CHƯƠNG 3: SẮP XẾP - TÌM KIẾM

Mục tiêu của chương

Nắm vững:

- Các phương pháp tìm kiếm tuần tự, tìm kiếm nhị phân.

- Các phương pháp sắp xếp đơn giản bao gồm: sắp xếp kiểu lựa chọn, sắp xếp kiểu đổi chỗ, sắp xếp kiểu thêm dần. Ngoài ra còn có một số phương pháp sắp xếp phức tạp và hiệu quả hơn như sắp xếp nhanh và sắp xếp kiểu vun đống.

- Vận dụng lý thuyết viết giải thuật giải các bài tập tìm kiếm và bài tập sắp xếp.

Nội dung của chương

Nghiên cứu các bước thực hiện các thuật toán tìm kiếm và các thuật toán sắp xếp, sau đó lấy ví dụ cụ thể, minh họa thực hiện từng bước trên các ví dụ.

3.1. Tìm kiếm

Tìm kiếm là một thao tác rất quan trọng đối với nhiều ứng dụng tin học. Tìm kiếm có thể định nghĩa là việc thu thập một số thông tin nào đó từ một khối thông tin lớn đã được lưu trữ trước đó.

Thông tin khi lưu trữ thường được chia thành các bản ghi, mỗi bản ghi có một giá trị khoá để phục vụ cho mục đích tìm kiếm.

Mục tiêu của việc tìm kiếm là tìm tất cả các bản ghi có giá trị khoá trùng với một giá trị cho trước. Khi tìm được bản ghi này, các thông tin đi kèm trong bản ghi sẽ được thu thập và xử lý.

Một ví dụ về ứng dụng thực tiễn của tìm kiếm là từ điển máy tính. Trong từ điển có rất nhiều mục từ, khoá của mỗi mục từ chính là cách viết của từ. Thông tin đi kèm là định nghĩa của từ, cách phát âm, các thông tin khác như loại từ, từ đồng nghĩa, khác nghĩa v.v.

3.1.1. Đặt bài toán

Tìm kiếm là quá trình xử lý một danh sách các phần tử (hoặc các mẫu tin) với mục đích tìm phần tử có giá trị khóa tương ứng bằng x cho trước và kết xuất vị trí k trong danh sách nếu nó hiện diện, hay ngược lại kết xuất một thông báo thích hợp, nếu giá trị x không tìm thấy.

Để thuận tiện cho việc trình bày ý tưởng của từng giải thuật, mô tả bài toán như sau:

+ Tập dữ liệu được lưu trữ là dãy N số a_0, a_2, ..., a_{N-1}.

+ Khóa cần tìm là x.

+ Nếu tìm được ở ở vị trí thứ k trong dãy thì xuất kết quả là k, ngược lại nếu không tìm thấy thì xuất kết quả là -1 (chú ý dãy bắt đầu từ chỉ số 0)

Giải thuật tìm kiếm tuyến tính và tìm kiếm nhị phân là hai giải thuật thông dụng để tìm kiếm dữ liệu.

3.1.2. Tìm kiếm tuyến tính (hay còn gọi tìm kiếm tuần tự)

3.1.2.1. Ý tưởng giải thuật

Bắt đầu từ phần tử thứ nhất a[0]. Lần lượt so sánh x với các a[i]. Nếu có a[i] nào bằng x thì i chính là kết quả và kết thúc thuật toán. Nếu trong dãy không có số a[i] nào bằng x thì kết xuất -1 và cũng kết thúc thuật toán.

3.1.2.2. Các bước tiến hành

Bước 1: i = 0; // bắt đầu từ phần tử đầu tiên của dãy.

Bước 2: So sánh a[i] với x, hai khả năng có thể xảy ra:

+ a[i] = x : Tìm thấy, xuất kết quả là i; dừng

+ a[i] ≠ x : Chuyển qua bước 3.

Bước 3: i = i + 1; // xét tiếp phần tử kế tiếp trong dãy

Nếu i = N : Hết dãy, không tìm thấy. Dừng

Ngược lại: Lặp lại bước 2.

3.1.2.3. Cài đặt

```
1. int LinearSearch (int a[], int N, int x )
2. {
3.          int i = 0;
4.          while ((i<N) && (a[i]!=x))
5.                  i++;
6.          if( i==N)  return -1;  // tìm hết nhưng không có x
7.          return i;          // tìm thấy a[i] là phần tử có khóa x
8. }
```

Đánh giá giải thuật

Thuật toán tìm kiếm tuần tự có thời gian thực hiện là O(n). Trong trường hợp xấu nhất, thuật toán mất n lần thực hiện so sánh và mất khoảng n/2 lần so sánh trong trường hợp trung bình.

3.1.2.4. Ví dụ minh họa

Cho dãy số A[] ={1, 25, 6, 5, 2, 37, 40}; tìm phần tử x=37.

A[n]	1	25	6	5	2	37	40
i=	[0]	[1]		….			[n-1]

Kết quả là tìm thấy x trong dãy.

3.1.3. Tìm kiếm nhị phân

3.1.3.1. Ý tưởng giải thuật

Với phương pháp tìm kiếm nhị phân thì dãy N số a_0, a_1, a_2, .., a_{N-1} phải có thứ tự, giả sử có thứ tự không giảm.

Giả sử dãy tìm kiếm bao gồm các phần tử a_{left}, …, a_{right} và gọi midle=(left + right)/2.

Nhận xét nếu x > a[i] thì x chỉ có thể xuất hiện bên phải a[i], nghĩa là trong đoạn [$a_{midle+1}$, a_{right}] của dãy, ngược lại nếu x < a[i] thì x chỉ có thể xuất hiện bên trái a[i], trong đoạn [a_{left}, $a_{midle-1}$] của dãy, nhờ vậy giải thuật sẽ thu gọn phạm vi tìm kiếm một cách đáng kể.

