "nhẹ hơn" lại nằm dưới thì phải cho nó "nổi lên" bằng cách đổi chỗ hai phần tử này cho nhau. Cụ thể là:

- Bước 1: Xét các phần tử từ a[n] đến a[2], với mỗi phần tử a[j], so sánh khoá của nó với khoá của phần tử a[j-1] đứng ngay trước nó. Nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán đổi a[j] và a[j-1] cho nhau.
- Bước 2: Xét các phần tử từ a[n] đến a[3], và làm tương tự như trên.
- Sau n-1 bước thì kết thúc.

Ví dụ 2-3: Sắp xếp mảng gồm 10 mẩu tin đã cho trong ví dụ 2-1.

Bước 1: Xét a[10] có khoá là 3, nhỏ hơn khoá của a[9] nên ta hoán đổi a[10] và a[9] cho nhau. Khoá của a[9] bây giờ là 3 nhỏ hơn khoá của a[8] nên ta hoán đổi a[9] và a[8] cho nhau. Khoá của a[8] bây giờ là 3 nhỏ hơn khoá của a[7] nên ta hoán đổi a[8] và a[7] cho nhau. Khoá của a[7] bây giờ là 3 nhỏ hơn khoá của a[6] nên ta hoán đổi a[7] và a[6] cho nhau. Khoá của a[6] bây giờ là 3 nhỏ hơn khoá của a[5] nên ta hoán đổi a[6] và a[5] cho nhau. Khoá của a[5] bây giờ là 3 **không nhỏ hơn** khoá của a[4] nên bỏ qua. Khoá của a[4] là 2 **không nhỏ hơn** khoá của a[3] nên bỏ qua. Khoá của a[3] là 2 nhỏ hơn khoá của a[2] nên ta hoán đổi a[3] và a[2] cho nhau. Khoá của a[2] bây giờ là 2 nhỏ hơn khoá của a[1] nên ta hoán đổi a[2] và a[1] cho nhau. Đến đây kết thúc bước 1 và a[1] có khoá nhỏ nhất là 2.

Bước 2: Xét a[10] có khoá là 9, nhỏ hơn khoá của a[9] nên ta hoán đổi a[10] và a[9] cho nhau. Khoá của a[9] bây giờ là 9 **không nhỏ hơn** khoá của a[8] nên bỏ qua. Khoá của a[8] là 9 nhỏ hơn khoá của a[7] nên ta hoán đổi a[8] và a[7] cho nhau. Khoá của a[7] bây giờ là 9 nhỏ hơn khoá của a[6] nên ta hoán đổi a[7] và a[6] cho nhau. Khoá của a[6] bây giờ là 9 **không nhỏ hơn** khoá của a[5] nên bỏ qua. Khoá của a[5] bây giờ là 3 **không nhỏ hơn** khoá của a[4] nên bỏ qua. Khoá của a[4] là 2 nhỏ hơn khoá của a[3] nên ta hoán đổi a[4] và a[3] cho nhau. Khoá của a[3] bây giờ là 2 nhỏ hơn khoá của a[2] nên ta hoán đổi a[3] và a[2] cho nhau. Đến đây kết thúc bước 2 và a[2] có khoá là 2.

Tiếp tục quá trình này và sau 9 bước thì kết thúc.

Bảng sau ghi lại các giá trị khoá tương ứng với từng bước.

Khóa / Bước	a[1]	a[2]	a[3]	A[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	5	6	2	3	10	12	9	10	9
Bước 2		**2**	5	6	3	9	10	12	9	10
Bước 3			**3**	5	6	9	9	10	12	10
Bước 4				**5**	6	9	9	10	10	12
Bước 5					**6**	9	9	10	10	12
Bước 6						**9**	9	10	10	12
Bước 7							**9**	10	10	12
Bước 8								**10**	10	12
Bước 9									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Hình 2-3: Sắp xếp nổi bọt

2.3.3.2 Chương trình

```
PROCEDURE BubbleSort;
VAR
     i,j: integer;
BEGIN
{1}  FOR i := 1 to  n-1 DO
{2}       FOR j :=  n DOWNTO i+1 DO
{3}            IF  a[j].key < a[j-1].key THEN
{4}                 Swap(a[j],a[j-1]);
END;
```

2.3.3.3 Đánh giá: Phương pháp sắp xếp nổi bọt lấy $O(n^2)$ để sắp n phần tử.

Dòng lệnh {3} lấy một hằng thời gian. Vòng lặp {2} thực hiện (n-i) bước, mỗi bước lấy O(1) nên lấy O(n-i) thời gian. Như vậy đối với toàn bộ chương trình ta có:

$$T(n)= \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2).$$

2.4 QUICKSORT

Trong phần này chúng ta sẽ nghiên cứu một giải thuật sắp xếp được dùng một cách phổ biến là Quick Sort do A.R. Hoare phát minh vào năm 1960. Quick Sort đã được cải tiến để trở thành phương pháp được chọn trong các ứng dụng sắp xếp thực tế khác nhau.

2.4.1 Ý tưởng

Chúng ta vẫn xét mảng a các mẩu tin a[1]..a[n]. Giả sử v là 1 giá trị khóa mà ta gọi là chốt (pivot). Ta phân hoạch dãy a[1]..a[n] thành hai mảng con "bên trái" và "bên phải". Mảng con "bên trái" bao gồm các phần tử có **khóa nhỏ hơn chốt**, mảng con "bên phải" bao gồm các phần tử có **khóa lớn hơn hoặc bằng chốt**.

Sắp xếp mảng con “bên trái” và mảng con “bên phải” thì mảng đã cho sẽ được sắp bởi vì tất cả các khóa trong mảng con “bên trái“ đều nhỏ hơn các khóa trong mảng con “bên phải”.

Việc sắp xếp các mảng con “bên trái” và “bên phải” cũng được tiến hành bằng phương pháp nói trên.

Một mảng chỉ gồm một phần tử hoặc gồm nhiều phần tử có khóa bằng nhau thì đã có thứ tự.

2.4.2 Thiết kế giải thuật

2.4.2.1 Vấn đề chọn chốt

Chọn **khóa lớn nhất** trong hai phần tử có khóa khác nhau đầu tiên kể từ trái qua. Nếu mảng chỉ gồm một phần tử hay gồm nhiều phần tử có khóa bằng nhau thì không có chốt.

Ví dụ 2-5: Chọn chốt trong các mảng sau

Cho mảng gồm các phần tử có khoá là 6, 6, 5, 8, 7, 4, ta chọn chốt là 6 (khoá của phần tử đầu tiên).

Cho mảng gồm các phần tử có khoá là 6, 6, 7, 5, 7, 4, ta chọn chốt là 7 (khoá của phần tử thứ 3).

Cho mảng gồm các phần tử có khoá là 6, 6, 6, 6, 6, 6 thì không có chốt (các phần tử có khoá bằng nhau).

Cho mảng gồm một phần tử có khoá là 6 thì không có chốt (do chỉ có một phần tử).

2.4.2.2 Vấn đề phân hoạch

Để phân hoạch mảng ta dùng 2 "con nháy" L và R trong đó L từ bên trái và R từ bên phải, ta cho L chạy sang phải cho tới khi gặp phần tử có khóa ≥ chốt và cho R chạy sang trái cho tới khi gặp phần tử có khóa < chốt. Tại chỗ dừng của L và R nếu L < R thì hoán vị a[L],a[R]. Lặp lại quá trình dịch sang phải, sang trái của 2 "con nháy" L và R cho đến khi L > R. Khi đó L sẽ là điểm phân hoạch, cụ thể là a[L] là phần tử đầu tiên của mảng con “bên phải”.

2.4.2.3 Giải thuật QuickSort

Để sắp xếp mảng a[i]..a[j] ta tiến hành các bước sau:

- Xác định chốt.
- Phân hoạch mảng đã cho thành hai mảng con a[i]..a[k-1] và a[k]..a[j].
- Sắp xếp mảng a[i]..a[k-1] (Đệ quy).
- Sắp xếp mảng a[k]..a[j] (Đệ quy).

Quá trình đệ quy sẽ dừng khi không còn tìm thấy chốt.

Ví dụ 2-4: Sắp xếp mảng gồm 10 mẩu tin có khóa là các số nguyên: 5, 8, 2, 10, 5, 12, 8, 1, 15 và 4.

Với mảng a[1]..a[10], hai phần tử đầu tiên có khóa khác nhau là là a[1] và a[2] với khoá tương ứng là 5 và 8, ta chọn chốt v = 8.

