SẮP XẾP

Đỗ Thanh Nghị

dtnghi@cit.ctu.edu.vn

NỘI DUNG

- GIẢI THUẬT SẮP XẾP ĐƠN GIẢN
  - bubble sort, selection sort, insertion sort
- GIẢI THUẬT SẮP XẾP NHANH
  - quick sort, heap sort, bin sort

GIỚI THIỆU

- TẠI SAO CẦN SẮP XẾP
  - Sắp xếp một danh sách các đối tượng theo một thứ tự nào đó là một bài toán có ý nghĩa trong thực tiễn
  - Sắp xếp là một yêu cầu không thể thiếu trong khi thiết kế các phần mềm ứng dụng
  - Nghiên cứu phương pháp sắp xếp là rất cần thiết

GIỚI THIỆU

- KHÁI NIỆM
  - Sắp xếp trong là sự sắp xếp dữ liệu được tổ chức trong bộ nhớ trong của máy tính
  - 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 là một mảng của các mẩu tin vừa nói ở trên

GIỚI THIỆU

- KHÁI NIỆM
  - 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
  - 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

GIỚI THIỆU

- ĐỊNH NGHĨA VÀ KHAI BÁO TRONG CÁC VÍ DỤ MINH HỌA

```
#define  N     10000
typedef  int keytype;
typedef  float othertype;
typedef  struct {
    keytype key;
    othertype others;
} recordtype;
```

GIỚI THIỆU

- HÀM HOÁN VỊ TRONG CÁC VÍ DỤ MINH HỌA: O(1)

```
void swap(recordtype *x, recordtype *y) {
        recordtype temp;
        temp = *x;
        *x = *y;
        *y = temp;
}
```

Bubble sort

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	A[5]	a[6]	a[7]	a[8]	a[9]
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**

Bubble sort

- GIẢI THUẬT
  - Bước 1: Xét các phần tử a[j] (j từ n-1 đến 1), nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán vị a[j] và a[j-1]. Sau bước này thì a[0] có khoá nhỏ nhất
  - Bước 2: Xét các phần tử a[j] (j từ n-1 đến 2), nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán vị a[j] và a[j-1]. Sau bước này thì a[1] có khoá nhỏ thứ 2
  - …
  - Sau n-1 bước thì kết thúc

Bubble sort

```
void bubble_sort(recordtype  a[], int n) {
    int i,j;
    for(i= 0; i<= n-2; i++)
          for(j=n-1;j>=i+1; j--)
                if (a[j].key < a[j-1].key)
                     swap(&a[j],&a[j-1]);
}
```

Bubble sort

- ĐỘ PHỨC TẠP
  - Số phép so sánh: $O(n^2)$
  - Số lần hoán vị tối đa: $O(n^2)$
  - Độ phức tạp: $O(n^2)$

Selection sort

Bước \ Khóa	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 0	**2**	6	5	2	10	12	9	10	9	3
Bước 1		**2**	5	6	10	12	9	10	9	3
Bước 2			**3**	6	10	12	9	10	9	5
Bước 3				**5**	10	12	9	10	9	6
Bước 4					**6**	12	9	10	9	10
Bước 5						**9**	12	10	9	10
Bước 6							**9**	10	12	10
Bước 7								**10**	12	10
Bước 8									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Selection sort

- GIẢI THUẬT
  - Bước 0, chọn phần tử có khóa nhỏ nhất trong n phần tử a[0] ,..., a[n-1] và hoán vị nó với a[0]
  - Bước 1, chọn phần tử có khóa nhỏ nhất trong n-1 phần tử a[1] ,..., a[n-1] và hoán vị nó với a[1]
  - Bước i, chọn phần tử có khoá nhỏ nhất trong n-i phần tử a[i], …, a[n-1] và hoán vị nó với a[i],
  - …
  - Sau n-1 bước này thì mảng đã được sắp xếp