3.1.3.2. Các bước tiến hành

Các bước tiến hành: (không đệ quy)

Bước 1: left=0; right=N-1; midle= (left+right)/2

Bước 2: if a[midle]=x

{tìm thấy, dừng}

else

if a[midle] > x

right=midle-1; //tìm tiếp x trong dãy con a[left],.., a[mid-1]

else

left=midle+1; //tìm tiếp x trong dãy con a[mid+1],.., a[right]

Bước 3: if left<=right //còn phần tử chưa xét

lặp lại bước 2

else kết thúc

3.1.3.3. Cài đặt

a. Cài đặt theo kiểu đệ quy

```
1. int TKnhiphan_dequy(int a[], int N, int x, int left, int right)
2. {
3.      if (left>right)     return -1;
4.      int mid=(left+right)/2;
5.      if (x==a[mid])   return mid;
6.      if (x<a[mid])     return TKnhiphan_dequy(a, N, x, left, mid-1);
7.      return TKnhiphan_dequy (a, N, x, mid+1, right);
8. }
```

b. Cài đặt theo kiểu không đệ quy

```
1. int TKnhiphan(int a[], int N, int x)
```

```
2.  {
3.      int left = 0, right = N-1, mid;
4.      do
5.      {
6.          mid=(left+right)/2;
7.          if (x==a[mid])      return mid;
8.          else
9.              if (x<a[mid])
10.                 right = mid -1;
11.             else
12.                 left = mid+1;
13.     }
14.     while (left <= right);
15.     return -1;
16. }
```

Đánh giá giải thuật: độ phức tạp thuật toán là O(log(n)).

3.1.3.4. Ví dụ minh họa

Cho dãy số a gồm 8 phần tử

1 2 4 5 6 8 12 15

Và khóa cần tìm là x=8

Đầu tiên left = 0, right =7; midle = 3;

1	2	4	5	6	8	12	15

do a[midle]<x nên cập nhật lại các cận như sau:

left=midle+1 = 3+1=4, right=7; và lúc này midle=(left+right)/2=(4+7)/2 = 5;

1	2	4	5	6	8	12	15

Rõ ràng a[midle]=a[5] = 8; nên kết quả là tìm thấy tại ví trị thứ 5 của dãy.

3.2. Sắp xếp

Sắp xếp là quá trình bố trí lại các phần tử của một tập hợp theo thứ tự nào đó. Mục đích chính của sắp xếp là làm cho thao tác tìm kiếm phần tử trên tập đó được dễ dàng hơn.

Ví dụ về tập các đối tượng được sắp xếp phổ biến trong thực tế là: danh bạ điện thoại được sắp theo tên, các từ trong từ điển được sắp theo vần, sách trong thư viện được sắp theo mã số, theo tên, .v.v

Các giải thuật sắp xếp còn là một ví dụ điển hình cho sự đa dạng của thuật toán.

Cùng một mục đích, nhưng có rất nhiều cách thực hiện, mỗi cách tối ưu trên một khía cạnh nào đó, và có một số cách sắp xếp có nhiều ưu điểm hơn những cách khác. Do đó, sắp xếp cũng được sử dụng như một ví dụ điển hình trong việc phân tích thuật toán.

Thông thường, các giải thuật sắp xếp được chia làm 2 loại. Loại thứ nhất là các giải thuật được cài đặt đơn giản, không hiệu quả (phải sử dụng nhiều thao tác). Loại thứ hai là các giải thuật được cài đặt phức tạp, nhưng hiệu quả hơn về mặt tốc độ (dùng ít thao tác hơn).

3.2.1. Sắp xếp kiểu lựa chọn (Selection Sort)

3.2.1.1. Ý tưởng giải thuật

Ban đầu dãy gồm n phần tử không có thứ tự. Ta chọn phần tử nhỏ nhất trong n phần tử của dãy cần sắp xếp và đổi giá trị của nó với a[1], khi đó a[1] có giá trị nhỏ nhất trong dãy;

Lần thứ 2, ta chọn phần tử nhỏ nhất trong n-1 phần tử còn lại từ a[2], a[3], ..., a[n] và đổi giá trị của nó với a[2];

Ở lượt thứ i ta chọn trong dãy a[i ... n] ra phần tử nhỏ nhất và đổi chỗ giá trị của nó với a[i];

Tới lượt thứ n-1 chọn trong 2 phần tử a[n – 1] và a[n] ra phần tử có giá trị nhỏ hơn và đổi giá trị của nó với a[n – 1]. Khi đó dãy thu được có thứ tự không giảm.

3.2.1.2. Các bước tiến hành

Bước 1: i = 0;

Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[n-1].

Bước 3: Hoán vị a[min] và a[i].

Bước 4: Nếu i < n-1 thì i = i+1; lặp lại bước 2.

Ngược lại: dừng. // n-1 phần tử đã được sắp đúng vị trí.

3.2.1.3. Cài đặt

Thuật toán Selection-Sort:

Input:

- Dãy các đối tượng (các số) : Arr[0], Arr[1],..,Arr[n-1].
- Số lượng các đối tượng cần sắp xếp: n.

Output:

- Dãy các đối tượng đã được sắp xếp (các số) : Arr[0], Arr[1],..,Arr[n-1].