Để phân hoạch, khởi đầu ta cho L := 1 (đặt L ở cực trái) và R := 10 (đặt R ở cực phải). Do a[L] có khoá là 5 nhỏ hơn chốt nên L := L+1 = 2 (di chuyển L sang phải), lúc này a[L] có khoá là 8 = chốt nên dừng lại. Do a[R] có khoá là 4 nhỏ hơn chốt nên R cũng không chuyển sang trái được. Tại các điểm dừng của L và R ta có L < R (L=2 và R=10) nên hoán đổi a[L] và a[R] (a[2] và a[10]) cho nhau. Sau khi hoán đổi, a[L] lại có khoá là 4 nhỏ hơn chốt nên di chuyển L sang phải (L := L+1 = 3). Khoá của a[L] là 2 nhỏ hơn chốt nên lại di chuyển L sang phải (L := L+1 = 4). Khoá của a[L] là 10 lớn hơn chốt nên dừng lại. Với R, khoá của a[R] bây giờ là 8 bằng chốt nên di chuyển R sang trái (R := R-1 = 9). Khoá của a[R] là 15 lớn hơn chốt nên di chuyển R sang trái (R := R-1 = 8). Khoá của a[R] là 1 nhỏ hơn chốt nên dừng lại. Tại các điểm dừng của L và R ta có L < R (L=4 và R=8) nên hoán đổi a[L] và a[R] (a[4] và a[8]) cho nhau. Sau khi hoán đổi, a[L] có khoá là 1 nhỏ hơn chốt nên di chuyển L sang phải (L := L+1 = 5). Khoá của a[L] là 5 nhỏ hơn chốt nên lại di chuyển L sang phải (L := L+1 = 6). Khoá của a[L] là 12 lớn hơn chốt nên dừng lại. Với R, khoá của a[R] bây giờ là 10 lớn hơn chốt nên di chuyển R sang trái (R := R-1 = 7). Khoá của a[R] là 8 bằng chốt nên di chuyển R sang trái (R := R-1 = 6). Khoá của a[R] là 12 lớn hơn chốt nên di chuyển R sang trái (R := R-1 = 5). Khoá của a[R] là 5 nhỏ hơn chốt nên dừng lại. Tại các điểm dừng của L và R ta có L > R (L=6 và R=5) nên ta đã xác định được điểm phân hoạch ứng với L = 6. Tức là mảng đã cho ban đầu được phân thành hai mảng con bên trái a[1]..a[5] và mảng con bên phải a[6]..a[10]. Hình ảnh của sự phân hoạch này được biểu diễn trong hình sau:

Chỉ số	1	2	3	4	5	6	7	8	9	10
Khoá Ban đầu	5	8 4	2	10 1	5	12	8	1 10	15	4 8
					v = 8					
Cấp 1	5	4	2	1	5	12	8	10	15	8

Hình 2-4 : Chọn chốt và phân hoạch mảng a[1]..a[10]

Trong bảng trên, dòng *chỉ số* ghi các chỉ số của các phần tử của mảng (từ 1 đến 10).

Trong dòng *khoá ban đầu*, các giá trị khoá ở dòng trên (5, 8, 2, 10, 5, 12, 8, 1, 15 và 4) là các giá trị khoá của mảng đã cho ban đầu, các giá trị khoá ở dòng dưới (4, 1, 10 và 8) là các giá trị khoá mới sau khi thực hiện hoán đổi a[2] với a[10] và a[4] với a[8].

Giá trị chốt là v = 8.

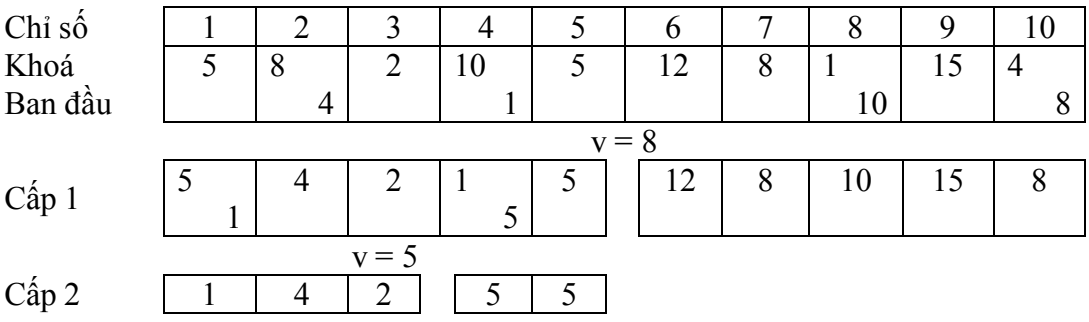
Dòng cấp c*ấp 1*, biểu diễn hai mảng con sau khi phân hoạch. Mảng bên trái từ a[1] đến a[5] gồm các phần tử có khoá là 5, 4, 2, 1 và 5. Mảng con bên phải từ a[6] đến a[10] gồm các phần tử có khoá 12, 8, 10, 15 và 8.

Tiếp tục sắp xếp đệ quy cho mảng con bên trái và mảng con bên phải.

Với mảng con bên trái a[1]..a[5], hai phần tử đầu tiên có khóa khác nhau là là a[1] và a[2] với khoá tương ứng là 5 và 4, ta chọn chốt v = 5.

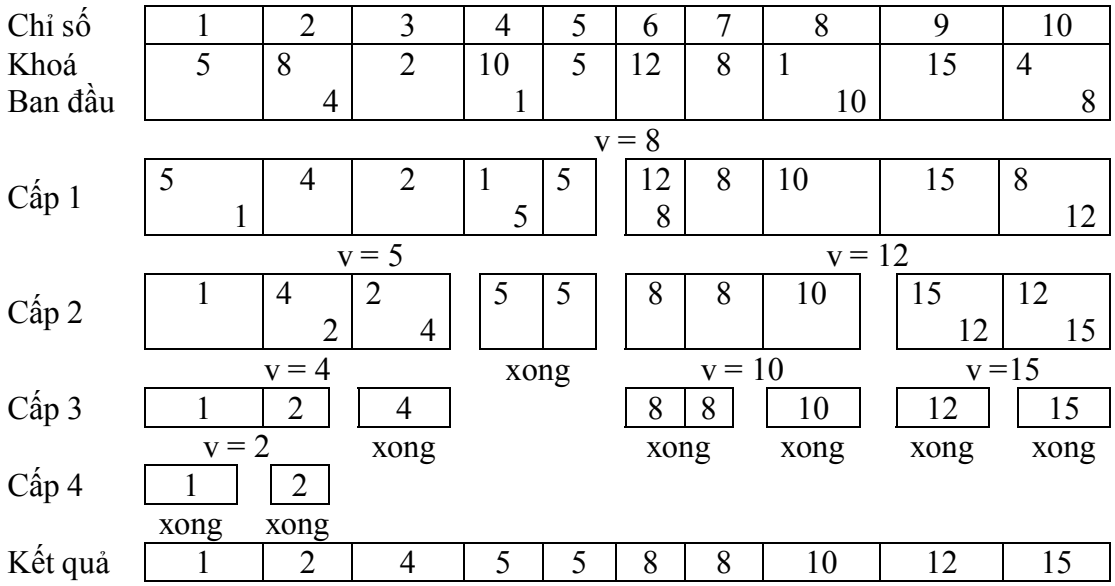
Để phân hoạch, khởi đầu ta cho L := 1 (đặt L ở cực trái) và R := 5 (đặt R ở cực phải). Do a[L] có khoá là 5 bằng chốt nên không thể di chuyển L. Do a[R] có khoá là 5 bằng chốt nên di chuyển R sang trái (R := R-1 = 4). Khoá của a[R] bây giờ là 1 nhỏ hơn chốt nên dừng lại. Tại các điểm dừng của L và R ta có L < R (L= và R=4) nên hoán đổi a[L] và a[R] (a[1] và a[4]) cho nhau. Sau khi hoán đổi, a[L] lại có khoá là 1 nhỏ hơn chốt nên di chuyển L sang phải (L := L+1 = 2). Khoá của a[L] là 4 nhỏ hơn chốt nên lại di chuyển L sang phải (L := L+1 = 3). Khoá của a[L] là 2 nhỏ hơn chốt nên lại di chuyển L sang phải (L := L+1 = 4). Khoá của a[L] là 5 bằng chốt nên dừng lại. Với R, khoá của a[R] bây giờ là 5 bằng chốt nên di chuyển R sang trái (R := R-1 = 4). Khoá của a[R] là 5 bằng chốt nên di chuyển R sang trái (R := R-1 = 3). Khoá của a[R] là 2 nhỏ hơn chốt nên dừng lại. Tại các điểm dừng của L và R ta có L > R (L=4 và R=3) nên ta đã xác định được điểm phân hoạch ứng với L = 4. Tức là mảng bên trái phân thành hai mảng con bên trái a[1]..a[3] và mảng con bên phải a[4]..a[6].

Hình ảnh của sự phân hoạch này được biểu diễn trong hình sau:



Hình 2-5 : Chọn chốt và phân hoạch mảng a[1]..a[5]

Tiếp tục sắp xếp cho các mảng con của cấp 1 và mảng con bên phải của mảng ban đầu cho đến khi dừng (các mảng không có chốt). Cuối cùng ta có mảng được sắp thứ tự. Hình sau biểu diễn toàn bộ quá trình sắp xếp.



Hình 2-6 : QuickSort

2.4.3 Cài đặt giải thuật

2.4.3.1 Hàm FindPivot

Ta thiết kế hàm FindPivot để xác định trong dãy a[i]..a[j] có hay không hai phần tử có khóa khác nhau. Nếu không tìm thấy hai phần tử có khóa khác nhau thì trả về giá trị 0 (không tìm thấy chốt), ngược lại hàm trả về giá trị là chỉ số của phần tử có khóa lớn hơn trong hai phần tử có khóa khác nhau đầu tiên. Khóa lớn hơn này sẽ trở thành phần tử chốt mà ta sẽ xác định trong thủ tục QuickSort.

