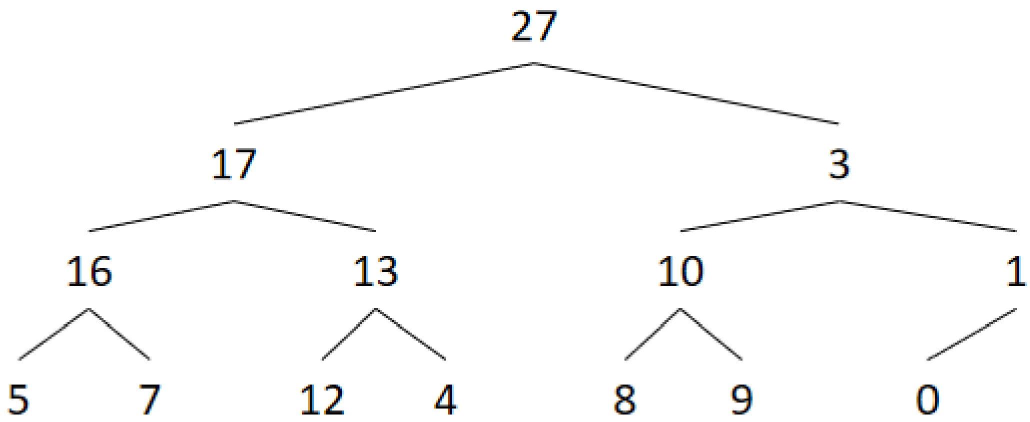
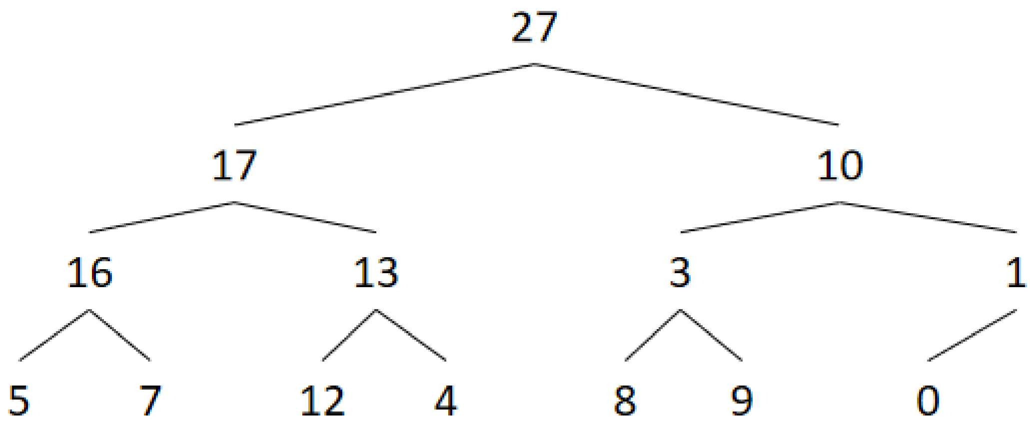
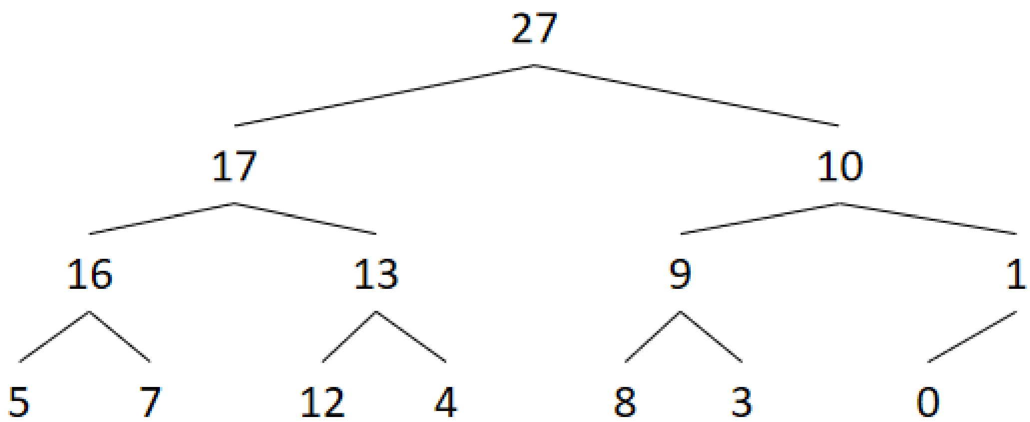
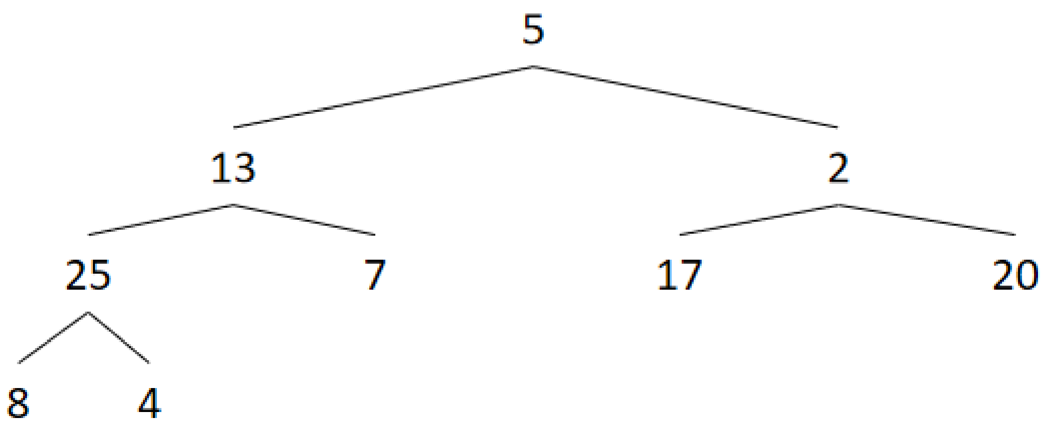
6.2-1