Formats: Selection-Sort(Arr, n);

Actions:

```
for (i =0; i<n-1; i++) { //duyệt các phần tử i=0,1,.., n-1
        min_idx = i;  //gọi min_idx là vị trí của phần tử nhỏ nhất trong dãy con
        for ( j = i +1; j<n; j++ ) { //duyệt từ phần tử tiếp theo j=i+1,..,n.
```

```
            if (Arr[i] > Arr[j] ) // nếu Arr[i] không phải nhỏ nhất trong dãy con
                min_idx = j; //ghi nhận đây mới là vị trí phần tử nhỏ nhất.
        }
        //đặt phần tử nhỏ nhất vào vị trí đầu tiên của dãy con chưa được sắp
        Temp = Arr[i] ; Arr[i] = Arr[min_idx]; Arr[min_idx] = temp;
    End.
```

Đánh giá giải thuật:

Tổng số phép toán so sánh là:

$S_{so\ sánh} = (n - 1) + (n - 2) + ... + 2 + 1 = n * (n - 1) / 2.$

Trong trường hợp tốt nhất là khi dãy đã có thứ tự tăng dần:

Số phép gán $S_{gán} = 0$

Trong trường hợp tồi nhất là khi dãy có thứ tự giảm dần:

$S_{gán} = 3 * (n - 1) + n * (n - 1) / 2 = (n - 1) * (n + 6) / 2.$

Như vậy độ phức tạp của thuật toán sắp xếp lựa chọn này là $O(n^2)$.

3.2.1.4. Ví dụ

Cho dãy a:

12 2 8 5 1 6 4 15

Dùng thuật toán sắp xếp lựa chọn để sắp xếp dãy trên theo thứ tự tăng dần.

Dòng	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	Doicho(a[min],a[i])
1	12	2	8	5	1	6	4	15	a[4]=1 là phần min
2	1	2	8	5	12	6	4	15	Đổi chỗ(1,12)
3		2	8	5	12	6	4	15	
4		2	8	5	12	6	4	15	Số 2 đã nằm đúng chỗ
5			4	5	12	6	8	15	Đổi chỗ (4,8)
6				5	12	6	8	15	Số 5 đã nằm đúng chỗ
7					12	6	8	15	
8					6	12	8	15	Đổi chỗ (6,12)
9						8	12	15	Đổi chỗ (8,12)
10							12	15	Số 12 đã nằm đúng chỗ
11								15	

3.2.2. Sắp xếp kiểu thêm dần (Insertion Sort)

3.2.2.1. Ý tưởng giải thuật

Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con (dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

+ Khởi tạo mảng với dãy con đã sắp xếp có k=1 phần tử (phần tử đầu tiên, phần tử có chỉ số 0).

+ Duyệt từng phần tử từ phần tử thứ 2, tại mỗi lần duyệt phần tử ở chỉ số i thì đặt phần tử đó vào một vị trí nào đó trong đoạn từ [0…i] sao cho dãy số từ [0…i] vẫn đảm bảo tính chất dãy số tăng dần. Sau mỗi lần duyệt, số phần tử đã được sắp xếp k trong mảng tăng thêm 1 phần tử.

Lặp cho tới khi duyệt hết tất cả các phần tử của mảng.

3.2.2.2. Các bước tiến hành

Bước 1:

i=0; // bắt đầu từ đầu dãy

Bước 2:

j=i+1;

Bước3:

Trong khi j<n thực hiện

Nếu a[i] >a[j] thì đổi chỗ a[i] với a[j]

j=j+1;

Bước 4:

i = i+1;

Nếu i < n-1 lặp lại bước 2.

Ngược lại: dừng

3.2.2.3. Cài đặt

Thuật toán Insertion-Sort:

Input:

- Dãy các đối tượng (các số): Arr[0], Arr[1],.., Arr[n-1].
- Số lượng các đối tượng cần sắp xếp: n.

Output:

- Dãy các đối tượng đã được sắp xếp (các số) : Arr[0], Arr[1],..,Arr[n-1].

Formats: Insertion_Sort(Arr, n);

Actions:

```
for (i = 1; i < n; i++) { //lặp i=1, 2,..,n.
        key = Arr[i];  //key là phần tử cần chèn váo dãy Arr[0],.., Arr[i-1]
        j = i-1;
        while (j >= 0 && Arr[j] > key) { //Duyệt lùi từ vị trí j=i-1
                Arr[j+1] = Arr[j]; //dịch chuyển Arr[j] lên vị trí Arr[j+1]
                j = j-1;
                }
        Arr[j+1] = key; // vị trí thích hợp của key trong dãy là Arr[j+1]
```

}

End.

Đánh giá giải thuật:

Trường hợp tốt nhất, khi dãy phần tử cần sắp xếp có thứ tự tăng:

Số phép gán: $S_{gán} = 2(n - 1)$

Số phép so sánh: $S_{so\ sánh} = n - 1$

Trường hợp tồi nhất, khi dãy phần tử cần sắp xếp có thứ tự giảm dần:

Số phép gán: $S_{gán} = n(n + 1)/2 - 1$

Số phép so sánh: $S_{so\ sánh} = n(n - 1)/2$

Như vậy độ phức tạp của thuật toán sắp xếp chèn là $O(n^2)$ trong cả trường hợp tồi nhất và trường hợp trung bình.

3.2.2.4. Ví dụ

Cho dãy số gồm 8 phần tử:

12 2 8 5 1 6 4 15

Dùng thuật toán sắp xếp thêm dần minh họa dãy số trên.