Để tiện so sánh ta sử dụng biến FirstKey để lưu giữ khóa của phần tử đầu tiên trong mảng a[i]..a[j] (FirstKey chính là a[i].key).

Ta sẽ dùng một chỉ số k để dò tìm trong mảng a[i]..a[j], kể từ vị trí i+1 đến hết mảng, một phần tử a[k] mà a[k].key <> FirstKey. Nếu không tìm thấy một a[k] như thế thì hoặc là mảng chỉ gồm một phần tử hoặc gồm nhiều phần tử có khóa bằng nhau. Trong trường hợp đó thì không tìm thấy chốt và hàm FindPivot sẽ trả về 0. Ngược lại ta sẽ phải xét xem a[k].key có lớn hơn FirstKey hay không, nếu đúng như thế thì chốt sẽ là khóa của a[k] và hàm FindPivot sẽ trả về k, nếu không thì chốt sẽ là khoá của a[i] và hàm FindPivot sẽ trả về i.

```
FUNCTION FindPivot(i,j:integer): integer;
VAR  FirstKey : KeyType;
     k : integer;
BEGIN
{1}    k := i+1;
{2}    FirstKey := a[i].key;
{3}    WHILE (k <= j) AND (a[k].key = FirstKey) DO k:= k+1;
{4}    IF k > j THEN FindPivot := 0
       ELSE
```

```
{5}       IF a[k].key > FirstKey THEN FindPivot := k
       ELSE FindPivot := i;
END;
```

Trong hàm FindPivot các lệnh {1}, {2}, {3} và {4} nối tiếp nhau, trong đó chỉ có lệnh WHILE là tốn nhiều thời gian nhất do đó thời gian thực hiện của hàm FindPivot phụ thuộc vào thời gian thực hiện của lệnh này. Trong trường hợp xấu nhất (không tìm thấy chốt) thì k chạy từ i+1 đến j, tức là vòng lặp thực hiện j-i lần, mỗi lần O(1) do đó tốn j-i thời gian. Đặc biệt khi i=1 và j=n, thì thời gian thực hiện là n-1 hay T(n) = O(n).

2.4.3.2 Hàm Partition

Hàm Partition nhận vào ba tham số i, j và Pivot để thực hiện việc phân hoạch mảng a[i]..a[j] theo chốt Pivot và trả về giá trị L là chỉ số đầu tiên của mảng "bên phải".

Hai con nháy L, R sẽ được sử dụng để thực hiện việc phân hoạch như đã trình bày trong phần 2.4.2.3.

```
FUNCTION Partition(i,j:integer; pivot :KeyType):integer ;
VAR  L,R : integer;
BEGIN
{1}  L := i;  {Đặt con nháy L ở cực trái}
{2}  R := j;  {Đặt con nháy R ở cực phải}
{3}  WHILE L <= r DO BEGIN
          {L tiến sang phải}
{4}       WHILE a[L].key < pivot DO L := L+1;
          {R tiến sang trái}
{5}       WHILE a[R].key >= pivot DO R := R-1;
{6}       IF L < R THEN Swap(a[L],a[R]);
     END;
{7}  Partition := L;  {Trả về điểm phân hoạch}
END;
```

Trong hàm Partition các lệnh {1}, {2}, {3} và {7} nối tiếp nhau, trong đó thời gian thực hiện của lệnh {3} là lớn nhất, do đó thời gian thực hiện của lệnh {3} sẽ là thời gian thực hiện của hàm Partition. Các lệnh {4}, {5} và {6} là thân của lệnh {3}, trong đó lệnh {6} lấy O(1) thời gian. Lệnh {4} và lệnh {5} thực hiện việc di chuyển L sang phải và R sang trái, thực chất là duyệt các phần tử mảng, mỗi phần tử một lần, mỗi lần tốn O(1) thời gian. Tổng cộng việc duyệt này tốn j-i thời gian. Vòng lặp {3} thực chất là để xét xem khi nào thì duyệt xong, do đó thời gian thực hiện của lệnh {3} chính là thời gian thực hiện của hai lệnh {4} và {5} và do đó là j-i. Đặc biệt khi i=1 và j=n ta có T(n) = O(n).

2.4.3.3 Thủ tục QuickSort

Bây giờ chúng ta trình bày thủ tục cuối cùng có tên là QuickSort và chú ý rằng để sắp xếp mảng A các record gồm n phần tử của kiểu Recordtype ta chỉ cần gọi QuickSort(1,n).

Ta sẽ sử dụng biến PivotIndex để lưu giữ kết quả trả về của hàm FindPivot, nếu biến PivotIndex nhận được một giá trị khác 0 thì mới tiến hành phân hoạch mảng.

Ngược lại, mảng không có chốt và do đó đã có thứ tự. Biến Pivot sẽ được sử dụng để lưu giữ giá trị chốt và biến k để lưu giữ giá trị của điểm phân hoạch do hàm Partition trả về. Sau khia đã phân hoạch xong ta sẽ gọi đệ quy QuickSort cho mảng con "bên trái" a[i]..a[k-1] và mảng con "bên phải" a[k]..a[j].

```
PROCEDURE Quicksort(i,j:integer);
VAR
   Pivot : KeyType;
   PivotIndex, k : integer;
BEGIN
     PivotIndex := FindPivot(i,j);
     IF PivotIndex <> 0 THEN BEGIN
          Pivot := a[PivotIndex].key;
          k := Partition(i,j,Pivot);
          QuickSort(i,k-1);
          QuickSort(k,j);
     END;
END;
```

2.4.4 Thời gian thực hiện của QuickSort

QuickSort lấy O(nlogn) thời gian để sắp xếp n phần tử trong trường hợp tốt nhất và $O(n^2)$. trong trường hợp xấu nhất.

Giả sử các giá trị khóa của mảng khác nhau nên hàm FindPivot luôn tìm được chốt và đệ quy chỉ dừng khi kích thước bài toán bằng 1.

Gọi T(n) là thời gian thức hiện việc QuickSort mảng có n phần tử.

Thời gian để tìm chốt và phân hoạch mảng như đã phân tích trong các phần 2.4.3.1 và 2.4.3.2 đều là O(n) = n.

Khi n = 1, thủ tục QuickSort chỉ làm một nhiệm vụ duy nhất là gọi hàm Findpivot với kích thước bằng 1, hàm này tốn thời gian O(1) =1.

Trong trường hợp xấu nhất là ta luôn chọn phải phần tử có khóa lớn nhất làm chốt, lúc bấy giờ việc phân hoạch bị lệch tức là mảng bên phải chỉ gồm một phần tử chốt, còn mảng bên trái gồm n-1 phần tử còn lại. Khi đó ta có thể thành lập phương trình đệ quy như sau:

$$T(n) = \begin{cases} 1 & \text{nêu } n = 1 \\ T(n-1) + T(1) + n & \text{nêu } n > 1 \end{cases}$$

Giải phương trình này bằng phương pháp truy hồi

Ta có $T(n) = T(n-1) + T(1) + n = T(n-1) + (n+1)$

$= [T(n-2) + T(1) + (n-1)] + (n+1) = T(n-2) + n + (n+1)$

$= [T(n-3) + T(1) + (n-2)] + n + (n+1) = T(n-3) + (n-1) + n + (n+1)$

. . . . . .

$$T(n) = T(n-i) + (n-i+2) + (n-i+3) + \ldots + n + (n+1) = T(n-i) + \sum_{j=n-i+2}^{n+1} j$$

Quá trình trên kết thúc khi i = n-1, khi đó ta có $T(n) = T(1) + \sum_{j=3}^{n+1} j = 1 + \sum_{j=3}^{n+1} j$

$= \sum_{j=1}^{n+1} j - 2 = \frac{n^2 + 3n - 2}{2} = O(n^2)$

Trong trường hợp tốt nhất khi ta chọn được chốt sao cho hai mảng con có kích thước bằng nhau và bằng n/2. Lúc đó ta có phương trình đệ quy như sau:

$$T(n) = \begin{cases} 1 & \text{nêu } n = 1 \\ 2T(\frac{n}{2}) + n & \text{nêu } n > 1 \end{cases}$$

Giải phương trình đệ quy này ta được T(n) = O(nlogn).

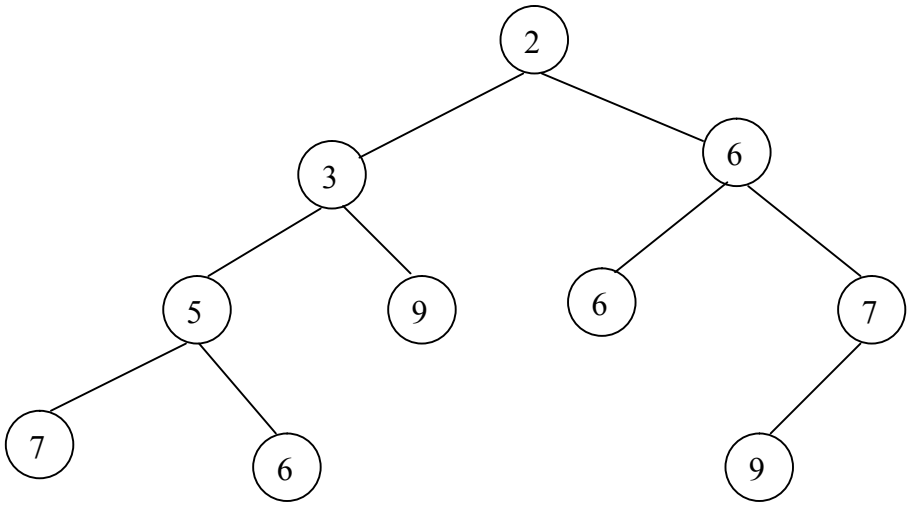
2.5 HEAPSORT

2.5.1 Định nghĩa Heap

Cây sắp thứ tự bộ phận hay còn gọi là heap là cây nhị phân mà giá trị tại mỗi nút (khác nút lá) đều không lớn hơn giá trị của các con của nó.