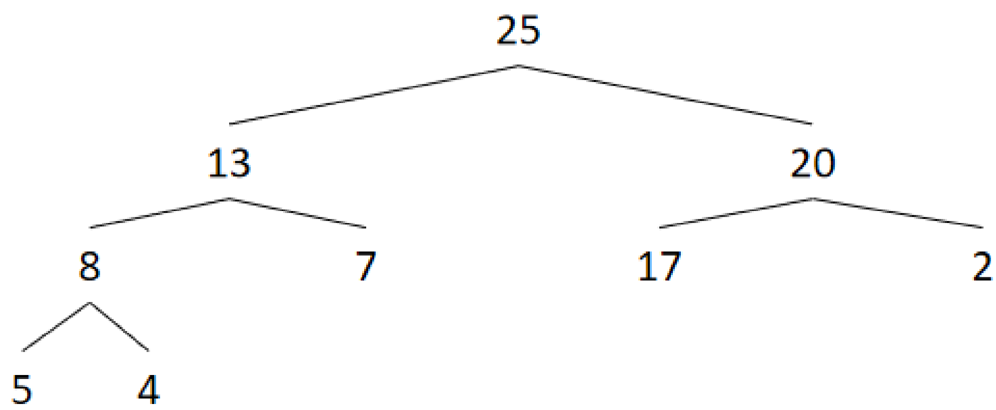

27
17
3
16
13
10
1
5
7
12
4
8
9
0

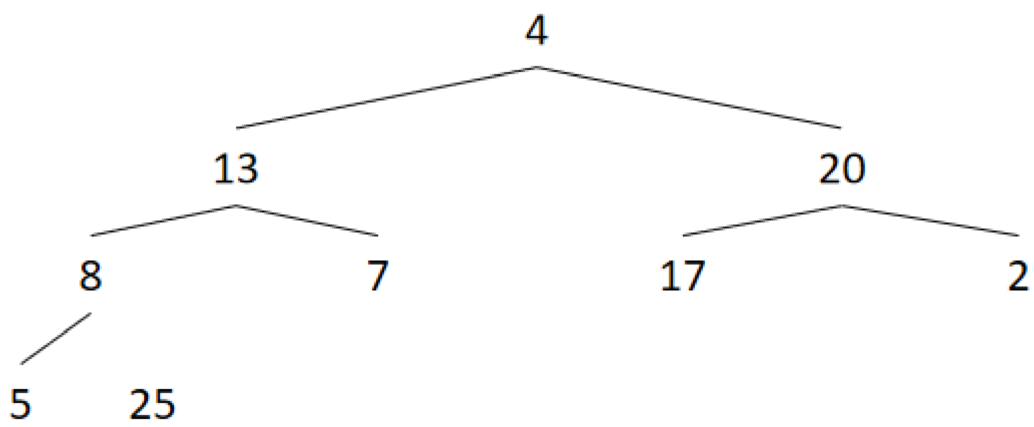

27
17
10
16
13
3
1
5
7
12
4
8
9
0

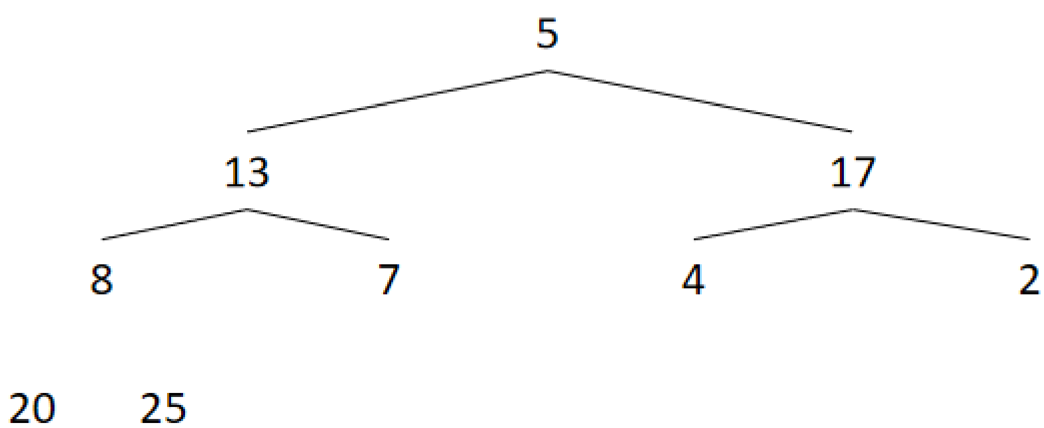

27
17
10
16
13
9
1
5
7
12
4
8
3
0

6.4-1

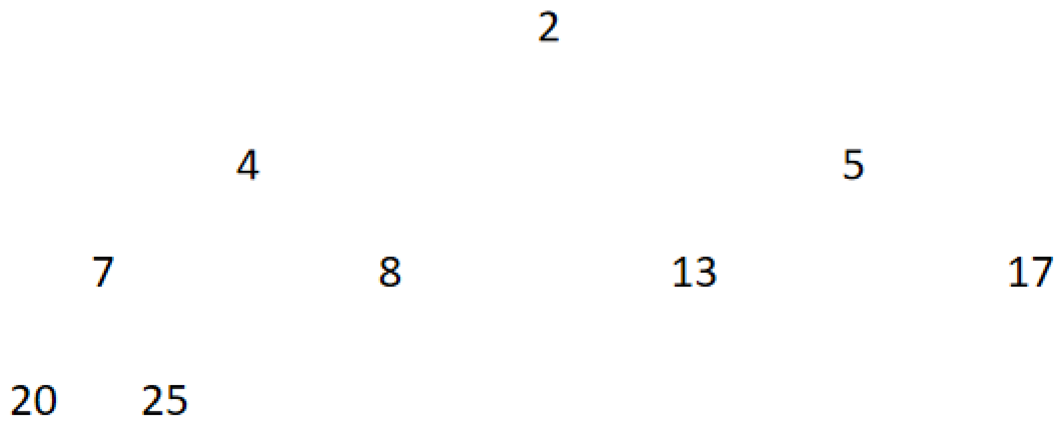

5
13
2
25
7
17
20
8
4


25
13
20
8
7
17
2
5
4


4
13
20
8
7
17
2
5
25

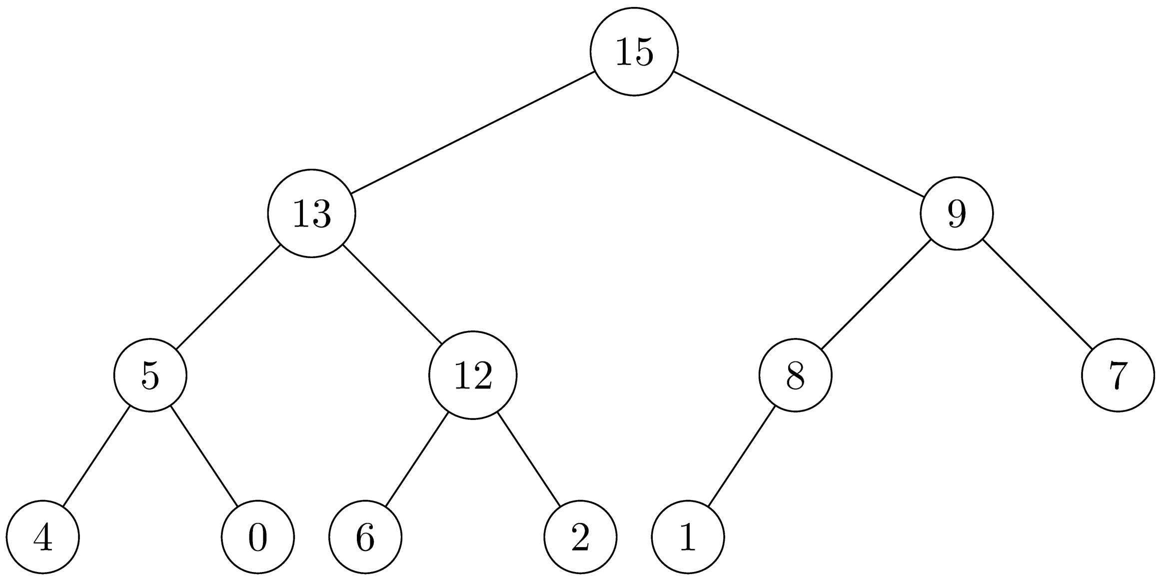