Kết quả từng bước như sau:

Dòng	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	Đổi chỗ (a[i],a[j])
1	12	2	8	5	1	6	4	15	Đổi chỗ (a[0],a[1])
2	2	12	8	5	1	6	4	15	Đổi chỗ (a[0],a[4])
3	1	12	8	5	2	6	4	15	Đổi chỗ (a[1],a[2])
4	1	8	12	5	2	6	4	15	Đổi chỗ (a[1],a[3])
5	1	5	12	8	2	6	4	15	Đổi chỗ (a[1],a[4])
6	1	2	12	8	5	6	4	15	Đổi chỗ (a[2],a[3])
7	1	2	8	12	5	6	4	15	Đổi chỗ (a[2],a[4])
8	1	2	5	12	8	6	4	15	Đổi chỗ (a[2],a[6])
9	1	2	4	12	8	6	5	15	Đổi chỗ (a[3],a[4])
10	1	2	4	8	12	6	5	15	Đổi chỗ (a[3],a[5])
11	1	2	4	6	12	8	5	15	Đổi chỗ (a[3],a[6])
12	1	2	4	5	12	8	6	15	Đổi chỗ (a[4],a[5])
13	1	2	4	5	8	12	6	15	Đổi chỗ (a[4],a[6])
14	1	2	4	5	6	12	8	15	Đổi chỗ (a[5],a[6])
15	1	2	4	5	6	8	12	15	

3.2.3. Sắp xếp kiểu nổi bọt (Bubble Sort)

3.2.3.1. Ý tưởng giải thuật

• Duyệt các phần tử đi từ cuối mảng về đầu mảng.

• Trong quá trình đi nếu phần tử ở dưới (đứng phía sau) nhỏ hơn phần tử đứng ngay trên (trước) nó thì hai phần tử này sẽ được đổi chỗ cho nhau.

• Kết quả là phần tử nhỏ nhất (nhẹ nhất) sẽ được đưa về đầu dãy rất nhanh.

• Sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i. Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét và ta có dãy sắp xếp theo một thứ tự.

3.2.3.2. Các bước tiến hành

Bước 1: i=0; // Lần xử lý đầu tiên

Bước 2: j = n-1; // Duyệt từ cuối dãy ngược về vị trí i

Trong khi (j<i) thực hiện:

Nếu a[j]< a[j-1] thì a[j] ⟷ a[i]; // xét cặp phần tử kế cận j = j-1;

Bước 3: i = i+1; //lần xử lý kế tiếp

Nếu i = n-1; Hết dãy: Dừng

Ngược lại: Lặp lại Bước 2.

3.2.3.3. Cài đặt

Thuật toán Bubble-Sort:

Input:

- Dãy các đối tượng (các số) : Arr[0], Arr[1],..,Arr[n-1].
- Số lượng các đối tượng cần sắp xếp: n.

Output:

- Dãy các đối tượng đã được sắp xếp (các số) : Arr[0], Arr[1],..,Arr[n-1].

Formats: Bubble_Sort(Arr, n);

Actions:

```
for (i = 0; i < n; i++) {        //lặp i=0, 1, 2,..,n.
    for (j = i+1; j< n-i-1; j ++ ) {        //lặp j =0, 1,.., n-i-1
        if (Arr[j] > Arr[j+1] ) {       //nếu Arr[j]>Arr[j+1] thì đổi chỗ
            temp = Arr[j];
            Arr[j] = Arr[j+1];
            Arr[j+1] = temp;
        }
    }
```

}

End.

Đánh giá giải thuật:

Trong trường hợp tốt nhất tức là dãy ban đầu đã có thứ tự tăng dần thì tổng số hoán vị $S_{hv} = 0$.

Trong trường hợp tồi nhất khi danh sách có thứ tự giảm dần:

Số lần so sánh là $S_{so\ sánh} = (n - 1) + (n - 2) + ... + 2 + 1 = n(n - 2) / 2$.

Số lần hoán vị là $S_{hoán\ vị} = (n - 1) + (n - 2) + ... + 2 + 1 = n(n - 2) / 2$.

Độ phức tạp của thuật toán này trong trường hợp tồi nhất là $O(n^2)$.

3.2.3.4. Ví dụ

Cho dãy số gồm 8 phần tử:

12 2 8 5 1 6 4 15

Dùng thuật toán sắp xếp nổi bọt minh họa dãy số trên theo thứ tự tăng dần.

dòng	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
1	12	2	8	5	1	6	4	15
2	12	2	8	5	1	4	6	15
3	12	2	8	1	5	4	6	15
4	12	2	1	8	5	4	6	15
5	12	1	2	8	5	4	6	15
6	1*	12	2	8	5	4	6	15
7		12	2	8	5	4	6	15
8		12	2	8	4	5	6	15
9		12	2	4	8	5	6	15
10		2*	12	4	8	5	6	15
11			12	4	5	8	6	15
12			4*	12	8	5	6	15
13				12	5	8	6	15
14				5*	12	8	6	15
15					12	6	8	15
16					6*	12	8	15
17						8*	12	15
18							12*	15

3.2.4. Sắp xếp nhanh (Quick Sort)

3.2.1.1. Ý tưởng giải thuật

Thuật toán sắp xếp Quick-Sort được thực hiện theo mô hình chia để trị. Thuật toán được thực hiện xung quang một phần tử gọi là chốt (key). Mỗi cách lựa chọn vị trí phần tử chốt trong dãy sẽ cho ta một phiên bản khác nhau của thuật toán. Các phiên bản (version) của thuật toán Quick-Sort thông dụng là:

- Luôn lựa chọn phần tử đầu tiên trong dãy làm chốt.
- Luôn lựa chọn phần tử cuối cùng trong dãy làm chốt.
- Luôn lựa chọn phần tử ở giữa dãy làm chốt.
- Lựa chọn phần tử ngẫu nhiên trong dãy làm chốt.