Ta có nhận xét rằng nút gốc a[1] của cây sắp thứ tự bộ phận có giá trị nhỏ nhất.

Ví dụ 2-5: Cây sau là một heap.



Hình 2-7: Một heap

2.5.2 Ý tưởng

(1) Xem mảng ban đầu là một cây nhị phân. Mỗi nút trên cây lưu trữ một phần tử mảng, trong đó a[1] là nút gốc và mỗi nút không là nút lá a[i] có con trái là a[2i] và con phải là a[2i+1]. Với cách tổ chức này thì cây nhị phân thu được sẽ có các nút trong là các nút a[1],..,a[n DIV 2]. Tất cả các nút trong đều có 2 con, ngoại trừ nút a[n DIV 2] có thể chỉ có một con trái (trong trường hợp n là một số chẵn).

(2) Sắp xếp cây ban đầu thành một heap căn cứ vào giá trị khoá của các nút.

(3) Hoán đổi a[1] cho cho phần tử cuối cùng.

(4) Sắp lại cây sau khi đã bỏ đi phần tử cuối cùng để nó trở thành một heap mới.

Lặp lại quá trình (3) và (4) cho tới khi cây chỉ còn một nút ta sẽ được mảng sắp theo thứ tự giảm.

2.5.3 Thiết kế và cài đặt giải thuật

2.5.3.1 Thủ tục PushDown

Thủ tục PushDown nhận vào 2 tham số first và last để đẩy nút first xuống.

Giả sử a[first],..,a[last] đã đúng vị trí (giá trị khoá tại mỗi nút nhỏ hơn hoặc bằng giá trị khoá tại các nút con của nó) ngoại trừ a[first]. PushDown dùng để đẩy phần tử a[first] xuống đúng vị trí của nó trong cây (và có thể gây ra việc đẩy xuống các phần tử khác).

Xét a[first], có các khả năng có thể xẩy ra:

- Nếu a[firrst] chỉ có một con trái và nếu khoá của nó lớn hơn khoá của con trái (a[first].key > a[2*first].key) thì hoán đổi a[first] cho con trái của nó và kết thúc.
- Nếu a[first] có khoá lớn hơn con trái của nó (a[first].key > a[2*first].key) và khoá của con trái không lớn hơn khoá của con phải (a[2*first].key <= a[2*first+1].key) thì hoán đổi a[first] cho con trái a[2*first] của nó, việc này có thể gây ra tình trạng con trái sẽ không đúng vị trí nên phải xem xét lại con trái để có thể đẩy xuống.
- Ngược lại, nếu a[first] có khoá lớn hơn khoá của con phải của nó (a[first].key > a[2*first+1].key) và khoá của con phải nhỏ hơn khoá của con trái (a[2*first+1].key < a[2*first].key) thì hoán đổi a[first] cho con phải a[2*first+1] của nó, việc này có thể gây ra tình trạng con phải sẽ không đúng vị trí nên phải tiếp tục xem xét con phải để có thể đẩy xuống.
- Nếu tất cả các trường hợp trên đều không xẩy ra thì a[first] đã đúng vị trí.

Như trên ta thấy việc đẩy a[first] xuống có thể gây ra việc đẩy xuống một số phần tử khác, nên tổng quát là ta sẽ xét việc đẩy xuống của một phần tử a[r] bất kỳ, bắt đầu từ a[first].

```
PROCEDURE PushDown(first,last:integer);
VAR  r:integer;
BEGIN
  r:= first;  {Xét nút a[first] trước hết}
  WHILE r <= last DIV 2 DO
   If last = 2*r THEN BEGIN {nút r chỉ có con trái }
    IF a[r].key > a[last].key THEN swap(a[r],a[last]);
    r:=last;  {Kết thúc}
   END ELSE
    IF (a[r].key>a[2*r].key)and(a[2*r].key<= a[2*r+1].key)
    THEN BEGIN
           swap(a[r],a[2*r]);
           r := 2*r ; {Xét tiếp nút con trái }
         END
    ELSE
      IF (a[r].key>a[2*r+1].key)and(a[2*r+1].key<a[2*r].key)
      THEN BEGIN
             swap(a[r],a[2*r+1]);
             r := 2*r+1 ; {Xét tiếp nút con phải }
           END
       ELSE
             r := last; {Nút r đã đúng vị trí }
END;
```

Thủ tục PushDown chỉ duyệt trên một nhánh nào đó của cây nhị phân, tức là sau mỗi lần lặp thì số nút còn lại một nửa. Nếu số nút lúc đầu là n, trong trường hợp xấu nhất (luôn phải thực hiện việc đẩy xuống) thì lệnh lặp WHILE phải thực hiện i lần sao cho $2^i = n$ tức là i = logn. Mà mỗi lần lặp chỉ thực hiện một lệnh IF với thân lệnh IF là gọi thủ tục Swap và gán, do đó tốn O(1) = 1 đơn vị thời gian. Như vậy thủ tục PushDown lấy O(logn) để đẩy xuống một nút trong cây có n nút.

2.5.3.2 Thủ tục HeapSort

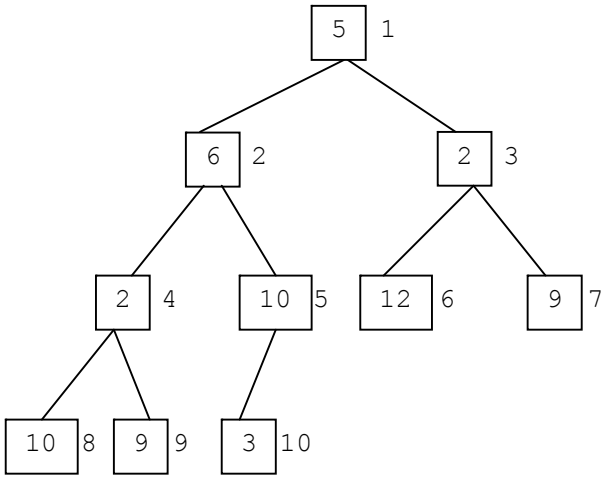
- Việc sắp xếp cây ban đầu thành một heap được tiến hành bằng cách sử dụng thủ tục PushDown để đẩy tất cả các nút trong chưa đúng vị trí xuống đúng vị trí của nó, khởi đầu từ nút a[n DIV 2], lần ngược tới gốc.
- Lặp lại việc hoán đổi a[1] cho a[i], sắp xếp cây a[1]..a[i-1] thành heap, i chạy từ n đến 2.

```
PROCEDURE HeapSort;
VAR   i:integer;
BEGIN
{1}  FOR i := (n div 2) DOWNTO 1 DO
{2}       PushDown(i,n);
{3}  FOR i := n DOWNTO 2 DO BEGIN
{4}       swap(a[1],a[i]);
{5}       pushdown(1,i-1);
     END;
END;
```

Ví dụ 2-6: Sắp xếp mảng bao gồm 10 phần tử có khoá là các số nguyên như trong các ví dụ 2.1:

Chỉ số	1	2	3	4	5	6	7	8	9	10
Khoá ban đầu	5	6	2	2	10	12	9	10	9	3

Mảng này được xem như là một cây nhị phân ban đầu như sau:

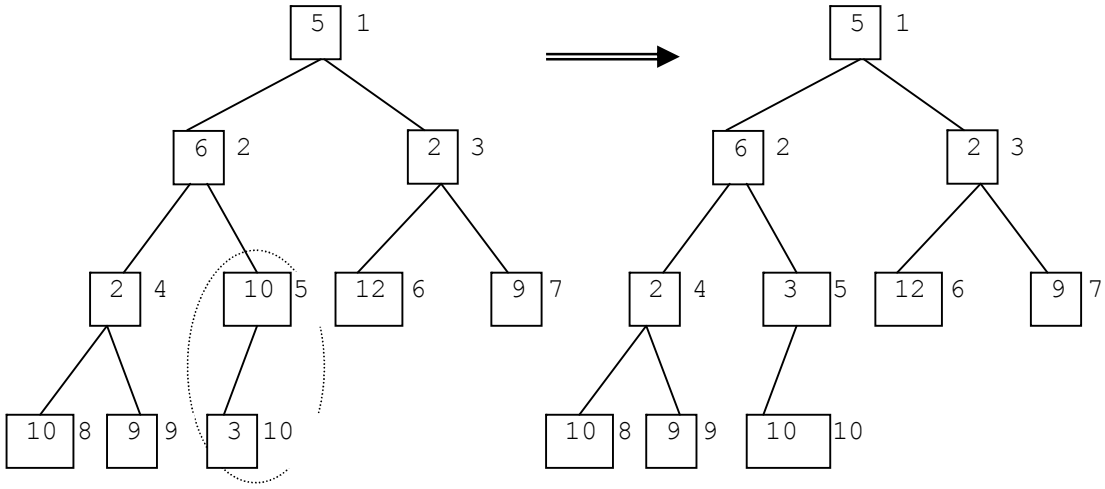


Hình 2-8: Cây ban đầu

Trong cây trên, giá trị ghi trong các nút là khoá của các phần tử mảng, giá trị ghi bên ngoài các nút là chỉ số của các phần tử mảng.