Selection sort

```
void selection_sort(recordtype  a[], int n) {
    int i,j,lowindex;
    keytype lowkey;
    for (i=0; i<(n-1); i++) {
        lowindex = i; lowkey = a[i].key;
        for (j = i+1; j <n; j++)
                if (a[j].key < lowkey) {
                    lowkey = a[j].key; lowindex = j;
                }
        swap(&a[i],&a[lowindex]);
    }
}
```

Selection sort

```
void selection_sort2(recordtype  a[], int n) {
    int i,j,lowindex;
    for (i=0; i<(n-1); i++) {
        lowindex = i;
        for (j = i+1; j <n; j++)
                if (a[j].key < a[lowindex].key)
                    lowindex = j;
        swap(&a[i],&a[lowindex]);
    }
}
```

Selection sort

- ĐỘ PHỨC TẠP
  - Số phép so sánh: $O(n^2)$
  - Số phép giữ khóa bé tối đa: $O(n^2)$
  - Số lần hoán vị tối đa: $O(n)$
  - Độ phức tạp: $O(n^2)$

Insertion sort

Bước	Khóa	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	A[7]	a[8]	a[9]
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**

Insertion sort

- GIẢI THUẬT
  - Xem phần tử a[0] là một dãy đã có thứ tự
  - Bước 1: xen a[1] vào danh sách đã có thứ tự a[0] sao cho a[0], a[1] là danh sách có thứ tự
  - Bước 2, xen a[2] vào danh sách đã có thứ tự a[0], a[1] sao cho a[0], a[1], a[2] là một danh sách có thứ tự
  - Tổng quát, bước i, xen a[i] vào danh sách đã có thứ tự a[0], a[1], … a[i-1] sao cho a[0], a[1],.. a[i] là một danh sách có thứ tự.
  - Sau n-1 bước thì kết thúc

Insertion sort

```
void insertion_sort(recordtype  a[], int n) {
    int i,j;
    for (i = 1; i<= n-1; i++) {
        j = i;
        while ((j>0) && (a[j].key < a[j-1].key)) {
                swap(&a[j], &a[j-1]);
                j= j-1;
        }
    }
}
```

Insertion sort

- ĐỘ PHỨC TẠP
  - Số phép so sánh: $O(n^2)$
  - Số lần hoán vị tối đa: $O(n^2)$
  - Độ phức tạp: $O(n^2)$
  - Nếu mảng có thứ tự một phần => giải thuật thực hiện với độ phức tạp ít hơn rất nhiều

Quick sort

- GIẢI THUẬT
  - Chọn một giá trị khóa v làm chốt (pivot)
  - Phân hoạch dãy a[0]..a[n-1] thành 2 mảng con "trái" và "phải". Mảng con "trái" là các phần tử có khóa nhỏ hơn chốt v, mảng con "phải" là các phần tử có khóa lớn hơn hoặc bằng chốt v
  - Sắp xếp mảng con “trái” và mảng con “phải”
  - Việc sắp xếp mảng con “trái” và “phải” cũng được tiến hành bằng phương pháp trên

Quick sort

- CHỌN KHÓA CHỐT v
  - Chọn giá trị 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: cho mảng có khoá là 6, 6, 5, 8, 7, 4

    ta chọn chốt là 6 (khoá của phần tử đầu tiên)
  - Ví dụ: cho mảng có khoá là 6, 6, 7, 5, 7, 4, ta chọn chốt là 7 (khoá của phần tử thứ 3)
  - Ví dụ: cho mảng có khoá là 6, 6, 6, 6, 6, 6 thì không có chốt (các phần tử có khoá bằng nhau)
  - Ví dụ: cho mảng có một khoá là 6 thì không có chốt (do chỉ có một phần tử)

Quick sort

- PHÂN HOẠCH
  - Dùng 2 "con nháy" L và R trong đó L từ bên trái và R từ bên phải
  - Cho L chạy sang phải cho tới khi gặp phần tử có khóa ≥ chốt
  - 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, a[L] là phần tử đầu tiên của mảng con "bên phải"