Mấu chốt của thuật toán Quick-Sort là làm thế nào ta xây dựng được một thủ tục phân đoạn (Partition). Thủ tục Partition có hai nhiệm vụ chính:

- Định vị chính xác vị trí của chốt trong dãy nếu được sắp xếp;
- Chia dãy ban đầu thành hai dãy con: dãy con ở phía trước phần tử chốt bao gồm các phần tử nhỏ hơn hoặc bằng chốt, dãy ở phía sau chốt có giá trị lớn hơn chốt.

3.2.1.2. Các bước tiến hành

Để chia dãy thành 2 phần thoả mãn yêu cầu như trên, ta lấy một phần tử của dãy làm khoá (chẳng hạn phần tử đầu tiên).

Tiến hành duyệt từ bên trái dãy và dừng lại khi gặp 1 phần tử lớn hơn hoặc bằng khoá. Đồng thời tiến hành duyệt từ bên phải dãy cho tới khi gặp 1 phần tử nhỏ hơn hoặc bằng khoá. Rõ ràng 2 phần tử này nằm ở những vị trí không phù hợp và chúng cần phải được đổi chỗ cho nhau.

Tiếp tục quá trình cho tới khi 2 biến duyệt gặp nhau, ta sẽ chia được dãy thành 2 nửa: Nửa bên phải khoá bao gồm những phần tử lớn hơn hoặc bằng khoá và nửa bên trái là những phần tử nhỏ hơn hoặc bằng khoá.

Có nhiều thuật toán khác nhau để xây dựng thủ tục Partition, dưới đây là một phương pháp xây dựng thủ tục Partion với khóa chốt là phần tử cuối cùng của dãy.

3.2.1.3. Cài đặt

a. Thuật toán Partition:

Input:

- Dãy Arr[] bắt đầu tại vị trí l và kết thúc tại h.

- Cận dưới của dãy con: l
- Cận trên của dãy con: h

Output:

- Vị trí chính xác của Arr[h] nếu dãy Arr[] được sắp xếp.

Formats: Partition(Arr, l, h);

Actions:

```
x = Arr[h];     // Chọn x là chốt của dãy Arr[l],Arr[l+1],.., Arr[h].
i = (l - 1);       // i là chỉ số của các phần tử nhỏ hơn chốt x.
for ( j = l; j <= h- 1;  j++) { //duyệt các chỉ số j=l, l+1, …, h-1.
        if (Arr[j] <= x){ //nếu Arr[j] nhỏ hơn hoặc bằng chốt x.
                i++; //tăng vị trí các phần tử nhỏ hơn chốt.
                swap(&Arr[i], &Arr[j]);// đổi chỗ Arr[i] cho Arr[j].
        }
}
swap(&Arr[i + 1], &Arr[h]); // đổi chỗ Arr[i+1] cho Arr[h].
return (i + 1); //Vị trí i+1 chính là vị trí chốt nếu được sắp xếp.
End.
```

b. Thuật toán Quick-Sort

Thuật toán Partition:

Input :

- Dãy Arr[] gồm n phần tử.
- Cận dưới của dãy: l.
- Cận trên của dãy : h

Output:

- Dãy Arr[] được sắp xếp.

Formats: Quick-Sort(Arr, l, h);

Actions:

```
if( l<h) { // Nếu cận dưới còn nhỏ hơn cận trên
        p = Partition(Arr, l, h);        //Thực hiện Partition() chốt h.
        Quick-Sort(Arr, l, p-1);       //Thực hiện Quick-Sort nửa bên trái.
        Quick-Sort(Arr, p+1, h);   //Thực hiện Quick-Sort nửa bên phải
}
End.
```

Đánh giá giải thuật:

Trường hợp tốt nhất: mỗi lần phân hoạch ta đều chọn được phần tử (phần tử lớn hơn hay bằng nửa số phần tử và nhỏ hơn hay bằng nửa số phần tử còn lại) làm mốc. Khi đó dãy được phân hoạch thành hai phần bằng nhau, trong mỗi lần phân hoạch ta cần duyệt qua n phần tử. Vậy độ phức tạp trong trường hợp tốt nhất thuộc O(nlog(n)).

Trường hợp xấu nhất: mỗi lần phần hoạch ta chọn phải phần tử có giá trị cực đại hoặc cực tiểu làm mốc. Khi đó dãy bị phân hoạch thành hai phần không đều: một phần chỉ có một phần tử, phần còn lại có n-1 phần tử. Do đó, ta cần tới n lần phân hoạch mới sắp xếp xong. Vậy độ phức tạp trong trường hợp xấu nhất thuộc $O(n^2)$.

Ta có độ phức tạp của Quick Sort như sau:

- Trường hợp tốt nhất: O(nlogn)
- Trường hợp xấu nhất: $O(n^2)$
- Trường hợp trung bình: O(nlogn)

3.2.1.4. Ví dụ

Cho dãy số: Arr[] = {10, 27, 15, 29, 21, 11, 14, 18, 12, 17};

Dùng thuật toán sắp xếp Quick-Sort minh họa dãy số trên theo thứ tự tăng dần.

a. Sắp xếp phân đoạn (Partition)

p = Partition(Arr, 0, 9); l=0, h = 9.