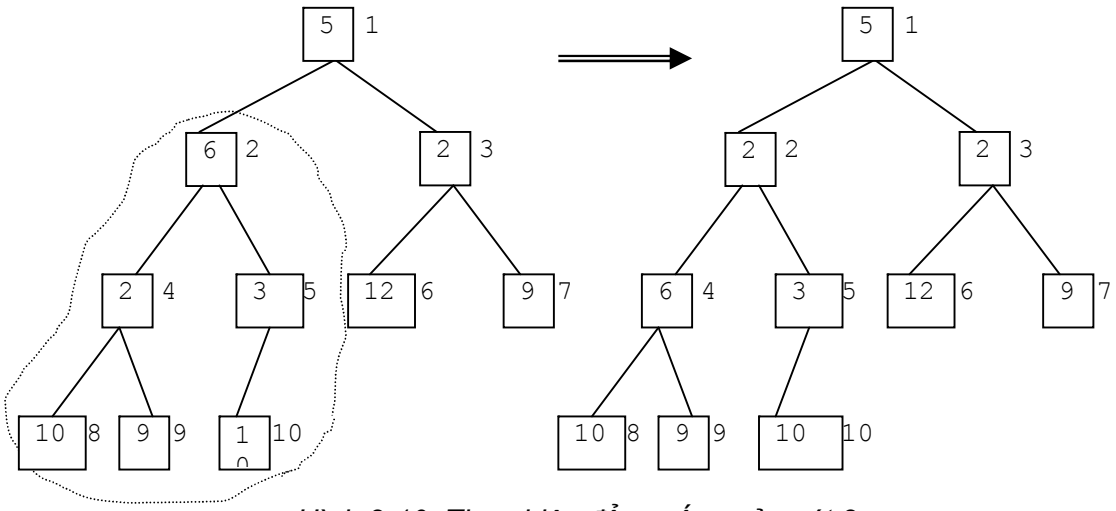
Việc sắp xếp cây này thành một heap sẽ bắt đầu từ việc đẩy xuống nút a[5] (vì 5 = 10 DIV 2)

Xét nút 5 ta thấy a[5] chỉ có một con trái và giá trị khóa tương ứng của nó lớn hơn con trái của nó nên ta đổi hai nút này cho nhau và do con trái của a[5] là a[10] là một nút lá nên việc đẩy xuống của a[5] kết thúc.



Hình 2-9: Thực hiện đẩy xuống của nút 5

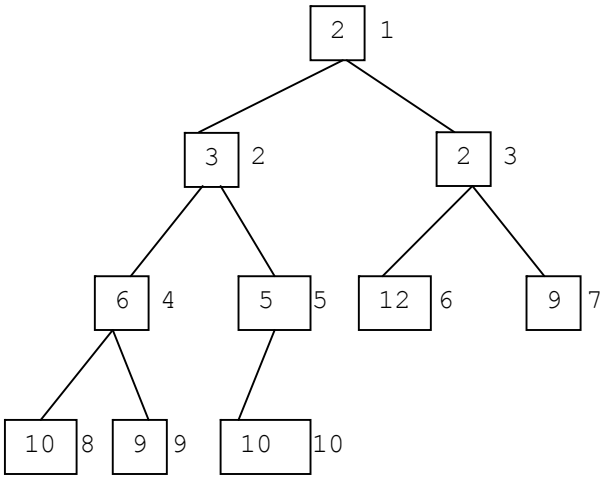
Nút 4 và nút 3 đã đúng vị trí nên không phải thực hiện sự hoán đổi. Tại nút 2, giá trị khóa của nó lớn hơn khoá con trái và khoá của con trái nhỏ hơn khoá của con phải nên ta hóan đổi nút 2 cho con trái của nó (nút 4), sau khi hoán đổi, ta xét lại nút 4, thấy nó vẫn đúng vị trí nên kết thúc việc đẩy xuống của nút 2.



Hình 2-10: Thực hiện đẩy xuống của nút 2

Cuối cùng ta xét nút 1, ta thấy giá trị khoá của nút 1 lớn hơn khoá của con trái và con trái có khoá bằng khoá của con phải nên hóan đổi nút 1 cho con trái của nó (nút 2).

Sau khi thực hiện phép hóan đổi nút 1 cho nút 2, ta thấy nút 2 có giá trị khoá lớn hơn khoá của con phải của nó (nút 5) và con phải có khoá nhỏ hơn khoá của con trái nên phải thực hiện phép hoán đổi nút 2 cho nút 5. Xét lại nút 5 thì nó vẫn đúng vị trí nên ta được cây mới trong hình 2-11.



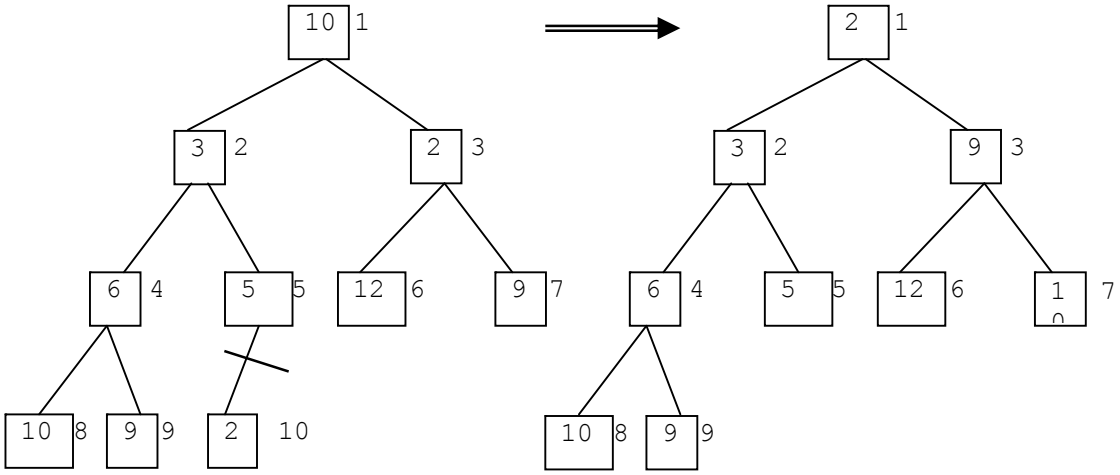
Hình 2-11: Cây ban đầu đã được tạo thành heap

Cây này là một heap tương ứng với mảng

Chỉ số	1	2	3	4	5	6	7	8	9	10
Heap	2	3	2	6	5	12	9	10	9	10

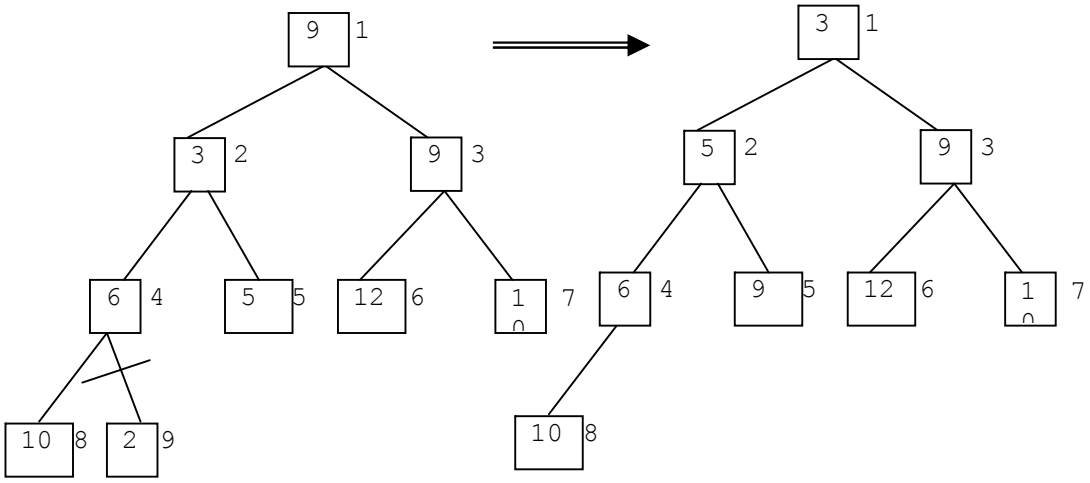
Từ heap đã có ở trên, hoán đổi a[1] cho a[10] ta có a[10] là nút có khóa nhỏ nhất, cắt bỏ nút a[10] ra khỏi cây. Như vậy phần cuối mảng chỉ gồm một phần tử a[10] đã được sắp.