Quick sort

L=0 R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	8	2	10	5	12	8	1	15	4

Chốt p = 8

Quick sort

L=1 R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	8	2	10	5	12	8	1	15	4

Chốt p = 8

Quick sort

L=1 R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	10	5	12	8	1	15	8

Chốt p = 8

Quick sort

L=2 R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	10	5	12	8	1	15	8

Chốt p = 8

Quick sort

L=3 R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	10	5	12	8	1	15	8

Chốt p = 8

Quick sort

			L=3						R=8
Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	10	5	12	8	1	15	8

Chốt p = 8

Quick sort

			L=3					R=7		
Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	10	5	12	8	1	15	8

Chốt p = 8

Quick sort

			L=3					R=7		
Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

Quick sort

L=4 R=7

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

Quick sort

					L=5				R=7	
Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

Quick sort

L=5 R=6

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

Quick sort

L=5 R=5

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

Quick sort

R=4 L=5

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

0	1	2	3	4	5	6	7	8	9
5	4	2	1	5	12	8	10	15	8

Quick sort

- GIẢI THUẬT SẮP XẾP MẢNG a[i]..a[j]
  - Xác định chốt
  - Phân hoạch mảng đã cho thành hai mảng con a[i]..a[k-1] và a[k]..a[j].
  - Gọi đệ quy sắp xếp mảng a[i]..a[k-1]
  - Gọi đệ quy sắp xếp mảng a[k]..a[j]
  - Đệ quy sẽ dừng khi không còn tìm thấy chốt

Quick sort

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	8	2	10	5	12	8	1	15	4

Chốt p = 8

	5	4	2	1	5	12	8	10	15	8

Chốt p = 5

Quick sort

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

	5	4	2	1	5	12	8	10	15	8
	1			5						

Chốt p = 5

	1	4	2	5	5					

Chốt p = 4

Quick sort

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

5 4 2 1 5 12 8 10 15 8

1 5

Chốt p = 5 Chốt p = 12

1 4 2 5 5

2 4

Chốt p = 4 xong

1 2 4

Chốt p = 2 xong

1 2

xong xong

Quick sort

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

	5	4	2	1	5	12	8	10	15	8
	1			5		8				12

Chốt p = 5 Chốt p = 12

1	4	2	5	5	8	8	10	15	12
	2	4							

Chốt p = 4 xong Chốt p = 10 Chốt p = 15

1	2	4		8	8	10

Chốt p = 2 xong xong xong

1	2

xong xong

Quick sort

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	5	4	2	1	5	12	8	10	15	8

Chốt p = 8

	5	4	2	1	5	12	8	10	15	8
	1			5		8				12

Chốt p = 5 Chốt p = 12

	1	4	2	5	5	8	8	10	15	12
		2	4						12	15

Chốt p = 4 xong Chốt p = 10 Chốt p = 15

	1	2	4			8	8	10	12	15

Chốt p = 2 xong xong xong xong xong

	1	2

xong xong

Quick sort

```
int find_pivot(recordtype  a[], int i, int j) {
    int k = i+1;
    keytype  firstkey = a[i].key;
    while ((k <= j) && (a[k].key == firstkey))
          k++;
    if (k > j)
          return -1;
    else
          if (a[k].key>firstkey)
                return k;
          else
                return i;
}
```

Quick sort

```
int partition(recordtype  a[], int i, int j, keytype pivot) {
    int L,R;
    L = i;
    R = j;
    while (L <= R) {
           while (a[L].key < pivot)  L++;
           while (a[R].key >= pivot) R--;
           if (L<R)
                  swap(&a[L],&a[R]);
    }
    return L;
}
```

Quick sort

```
void quick_sort(recordtype  a[], int i, int j) {
    keytype pivot;
    int pivotindex, k;
    pivotindex = find_pivot(a, i, j);
    if (pivotindex != -1) {
           pivot = a[pivotindex].key;
           k = partition(a, i, j, pivot);
           quick_sort(a, i, k-1);
           quick_sort(a, k, j);
    }
}
```