Khóa x = Arr[h] = 17. Vị trí bắt đầu i= l - 1 = -1

Các bước thực hiện

Bước j=?	(Arr[j]<=x) ?	Đổi chỗ Arr[i] cho Arr[j]
j = 0 : i=0	(10<=17): Yes	Arr[] = {10, 27, 15, 29, 21, 11, 14, 18, 12, 17}
j = 1: i=0	(27<=17):No	Arr[] = {10, 27, 15, 29, 21, 11, 14, 18, 12, 17}
j = 2: i=1	(15<=17):Yes	Arr[] = {10, 15, 27, 29, 21, 11, 14, 18, 12, 17}
j = 3: i=1	(29<=17):No	Arr[] = {10, 15, 27, 29, 21, 11, 14, 18, 12, 17}
j = 4: i=1	(21<=17):No	Arr[] = {10, 15, 27, 29, 21, 11, 14, 18, 12, 17}
j = 5: i=2	(11<=17):Yes	Arr[] = {10, 15, 11, 29, 21, 27, 14, 18, 12, 17}
j = 6: i=3	(14<=17):Yes	Arr[] = {10, 15, 11, 14, 21, 27, 29, 18, 12, 17}
j = 7: i=3	(18<=17):No	Arr[] = {10, 15, 11, 14, 21, 27, 29, 18, 12, 17}
j = 8:i=4	(12<=17)Yes	Arr[] = {10, 15, 11, 14, 12, 27, 29, 18, 21, 17}
i+1 =5	Arr[] = {10, 15, 11, 14, 12} (17) { 29, 18, 21, 27}	

Kết luận p = i+1 = 5 là vị trí của khóa x =17 trong dãy Arr[] nếu được sắp xếp.

b. Sắp xếp nhanh (Quick-Sort)

Arr[] = {10, 27, 15, 29, 21, 11, 14, 18, 12, 17};

Quick-Sort(Arr, 0, 9);

p = Partition(Arr,l,h)	**Giá trị Arr[]=?**
p=5:l=0, h=9	{10,15,11,14,12}, **(17)**,{29,18, 21, 27}
P=2:l=0, h=4	{10,11},**(12)**, {14,15}, (17),{29,18, 21, 27}
P=1:l=0, h=1	{10,**11**},(12), {14,15}, (17),{29,18, 21, 27}
P=4: l=3, h=4	{10,11},(12), {14,**15**}, (17),{29,18, 21, 27}
P=8: l=6, h=9	{10,11},(12), {14,15}, (17),{18,21},(**27**),{29}
P=7: l=6, h=7	{10,11},(12), {14,15}, (17),{18,**21**},(27),{29}

Kết luận dãy được sắp Arr[] ={ 10, 11, 12, 14, 15, 17, 18, 21, 27, 29}.

3.2.6. Sắp xếp kiểu vun đống (Heap Sort)

3.2.6.1. Ý tưởng giải thuật

Ý tưởng cơ bản của giải thuật này là thực hiện sắp xếp thông qua việc tạo các heap, trong đó heap là 1 cây nhị phân hoàn chỉnh có tính chất là khóa ở nút cha bao giờ cũng lớn hơn khóa ở các nút con.

Việc thực hiện giải thuật này được chia làm 2 giai đoạn.

Đầu tiên là việc tạo heap từ dãy ban đầu. Theo định nghĩa của heap thì nút cha bao giờ cũng lớn hơn các nút con. Do vậy, nút gốc của heap bao giờ cũng là phần tử lớn nhất.

Giai đoạn thứ 2 là việc sắp dãy dựa trên heap tạo được. Do nút gốc là nút lớn nhất nên nó sẽ được chuyển về vị trí cuối cùng của dãy và phần tử cuối cùng sẽ được thay vào gốc của heap. Khi đó ta có 1 cây mới, không phải heap, với số nút được bớt đi 1. Lại chuyển cây này về heap và lặp lại quá trình cho tới khi heap chỉ còn 1 nút. Đó chính là phần tử bé nhất của dãy và được đặt lên đầu.

3.2.6.2. Các bước tiến hành

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. chi 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 chi 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 cỏ gốc ở vị trí 1.

3.2.6.3. Cài đặt

Giải thuật sắp xếp dãy n khoá $X_1, X_2,..., X_n$ được biểu diễn sau đây:

```
voìd Hoan_vi(X[ ], k, r)
{
  if (X[kj khác lá và có giá trị nhỏ hơn 2 con)
          {+ Chọn con có giá trị lớn hơn, già sừ là X[j];
            + Đổi chỗ X[k], và X[j] ;
            + 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.

```
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.

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

Đánh giá giải thuật

Một cây nhị phân hoàn chinh có n nút thì chiều cao của cây đó là $[\log_2(n+l)]$. 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 chi tỷ lệ với chiều cao của cây.

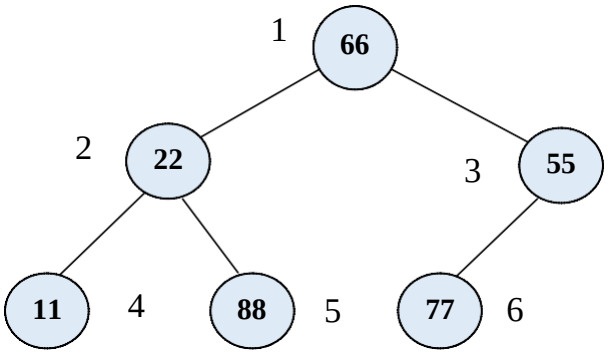
Do đó có thể suy ra, trong trường hợp xấu nhất, cấp độ lớn của thời gian thực hiện Heap Sort là $O(n\log_2 n)$. Việc đánh giá thời gian thực hiện trung bình giải thuật Heap Sort cũng là $O(n\log_2 n)$.