Thực hiện việc đẩy a[1] xuống đúng vị trí của nó trong cây a[1]..a[9] ta được cây:



Hình 2-12: Hoán đổi a[1] cho a[10] và đẩy a[1] xuống trong a[1..9]

Hoán đổi a[1] cho a[9] và cắt a[9] ra khỏi cây. Ta được phần cuối mảng bao gồm hai phần tử a[9]..a[10] đã được sắp. Thực hiện việc đẩy a[1] xuống đúng vị trí của nó trong cây a[1]..a[8] ta được cây



Hình 2-13: Hoán đổi a[1] cho a[9] và đẩy a[1] xuống trong a[1..8]

Tiếp tục quá trình trên ta sẽ được một mảng có thứ tự giảm.

Trình bày heapsort bằng mảng

Như trong phần ý tưởng đã nói, chúng ta chỉ xem mảng như là một cây. Điều đó có nghĩa là các thao tác thực chất vẫn là các thao tác trên mảng. Để hiểu rõ hơn, ta sẽ trình bày ví dụ trên sử dụng mô hình mảng.

Mảng của 10 mẩu tin, có khoá là các số nguyên đã cho là:

Chỉ số	1	2	3	4	5	6	7	8	9	10
Khoá ban đầu	5	6	2	2	10	12	9	10	9	3

Mặc dù không vẽ thành cây, nhưng ta vẫn tưởng tượng mảng này như là một cây nhị phân với nút gốc là a[1], các nút a[i] có con trái là a[2i] và on phải là a[2i+1]. Chỉ có các nút từ a[1] đến a[5] là nút trong, còn các nút từ a[6] đến a[10] là nút lá.

Từ mảng ban đầu, chúng ta sẽ tạo thành heap bằng cách áp dụng thủ tục PushDown từ a[5] đến a[1].

Xét a[5], nút này chỉ có một con trái là a[10] và khoá của a[5] lớn hơn khoá của a[10] (10 > 3) nên đẩy a[5] xuống (hoán đổi a[5] và a[10] cho nhau).

Xét a[4], nút này có hai con là a[8] và a[9] và khoá của nó đều nhỏ hơn khoá của hai con (2 < 10 và 2 < 9) nên không phải đẩy xuống.

Tương tự a[3] cũng không phải đẩy xuống.

Xét a[2], nút này có con trái là a[4] và con phải là a[5]. Khoá của a[2] lớn hơn khoá của con trái (6 > 2) và khoá của con trái nhỏ hơn khoá của con phải (2 < 3) do đó đẩy a[2] xuống bên trái (hoán đổi a[2] và a[4] cho nhau). Tiếp tục xét con trái của a[2], tức là a[4]. Khoá của a[4] bây giờ là 6, nhỏ hơn khoá của con trái a[8] (6 < 10) và khoá của con phải a[9] (6 < 9) nên không phải đẩy a[4] xuống.

Xét a[1], nút này có con trái là a[2] và con phải là a[3]. Khoá của a[1] lớn hơn khoá của con trái a[2] (5 > 2) và khoá của con trái bằng khoá của con phải (2 = 2) nên đẩy a[1] xuống bên trái (hoán đổi a[1] và a[2] cho nhau). Tiếp tục xét con trái a[2]. Nút này có con trái là a[4] và con phải là a[5]. Khoá của a[2] bây giờ là 5 lớn hơn khoá của con phải a[5] (5 > 3) và khoá của con phải a[5] nhỏ hơn khoá của con trái a[4] (3 < 6) nên đẩy a[2] xuống bên phải (hoán đổi a[2] và a[5] cho nhau). Tiếp tục xét con phải a[5]. Nút này chỉ có một con trái là a[10] và khoá của a[5] nhỏ hơn khoá của a[10] nên không phải đẩy a[5] xuống. Quá trình đến đây kết thúc và ta có được heap trong bảng sau:

Chỉ số	1	2	3	4	5	6	7	8	9	10
Ban đầu	5 2	6 2 5 3	2	2 6	10 3 5	12	9	10	9	3 10
Heap	2	3	2	6	5	12	9	10	9	10

Hình 2-14: Mảng ban đầu đã tạo thành heap

Trong bảng trên, dòng *Ban đầu* bao gồm hai dòng. Dòng trên ghi các giá trị khoá ban đầu của mảng. Dòng dưới ghi các giá trị khoá sau khi đã có một sự hoán đổi.

Thứ tự ghi từ trái sang phải, tức là số bên trái là giá trị khoá sau khi thực hiện việc hoán đổi đầu tiên trong quá trình PushDown.

Sau khi đã có heap, ta bắt đầu quá trình sắp xếp.

Ở bước đầu tiên, ứng với i = 10. hoán đổi a[1] và a[10] cho nhau, ta được a[10] có khóa nhỏ nhất. Để đẩy a[1] xuống trong cây a[1]..a[9], ta thấy khóa của a[1] bây giờ lớn hơn khóa của con phải a[3] (10 > 2) và khóa của con phải a[3] nhỏ hơn khóa của con trái a[2] (2 < 3) do đó đẩy a[1] xuống bên phải (hoán đổi a[1] và a[3] cho nhau). Tiếp tục xét a[3], khóa của a[3] lớn hơn khóa của con phải a[7] và khóa của con phải nhỏ hơn khóa của con trái, do đó ta đẩy a[3] xuống bên phải (hóan đổi a[3] và a[7] cho nhau) và vì a[7] là nút lá nên việc đẩy xuống kết thúc. Ta có bảng sau:

Chỉ số	1	2	3	4	5	6	7	8	9	10
Ban đầu	5 2	6 2 5 3	2	2 6	10 3 5	12	9	10	9	3 10
Heap	2 10 2	3	2 10 9	6	5	12	9 10	10	9	10 2
i = 10	2	3	9	6	5	12	10	10	9	**2**

Hình 2-15: Hoán đổi a[1] với a[10] và đẩy a[1] xuống trong a[1..9]

Với i = 9, ta hoán đổi a[1] và a[9] cho nhau. Để đẩy a[1] xuống trong cây a[1]..a[8], ta thấy khóa của a[1] bây giờ lớn hơn khóa của con trái a[2] và khóa của con trái nhỏ hơn khóa của con phải a[3] nên đẩy a[1] xuống bên trái (hoán đổi a[1] và a[2] cho nhau). Tiếp tục xét a[2], khóa của a[2] lớn hơn khóa của con phải a[5] và khóa của con phải nhỏ hơn khóa của con trái a[4] nên đẩy a[2] xuống bên phải (hoán đổi a[2] và a[5] cho nhau) và vì a[5] là nút lá (trong cây a[1]..a[8]) nên việc đẩy xuống kết thúc. Ta có bảng sau

Chỉ số	1	2	3	4	5	6	7	8	9	10
Ban đầu	5 2	6 2 5 3	2	2 6	10 3 5	12	9	10	9	3 10
Heap	2 10 2	3	2 10 9	6	5	12	9 10	10	9	10 2
i = 10	2 9 3	3 9 5	9	6	5 9	12	10	10	9 2	**2**
i = 9	3	5	9	6	9	12	10	10	**2**	

Hình 2-16: Hoán đổi a[1] với a[9] và đẩy a[1] xuống trong a[1..8]

Với i = 8, ta hoán đổi a[1] và a[8] cho nhau. Để đẩy a[1] xuống trong cây a[1]..a[7], ta thấy khóa của a[1] bây giờ lớn hơn khóa của con trái a[2] và khóa của con trái nhỏ hơn khóa của con phải a[3] nên đẩy a[1] xuống bên trái (hoán đổi a[1] và a[2] cho nhau). Tiếp tục xét a[2], khóa của a[2] lớn hơn khóa của con trái a[4] và khóa của con trái nhỏ hơn khóa của con phải a[5] nên đẩy a[2] xuống bên trái (hoán đổi a[2] và a[4] cho nhau) và vì a[4] là nút lá (trong cây a[1]..a[7]) nên việc đẩy xuống kết thúc. Ta có bảng sau

Chỉ số	1	2	3	4	5	6	7	8	9	10
Ban đầu	5 2	6 2 5 3	2	2 6	10 3 5	12	9	10	9	3 10
Heap	2 10 2	3	2 10 9	6	5	12	9 10	10	9	10 2
i = 10	2 9 3	3 9 5	9	6	5 9	12	10	10	9 2	**2**
i = 9	3 10 5	5 10 6	9	6 10	9	12	10	10 3	**2**	
i = 8	5	6	9	10	9	12	10	**3**		

Hình 2-17: Hoán đổi a[1] với a[8] và đẩy a[1] xuống trong a[1..7]

Tiếp tục quá trình trên và giải thuật kết thúc sau bước 9, ứng với bước i =2.

2.5.4 Phân tích HeapSort

Thời gian thực hiện của HeapSort là O(n logn)

Như đã phân tích trong mục 2.5.3.1, thủ tục PushDown lấy O(logn) để đẩy một nút xuống trong cây có n nút.