Quick sort

- ĐỘ PHỨC TẠP
  - Hàm find_pivot luôn tìm được chốt và đệ quy chỉ dừng khi kích thước bài toán bằng 1
  - T(n): độ phức tạp của quick_sort (n phần tử)
  - Độ phức tạp của find_pivot và partition là O(n) = n
  - Khi n = 1, quick_sort gọi find_pivot với độ phức tạp là O(1) =1
  - Trường hợp xấu nhất, phân hoạch bị lệch (phần tử chốt ngay cuối dãy số)

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

  - $T(n) = O(n^2)$

Quick sort

- ĐỘ PHỨC TẠP
  - T(n): độ phức tạp của quick_sort (n phần tử)
  - Độ phức tạp của find_pivot và partition là O(n) = n
  - Khi n = 1, quick_sort gọi find_pivot với độ phức tạp là O(1) =1
  - Trường hợp tốt nhất, phân hoạch cân bằng (phần tử chốt ngay giữa mảng)

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

  - T(n) = O(nlog(n))

Heap sort

- Ý TƯỞNG
  - Dựa trên cấu trúc heap hay priority queue
  - Cây nhị phân đầy đủ có nút gốc với độ ưu tiên cao hơn bất kỳ nút nào của 2 cây con: nút gốc có độ ưu tiên cao nhất
  - Lần lượt thực hiện cắt bỏ nút gốc và xây dựng lại cấu trúc heap cho các phần tử còn lại, quá trình lặp lại đến khi nào chỉ còn 1 phần tử
  - Các nút bị cắt tạo thành 1 dãy có thứ tự

Heap sort

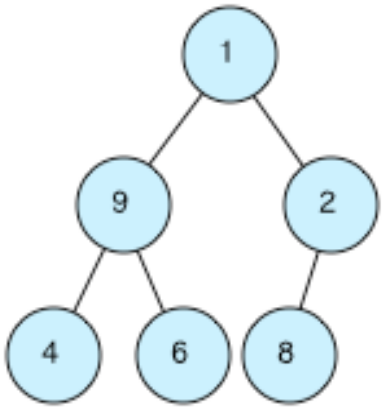
- GIẢI THUẬT

1. Xem mảng là một cây nhị phân. Mỗi nút trên cây lưu trữ một phần tử mảng, trong đó a[0] là nút gốc và mỗi nút không là nút lá a[i] có con trái là a[2i+1] và con phải là a[2i+2]. Với cách tổ chức này thì cây nhị phân thu được sẽ có các nút trong là các nút a[0], …, a[(n-2)/2]. Tất cả các nút trong đều có 2 con, ngoại trừ nút a[(n-2)/2] có thể chỉ có một con trái
2. Sắp xếp cây ban đầu thành một heap
3. Hoán vị nút gốc a[0] cho cho nút lá cuối cùng
4. Sắp lại cây sau khi đã bỏ đi nút lá cuối để tạo một heap mới
5. Lặp lại quá trình (3) và (4) cho tới khi cây chỉ còn một nút, *nút này cùng với các nút lá đã bỏ đi tạo thành một mảng sắp theo thứ tự*