5
13
17
8
7
4
2
20
25

以此类推最后得到

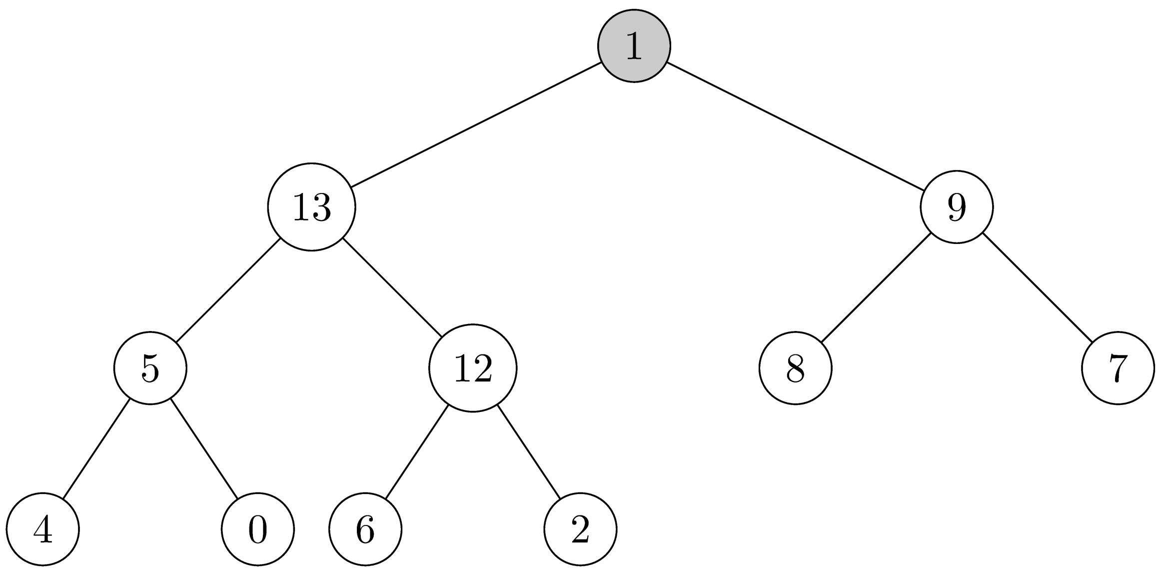


6.5-2

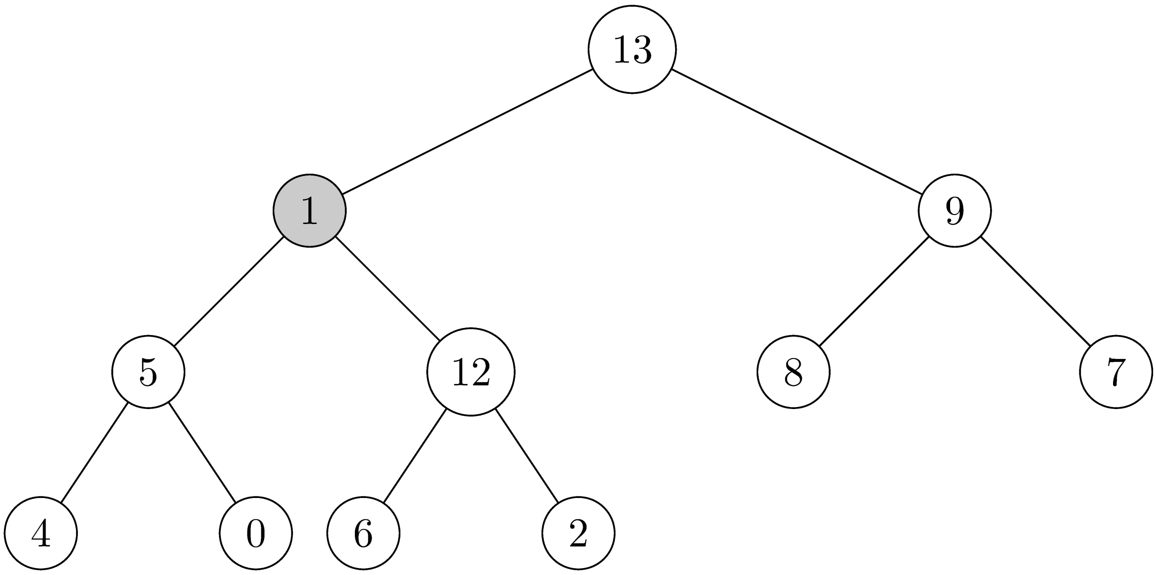
原始堆



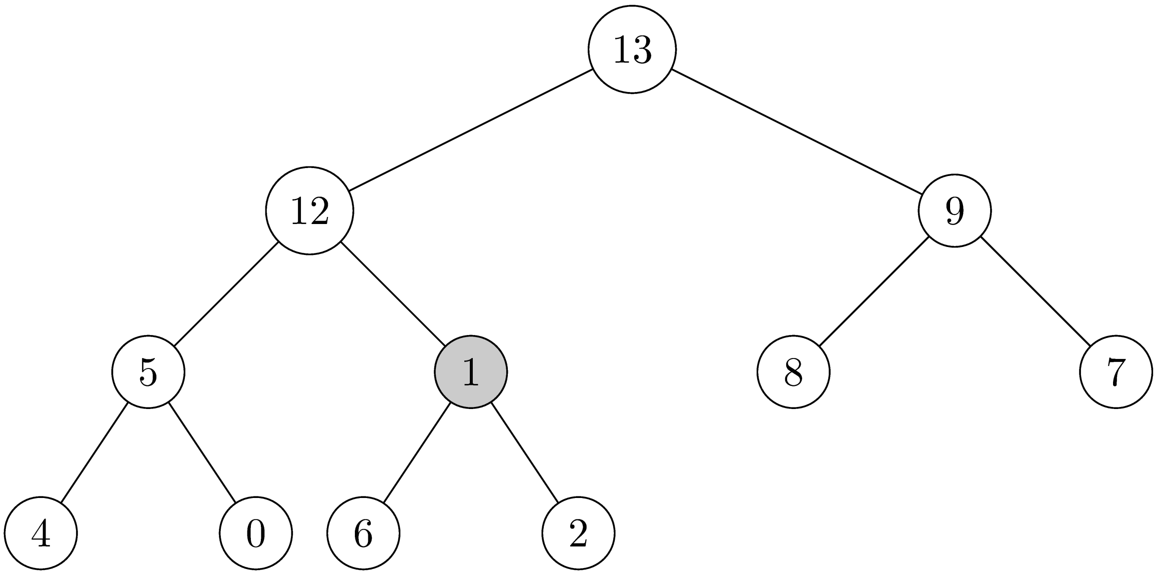
提取最大节点15，移动1至节点顶部



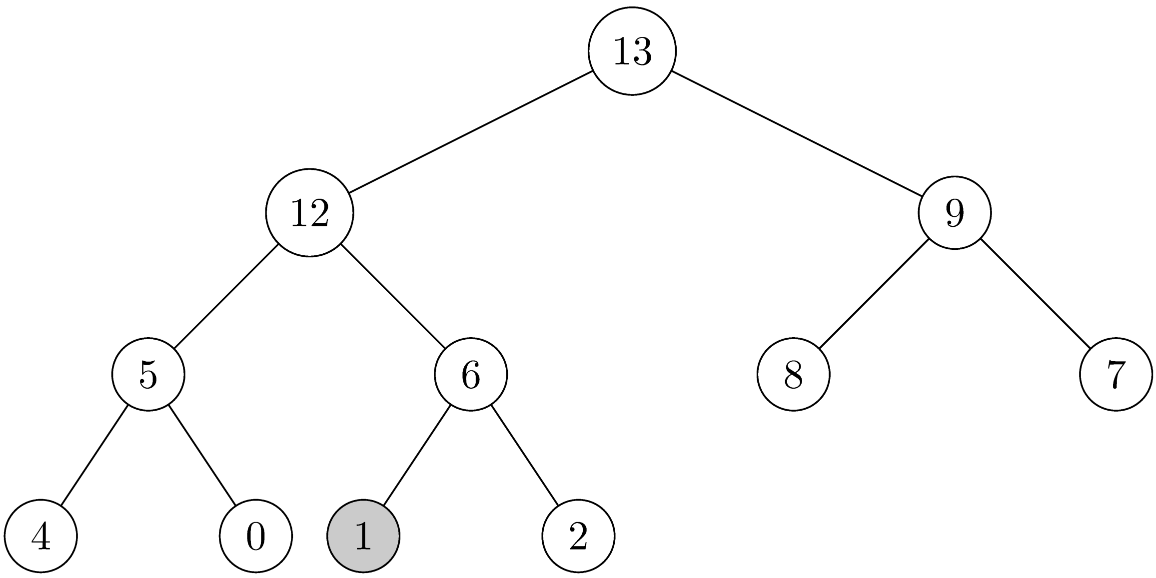
交换1和13



交换1和12



交换1和6



提取最大节点13，移动2至节点顶部；以此类推，依次移出最大节点

7.1-1

13	19	9	5	12	8	7	4	21	2	6	11

9	19	13	5	12	8	7	4	21	2	6	11

9	5	13	19	12	8	7	4	21	2	6	11