Trong thủ tục HeapSort dòng lệnh {1}-{2}) lặp n/2 lần mà mỗi lần PushDown lấy O(logn) nên thời gian thực hiện {1}-{2} là O(n logn). Vòng lặp {3}-{4}-{5} lặp n-1 lần, mỗi lần PushDown lấy O(logn) nên thời gian thực hiện của {3}-{4}-{5} là O(n logn). Tóm lại thời gian thực hiện HeapSort là O(n logn).

2.6 BINSORT

2.6.1 Giải thuật

Nói chung các giải thuật đã trình bày ở trên đều có độ phức tạp là $O(n^2)$ hoặc O(nlogn). Tuy nhiên khi kiểu dữ liệu của trường khoá là một kiểu đặc biệt, việc sắp xếp có thể chỉ chiếm O(n) thời gian. Sau đây ta sẽ xét một số trường hợp.

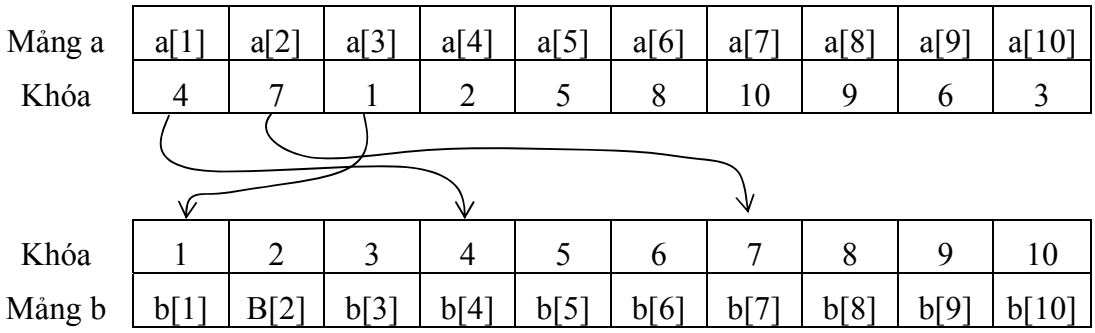
2.6.1.1 Trường hợp đơn giản:

Giả sử ta phải sắp xếp một mảng A gồm n phần tử có khoá là các số nguyên có giá trị khác nhau và là các giá trị từ 1 đến n. Ta sử dụng B là một mảng cùng kiểu với A và phân phối vào phần tử b[j] một phần tử a[i] mà a[i].key = j. Khi đó mảng B lưu trữ kết quả đã được sắp xếp của mảng A.

Ví dụ 2-7: Sắp xếp mảng A gồm 10 phần tử có khoá là các số nguyên có giá trị là các số 4, 7, 1, 2, 5, 8, 10, 9, 6 và 3

Ta sử dụng mảng B có cùng kiểu với A và thực hiện việc phân phối a[1] vào b[4] vì a[1].key = 4, a[2] vào b[7] vì a[2].key = 7, a[3] vào b[1] vì a[3].key = 1,...

Hình sau minh họa cho việc phân phối các phần tử của mảng a vào mảng b.



Hình 2-18: Phân phối các phân tử a[i] vào các bin b[j]

Để thực hiện việc phân phối này ta chỉ cần một lệnh lặp:

for i:=1 to n do b[a[i].key] := a[i]

Đây cũng là lệnh chính trong chương trình sắp xếp. Lệnh này lấy O(n) thời gian.

Các phần tử b[j] được gọi là các **bin** và phương pháp sắp xếp này được gọi là bin sort.

2.6.1.2 Trường hợp tổng quát

Là trường hợp có thể có nhiều phần tử có chung một giá trị khóa, chẳng hạn để sắp một mảng A có n phần tử mà các giá trị khóa của chúng là các số nguyên lấy giá trị trong khoảng 1..m với m <= n. Trong trường hợp này ta không thể sử dụng các phần tử của mảng B làm bin được vì nếu có hai phần tử của mảng A có cùng một khoá thì không thể lưu trữ trong cùng một bin.

Để giải quyết sự đụng độ này ta chuẩn bị một cấu trúc có m bin, mỗi bin có thể lưu trữ nhiều hơn một phần tử. Cụ thể là bin thứ j sẽ lưu các phần tử có khóa là j ($1 \le j \le m$) sau đó ta sẽ nối các bin lại với nhau để được một dãy các phần tử được sắp.

Cách tốt nhất là ta thiết kế mỗi bin là một danh sách liên kết của các phần tử mà mỗi phần tử có kiểu RecordType. Ta sẽ gọi kiểu của danh sách này là ListType.

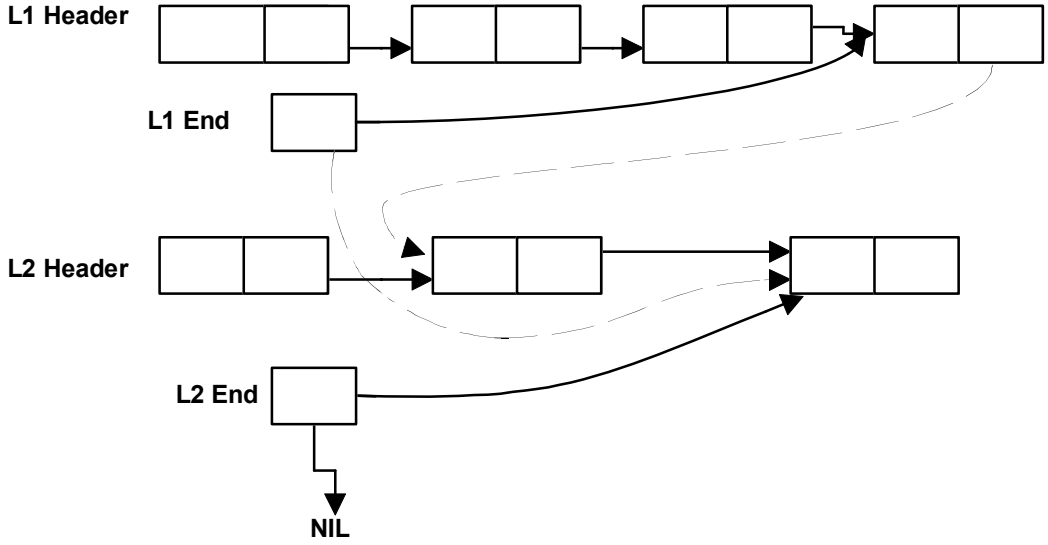
Ta có thể tạo kiểu ListType bằng cách ghép RecordType với một con trỏ để trỏ tới phần tử kế tiếp.

Lấy B là một mảng kiểu Array[KeyType] of ListType. Như vậy B là mảng các bin, mỗi bin là một danh sách. B được đánh chỉ số bởi KeyType, như thế có ít nhất một bin cho mỗi giá trị khoá.

Ta vẫn sẽ phân phối phần tử a[i] vào bin b[j] nếu j = a[i].key. Dĩ nhiên mỗi bin b[j] có thể chứa nhiều phần tử của mảng A. Các phần tử mới sẽ được đưa vào cuối danh sách b[j].

Sau khi tất cả các phần tử của mảng A đã được phân phối vào trong các bin, công việc cuối cùng là ta phải nối các bin lại với nhau, ta sẽ được một danh sách có thứ tự. Ta sẽ dùng thủ tục concatenate(L1,L2) để nối hai danh sách L1, L2. Nó thay thế danh sách L1 bởi danh sách nối L1L2. Việc nối sẽ được thực hiện bằng cách gắn con trỏ của phần tử cuối cùng của L1 vào đầu của L2. Ta biết rằng để đến được phần tử cuối cùng của danh sách liên kết L1 ta phải duyệt qua tất cả các phần tử của

nó. Để cho có hiệu quả, ta thêm một con trỏ nữa, trỏ đến phần tử cuối cùng của mỗi danh sách, điều này giúp ta đi thẳng tới phần tử cuối cùng mà không phải duyệt qua toàn bộ danh sách. Hình sau minh họa việc nối hai danh sách.



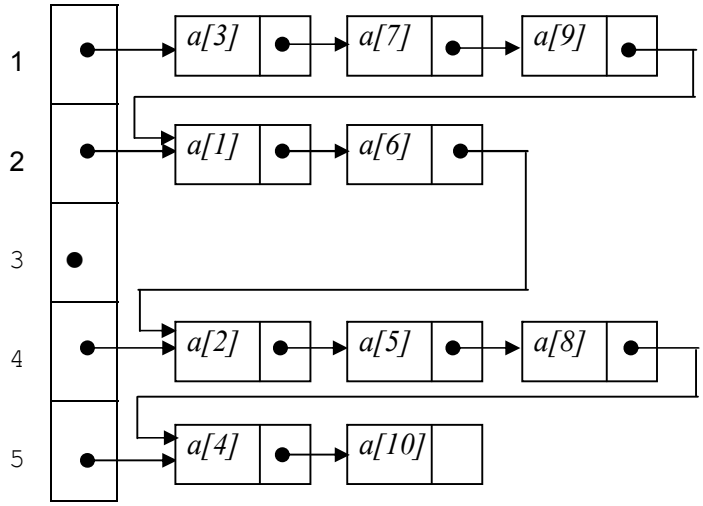
Hình 2-19: Nối các bin

Sau khi nối thì header và end của danh sách L2 không còn tác dụng nữa.