Heap sort

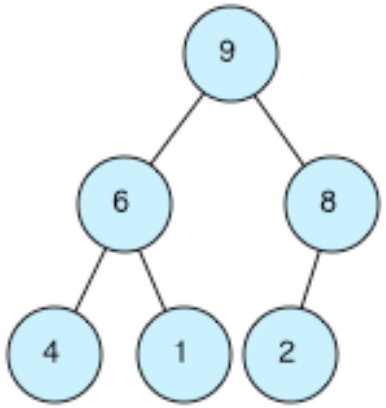
Original array

index	0	1	2	3	4	5
value	1	9	2	4	6	8

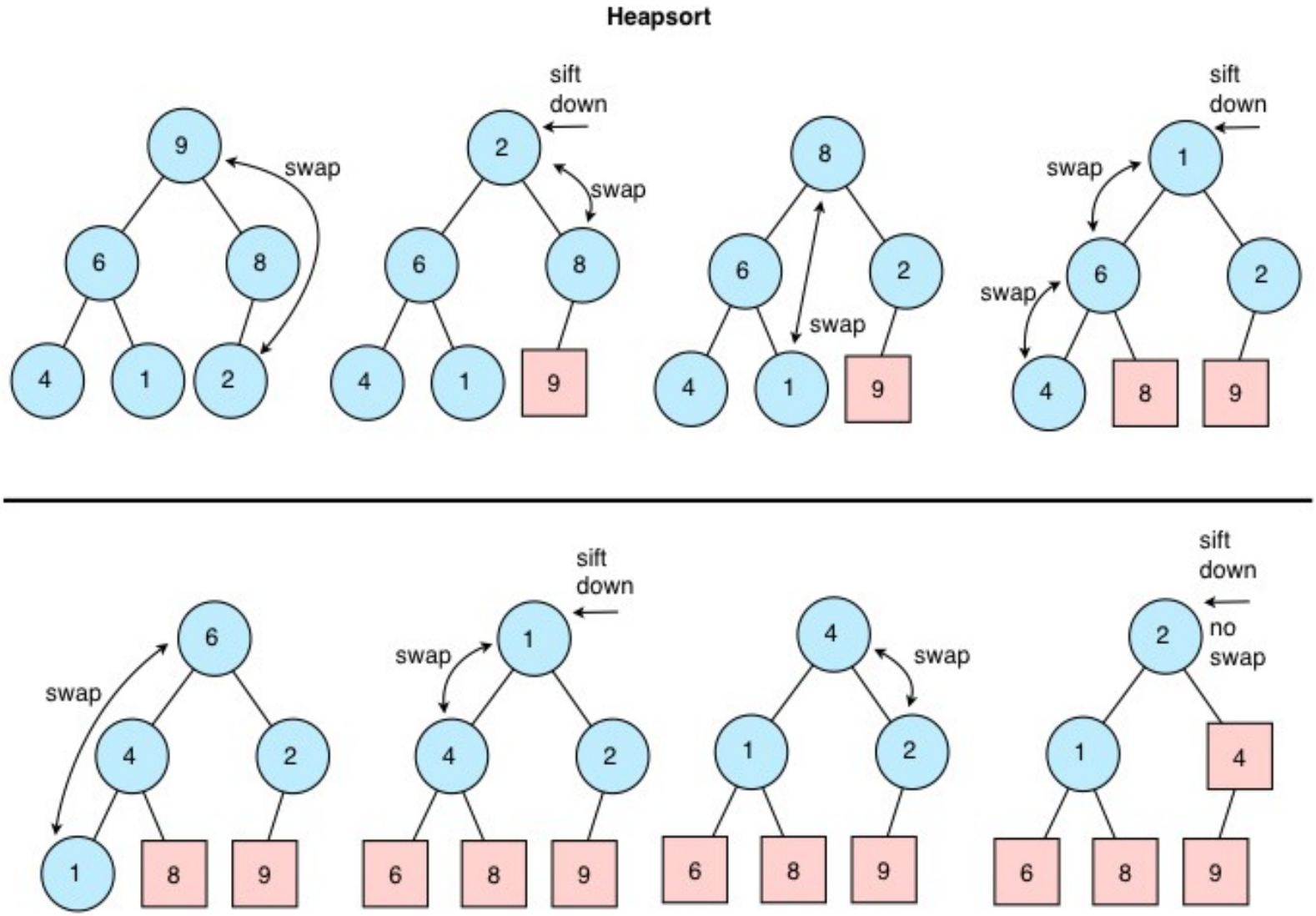


Heap array

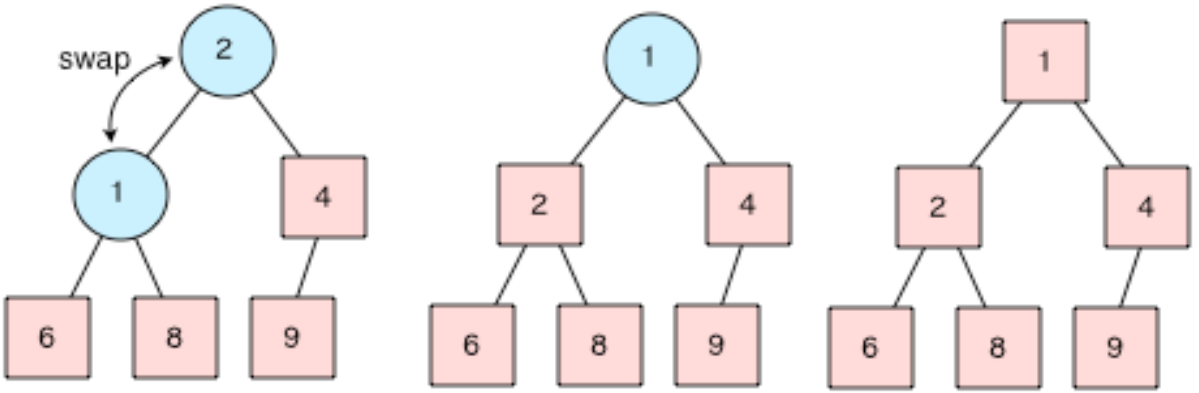
index	0	1	2	3	4	5
value	9	6	8	4	1	2



Heap sort


Heapsort
9
swap
6
8
4
1
2
sift
down
2
swap
6
8
4
1
9
8
6
2
swap
4
1
9
sift
down
1
swap
6
2
swap
4
8
9
6
swap
1
sift
down
4
2
1
8
9
swap
4
1
2
6
8
9
sift
down
2
no
swap
1
4
6
8
9

Heap sort



Sorted array

index	0	1	2	3	4	5
value	1	2	4	6	8	9

Heap sort

- push_down(a[], first, last) để đẩy a[first] xuống đúng vị trí của nó trong cây
  - Nếu a[first] chỉ có con trái và nếu khoá của nó nhỏ hơn khoá của con trái (a[first].key < a[2*first+1].key) thì hoán đổi a[first] cho con trái của nó và kết thúc
  - Nếu a[first] có khoá nhỏ hơn khóa con trái và khoá con trái lớn hơn khoá con phải thì hoán đổi a[first] cho con trái của nó, có thể con trái sẽ không đúng vị trí nên phải xem xét lại con trái để đẩy xuống
  - Nếu a[first] có khoá nhỏ hơn khoá con phải và khoá của con phải lớn hơn khoá của con trái thì hoán đổi a[first] cho con phải, có thể 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 xãy ra thì a[first] đã đúng vị trí