3.2.6.4. Ví dụ

Cho dãy số biểu diễn trong máy dưới dạng véc tơ như sau

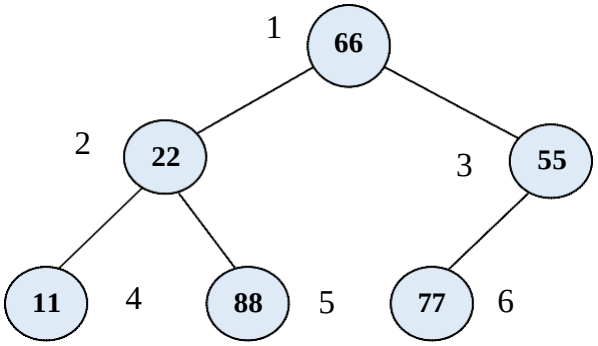
V	66	22	55	11	88	77

+ Có thể coi V là 1 véc tơ biểu diễn trong máy của cây nhị phân hoàn chỉnh có dạng như sau:



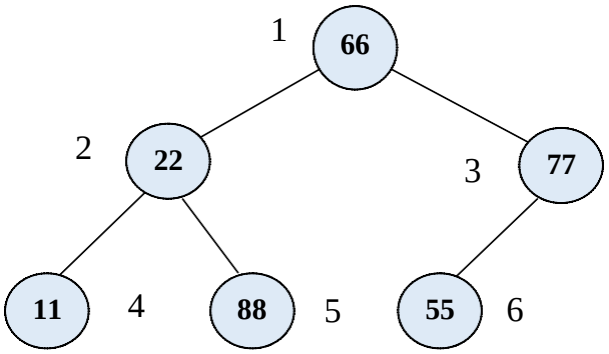
Biểu diễn cây nhị phân trên bằng thuật toán Heap Sort

+ Tạo đống



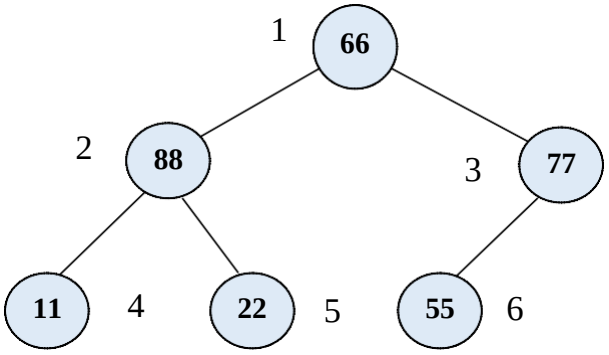
66	22	55	11	88	77
1	2	3	4	5	6

a) Cấu trúc dữ liệu và cấu trúc lưu trữ lúc ban đầu



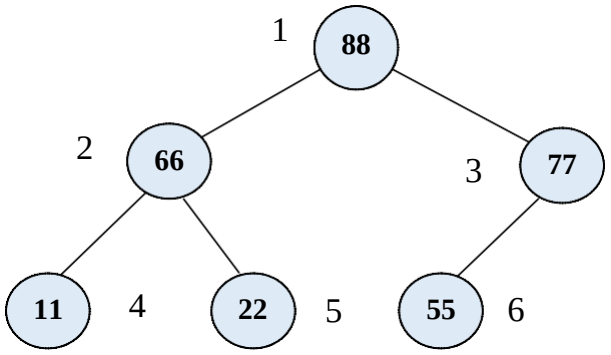
66	22	77	11	88	55
1	2	3	4	5	6

b) Sau khi thực hiện đổi chỗ (3,6)



66	88	77	11	22	55
1	2	3	4	5	6

c) Sau khi thực hiện đổi chỗ (2,6)

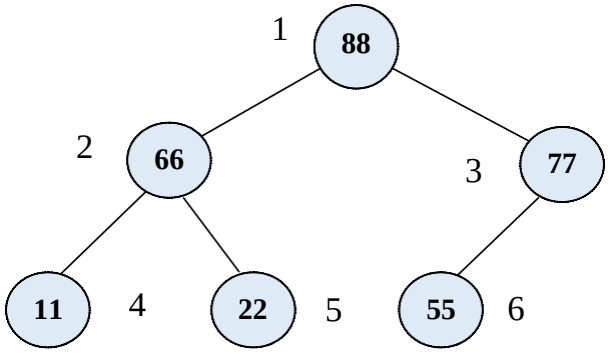


88	66	77	11	22	55
1	2	3	4	5	6

d) Sau khi thực hiện đổi chỗ (1,6)

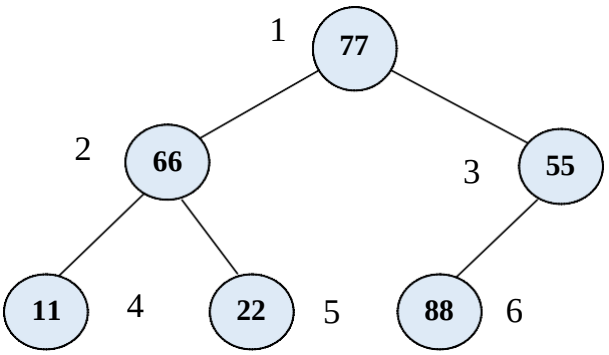
Cây đã là đống, khóa lớn nhất đang ở đỉnh đống