9	5	8	19	12	13	7	4	21	2	6	11

9	5	8	7	12	13	19	4	21	2	6	11

9	5	8	7	4	13	19	12	21	2	6	11

9	5	8	7	4	2	19	12	21	13	6	11

9	5	8	7	4	2	6	12	21	13	19	11

9	5	8	7	4	2	6	11	21	13	19	12

7.5

a. $p_i = \frac{(i-1)(n-i)}{C_n^3} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$

b. $\frac{6(\lfloor\frac{n+1}{2}\rfloor-1)(n-\lfloor\frac{n+1}{2}\rfloor)}{n(n-1)(n-2)} - \frac{1}{n}$

$\underset{n\to\infty}{lim} = \frac{\frac{6(\lfloor\frac{n+1}{2}\rfloor-1)(n-\lfloor\frac{n+1}{2}\rfloor)}{n(n-1)(n-2)}}{\frac{1}{n}} = \frac{3}{2}$

c. $\int_{\frac{n}{3}}^{\frac{2n}{3}} \frac{6(n-x)(x-1)}{n(n-1)(n-2)}dx = \int_{\frac{n}{3}}^{\frac{2n}{3}} \frac{6(-x^2+nx+x-n)}{n(n-1)(n-2)}dx = \frac{6(-7n^3/81+3n^3/18+3n^2/18-n^2/3)}{n(n-1)(n-2)} = \frac{13}{27}$

d. 即使我们总是选择中间元素作为枢轴（这是最好的情况），递归树的高度仍然是$\Theta(lgn)$,因此快速排序的时间复杂度仍然为$\Omega(nlgn)$

8.2-1

$C =< 2,2,2,2,1,0,2 >$

$C =< 2,4,6,8,9,9,11 >$

先把2放在正确的位置，$B[C[A[j]]] = A[j], B[C[2]] = B[6] = 2$, 说明2放在第六个位置。

$B =< 0,0,1,1,2,2,3,3,4,6,6 >$

8.3-2

插入排序和归并排序是稳定的，堆排序和快速排序不是稳定的。

为了使任何排序算法稳定，我们可以对数据预处理，用有序对替换数组的每个元素。第一个条目将是元素的值，第二个值将是元素的索引。

比如，数组[2, 1, 1, 3, 4, 4, 4]可以表示为[(2, 1), (1, 2), (1, 3), (3, 4), (4, 5), (4, 6), (4, 7)]