Heap sort

```
void pushdown(recordtype  a[], int first, int last) {
     int r = first;
     while (r <= (last-1)/2)
        if (last == 2*r+1) {
                if (a[r].key < a[last].key)
                     swap(&a[r],&a[last]);
                r = last;
        } else
        if ((a[r].key<a[2*r+1].key) && (a[2*r+1].key>=a[2*r+2].key)) {
                     swap(&a[r],&a[2*r+1]); r = 2*r+1;
        } else
        if ((a[r].key<a[2*r+2].key) && (a[2*r+2].key>a[2*r+1].key)) {
                     swap(&a[r],&a[2*r+2]); r = 2*r+2 ;
        } else
                     r = last;
}
```

Heap sort

```
void heap_sort(recordtype  a[], int n) {
    int i;
    for(i = (n-2)/2; i>=0; i--)              /* 1 */
          pushdown(a, i, n-1);               /* 2 */
    for(i = n-1; i>=2; i--) {                /* 3 */
          swap(&a[0],&a[i]);                 /* 4 */
          pushdown(a, 0, i-1);               /* 5 */
    }
    swap(&a[0],&a[1]);
}
```

Heap sort

- ĐỘ PHỨC TẠP
  - Độ phức tạp push_down(a[], 0, n-1): O(logn)
  - Tạo heap (1-2): O(nlogn)
  - Vòng lặp cắt và tạo heap (3-5), lặp n-2 lần, mỗi lần lấy O(logn): O(nlogn)