Ví dụ 2-8: Sắp xếp mảng A gồm 10 phần tử có khoá là các số nguyên có giá trị là các số 2, 4, 1, 5, 4, 2, 1, 4, 1, 5.

A	a[1]	a[2]	A[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
Khoá của A	2	4	1	5	4	2	1	4	1	5

Ta thấy các giá trị khoá nằm trong khoảng 1..5. Ta tổ chức một mảng B gồm 5 phần tử, mỗi phần tử là một con trỏ, trỏ đến một danh sách liên kết.



Hình 2-20: Binsort trong trường hợp tổng quát

Chương trình sử dụng cấu trúc danh sách liên kết làm các bin

```
VAR
a: ARRAY[1..n] OF RecordType;
b: ARRAY[keytype] OF ListType;
{Ta giả thiết keytype là kiểu miền con 1..m }
PROCEDURE BinSort;
VAR
  i:integer;
  j: KeyType;
BEGIN
{1}FOR i:=1 TO n DO
     Insert(A[i], END(B[A[i].key]), B[A[i}.key]);
{2}FOR j:= 2 TO m DO
     Concatenate(B[1], B[j]);
END;
```

2.6.2 Phân tích Bin Sort

Bin sort lấy O(n) thời gian để sắp xếp mảng gồm n phần tử.

Trước hết thủ tục INSERT cần một thời gian O(1) để xen một phần tử vào trong danh sách. Do cách tổ chức danh sách có giữ con trỏ đến phần tử cuối cùng nên việc nối hai danh sách bằng thủ tục CONCATENATE cũng chỉ mất O(1) thời gian. Ta thấy vòng lặp {1} thực hiện n lần, mỗi lần tốn O(1) = 1 nên lấy O(n) đơn vị thời gian. Vòng lặp {2} thực hiện m-1 lần, mỗi lần O(1) nên tốn O(m) đơn vị thời gian. Hai lệnh {1} và {2} nối tiếp nhau nên thời gian thực hiện của BinSort là T(n) = O(max(n,m)) = O(n) vì $m \le n$.

2.6.3 Sắp xếp tập giá trị có khoá lớn

Nếu m số các khoá không lớn hơn n số các phần tử cần sắp xếp, khi đó O(max(n,m)) thực sự là O(n). Nếu n > m thì T(n) là O(m) và đặc biệt khi $m = n^2$ thì T(n) là $O(n^2)$, như vậy Bin sort không tốt hơn các sắp xếp đơn giản khác.

Tuy nhiên trong một số trường hợp, ta vẫn có thể tổng quát hoá kĩ thuật bin sort để nó vẫn lấy O(n) thời gian.

Giả sử ta cần sắp xếp n phần tử có các giá trị khoá thuộc $0..n^2-1$. Nếu sử dụng phương pháp cũ, ta cần n^2 bin (từ bin 0 đến bin n^2-1) và do đó việc nối n^2 bin này tốn $O(n^2)$, nên bin sort lấy $O(n^2)$.

Để giải quyết vấn đề này, ta sẽ sử dụng n bin b[0], b[1],...b[n-1] và tiến hành việc sắp xếp trong hai kì.

Kì 1: Phân phối phần tử a[i] vào bin b[j] mà j = a[i].key MOD n.

Kì 2: Phân phối các phân tử trong danh sách kết quả của kỳ 1 vào các bin. Phần tử a[i] sẽ được phân phối vào bin b[j] mà j = a[i].key DIV n.

Chú ý rằng trong cả hai kỳ, ta xen các phần tử mới được phân phối vào cuối danh sách.

Ví dụ 2-9: Cần sắp xếp mảng gồm 10 phần tử có khoá là các số nguyên: 36, 9, 10, 25, 1, 8, 34, 16, 81 và 99.

Ta sử dụng 10 bin được đánh số từ 0 đến 9. Kì một ta phân phối phần tử a[i] vào bin có chỉ số a[i].key MOD 10. Nối các bin của kì một lại với nhau ta được danh sách có khóa là: 10, 1, 81, 34, 25, 36, 16, 8, 9, 99. Kì hai sử dụng kết quả của kì 1 để sắp tiếp. Phân phối phần tử a[i] vào bin có chỉ số a[i].key DIV 10. Nối các bin của kì hai lại với nhau ta được danh sách có thứ tự.

	Kì một				Kì hai		
Bin				Bin			
0	10			0	1	8	9
1	1	81		1	10	16	
2				2	25		
3				3	34	36	
4	34			4			
5	25			5			
6	36	16		6			
7				7			
8	8			8	81		
9	9	99		9	99		

Hình 2-21: Sắp xếp theo hai kỳ

Theo sự phân tích giải thuật Bin Sort thì mỗi kì lấy O(n) thời gian, hai kì này nối tiếp nhau nên thời gian tổng cộng là O(n).

2.6.3.1 Chứng minh giải thuật đúng

Để thấy tính đúng đắn của giải thuật ta xem các các giá trị khóa nguyên từ 0 đến n^2-1 như các số có hai chữ số trong hệ đếm cơ số n. Xét hai số K = s.n + t (lấy K chia cho n được s , dư t) và L = u.n + v trong đó s, t, u, v là các số 0..n-1. Giả sử K < L, ta cần chứng minh rằng sau 2 kì sắp thì K phải đứng trước L.

Vì K < L nên s ≤ u. Ta có hai trường hợp là s < u và s = u.

Trường hợp 1: Nếu s < u thì K đứng trước L trong danh sách kết quả vì trong kì hai, K được sắp vào bin b[s] và L được sắp vào bin b[u] mà b[s] đứng trước b[u]. Chẳng hạn trong ví dụ trên, ta chọn K = 16 và L = 25. Ta có K = 1 x 10 + 6 và L = 2 x 10 + 5 (s = 1, t = 6, u = 2 và v = 5; s < u). Trong kì hai, K = 16 được sắp vào bin 1 và L = 25 được sắp vào bin 2 nên K = 16 đứng trước L = 25.

Trường hợp 2: Nếu s = u thì t < v (do K < L). Sau kì một thì K đứng trước L, vì K được sắp vào trong bin b[t] và L được sắp vào trong bin b[v]. Đến kì hai, mặc dù cả K và L đều được sắp vào trong bin b[s], nhưng K được xen vào trước L nên kết quả

là K đứng trước L. Chẳng hạn trong ví dụ trên ta chọn K = 34 và L = 36. Ta có K = 3 x 10 + 4 và L = 3 x 10 + 6. Sau kì một thì K = 34 đứng trước L = 36 vì K được sắp vào bin 4 còn L được sắp vào bin 6. Trong kì hai, cả K và L đều được sắp vào bin 3, nhưng do K được xét trước nên K đứng trước L trong bin 3 và do đó K đứng trước L trong kết quả cuối cùng.

Chú ý: Từ chứng minh trên ta thấy để sắp các phần tử có khóa là các số nguyên (hệ đếm cơ số 10) từ 0 đến 99 ta dùng 10 bin có chỉ số từ 0 đến 9. Để sắp các phần tử có khóa là các số nguyên từ 0 đến 9999 ta dùng 100 bin có chỉ số từ 0 đến 99...

2.7 TỔNG KẾT CHƯƠNG 2

Các giải thuật sắp xếp đơn giản có giải thuật đơn giản nhưng kém hiệu quả về mặt thời gian. Tất cả các giải thuật sắp xếp đơn giản đều lấy $O(n^2)$ để sắp xếp n mẩu tin.

Các giải thuật QuickSort và HeapSort đều rất hiệu quả về mặt thời gian (độ phức tạp O(nlogn)), do đó chúng thường được sử dụng trong thực tế, nhất là QuickSort.

BinSort chỉ sử dụng được cho dữ liệu đặc biệt.

BÀI TẬP CHƯƠNG 2

Bài 1: Sắp xếp mảng gồm 12 phần tử có khóa là các số nguyên: 5, 15, 12, 2, 10, 12, 9, 1, 9, 3, 2, 3 bằng cách sử dụng:

a) Sắp xếp chọn.

b) Sắp xếp xen.

c) Sắp xếp nổi bọt.

d) QuickSort.

e) HeapSort (Sắp thứ tự giảm, sử dụng mô hình cây và sử dụng bảng).

Bài 2: Viết thủ tục sắp xếp trộn (xem giải thuật thô trong chương 1).

Bài 3: Viết lại hàm FindPivot để hàm trả về giá trị chốt và viết lại thủ tục QuickSort phù hợp với hàm FindPivot mới này.

Bài 4: Có một biến thể của QuickSort như sau: Chọn chốt là khóa của phần tử nhỏ nhất trong hai phần tử có khóa khác nhau đầu tiên. Mảng con bên trái gồm các phần tử có khóa nhỏ hơn hoặc bằng chốt, mảng con bên phải gồm các phần tử có khóa lớn hơn chốt. Hãy viết lại các thủ tục cần thiết cho biến thể này.

Bài 5: Một biến thể khác của QuickSort là chọn khóa của phần tử đầu tiên làm chốt. Hãy viết lại các thủ tục cần thiết cho biến thể này.

Bài 6: Hãy viết lại thủ tục PushDown trong HeapSort bằng giải thuật đệ quy.

Bài 7: Hãy viết lại thủ tục PushDown trong HeapSort để có thể sắp xếp theo thứ tự tăng.