+ Vun đống



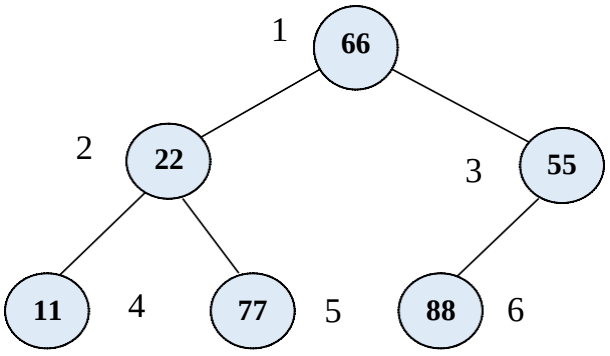
88	66	77	11	22	55
1	2	3	4	5	6

a) Đống được tạo sau khi thực hiện ở giai đoạn 1



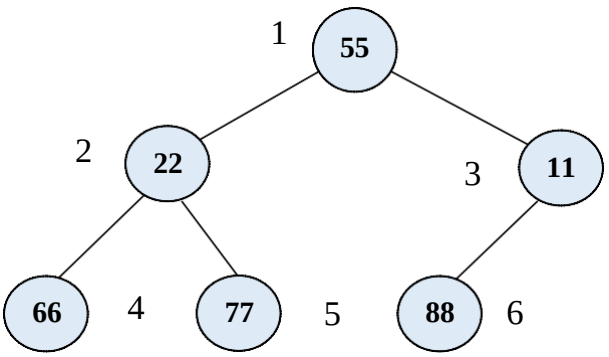
77	66	55	11	22	88
1	2	3	4	5	6

b) Sau khi thực hiện đổi chỗ (1,5)



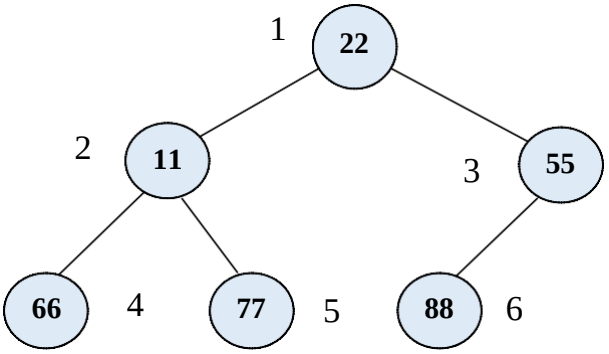
66	22	55	11	77	88
1	2	3	4	5	6

c) Sau khi thực hiện đổi chỗ (1,4)



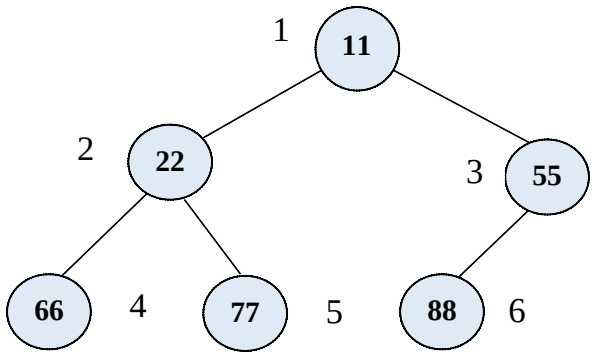
55	22	11	66	77	88
1	2	3	4	5	6

d) Sau khi thực hiện đổi chỗ (1,3)



22	11	55	66	77	88
1	2	3	4	5	6

e) Sau khi thực hiện đổi chỗ (1,2)



11	22	55	66	77	88
1	2	3	4	5	6

f) Sau khi thực hiện đổi chỗ (1,1)

Cây đã được sắp xếp

Bài tập vận dụng:

1. Cho dãy số:

2 7 9 8 6 5 3 1

Nêu các bước thực hiện sắp xếp chọn, sắp xếp chèn, và sắp xếp nổi bọt cho dãy trên.

2. Viết thuật giải sắp xếp Quick Sort, Heap Sort.

3. Cho dãy số gồm 7 phần tử:

15 37 12 58 07 24 67

Nêu các bước thực hiện sắp xếp giải thuật Quick sort, với cách chọn khoá là phần tử đầu tiên.

4. Viết chương trình tạo một danh sách sinh viên, mỗi sinh viên gồm các thông tin:

mã sinh viên, họ và tên, năm sinh, giới tính và điểm tổng kết. Hiển thị danh sách ra màn hình sao cho điểm trung bình của sinh viên theo thứ tự giảm dần.

a) Sử dụng phương pháp sắp xếp phân đoạn.

b) Sử dụng phương pháp sắp xếp vun đống.

5. Cho dãy số:

42 23 74 11 65 58 94 36 99 87

Nêu các bước thực hiện sắp xếp Quick Sort, Heap Sort.

6. Cho dãy số:

15 37 12 58 07 24 67

Nêu các bước trong quá trình tạo cây nhị phân tìm kiếm. Minh họa các bước tìm phần tử 07 trong cây.

7. Cho dãy số: 42 23 74 11 65 58

a) Biểu diễn dãy trên thành cây nhị phân hoàn chỉnh.

b) Vẽ hình ảnh của đống qua các bước sắp xếp kiểu vun đống.

8. Cho dãy số:

42 23 74 11 65 58 94 36 99 87

Nêu các bước thực hiện sắp xếp Quick Sort, Heap Sort.

9. Cho dãy số: 4 7 -2 12 -8 4 6 -6

Hãy trình bày ba bước đầu tiên của thuật toán sắp xếp chọn, sắp xếp chèn, sắp xếp nổi bọt để chuyển dãy trên thành dãy không giảm.

10. Cho dãy số:

4 7 -2 7 -8 4 6 -6

Hãy trình bày chi tiết các bước của thuật toán sắp xếp phân đoạn để chuyển dãy trên thành dãy không giảm.

11. Cho dãy số:

a) 50 08 34 06 98 17 83 25 66 42 21 59 62 71 85 76

b) 81 63 69 74 14 77 56 57 9 25

Áp dụng thuật toán: Quick Sort, Heap Sort

- Sắp xếp dãy số trên theo thứ tự tăng dần
- Sắp xếp dãy số trên theo thứ tự giảm dần