定义，如果满足i<=k且j<m，则(i, j) < (k, m)。

在该定义下，算法保证是稳定的，因为我们的每个新元素都是不同的，索引比较确保如果重复元素在原始数组中出现得更晚，它必须出现在排序数组的后面。这使空间需求加倍，但运行时间将逐渐保持不变。

9.2-4

当选择的分区始终是数组的最大元素时，我们得到最坏情况的性能。

依次选取主元：$<9,8,7,6,5,4,3,2,1,0>$

9.3-1

在$\lfloor n/7 \rfloor$个组中，除了当n不能被7整除时所含的元素少于7的那个组合包含x的那个组之外，至少有一半的组中有4个元素大于x。

不算这两个组，大于x的元素至少为

$4(\lceil \frac{1}{2} \lceil \frac{n}{7} \rceil \rceil - 2) \geq \frac{2n}{7} - 8$

同理小于x的元素至少有$\frac{2n}{7} - 8$, 分区最多会将子问题减小到大小$5n/7+8$，得出以下递推式

$$T(n) = \begin{cases} O(1) \quad if\ n < n_0 \\ T(\lceil n/7 \rceil) + T(5n/7+8) + O(n) \quad if\ n \geq n_0 \end{cases}$$

用替换法来证明运行时间是线性的，即证明对某个适当大的常数c和所有的$n>0$，有$T(n) \leq cn, O(n)$有上界an

$$\begin{aligned} T(n) &\leq c\lceil n/7 \rceil + c(5n/7+8) + an \\ &\leq cn/7 + c + 5cn + 7 + 8c + an \\ &= 6cn/7 + 9c + an = cn + (-cn/7 + 9c + an) \leq cn = O(n) \end{aligned}$$

需要满足$-cn/7 + 9c + an \leq 0$，即$c \geq \frac{7an}{n-63}$

$n_0 = 126, n \leq n_0, n/(n-63) \leq 2, c \geq 14a$即可。

在$\lfloor n/3 \rfloor$个组中，除了当n不能被3整除时所含的元素少于3的那个组合包含x的那个组之外，至少有一半的组中有2个元素大于x。

不算这两个组，大于x的元素至少为

$2(\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil \rceil - 2) \geq \frac{n}{3} - 4$

同理小于x的元素至少有$\frac{n}{3} - 4$, 分区最多会将子问题减小到大小$2n/3+4$，得出以下递推式

$$T(n) = \begin{cases} O(1) \quad if\ n < n_0 \\ T(\lceil n/3 \rceil) + T(2n/3+4) + O(n) \quad if\ n \geq n_0 \end{cases}$$

用替换法来证明运行时间满足$T(n) = \omega(n)$,假定 $T(n) > cn, , O(n)$有上界an

$$\begin{aligned} T(n) &> c\lceil n/3 \rceil + c(2n/3+2) + an \\ &> cn/3 + c + 2cn/3 + 2c + an = cn + 3c + an > an = \omega(n), c > 0, a > 0, n > 0 \end{aligned}$$

9.3-6

（1）如果k = 1，则返回一个空列表

（2）如果k是偶数，找到对应中位数，以中位数为界，划分为两个子问题$\lfloor n/2 \rfloor$,返回对应解决方案和中位数

（3）如果k是奇数，发现⌊ k / 2 ⌋和⌈ k / 2 ⌉ 边界，减少到两个子问题，最坏递归情况：

$T(n,k) = 2T(\lfloor n/2 \rfloor, k/2) + O(n)$

```
#include <iostream>
using namespace std;
void exchange(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
int partition(int A[],int p, int r, int key)
{
    int i = p - 1,j;
    for(j=p;j<r; ++j)
        if(A[j]==key)
            break;
    exchange(A[j],A[r]);
    for(j=p; j<r; ++j)
    {
        if(A[j]<=key){
            i++;
            exchange(A[i],A[j]);
        }
    }
    exchange(A[i+1],A[r]);
    return i+1-(p-1);
}
void insert_sort(int A[],int p, int r)
{
    if(p>=r)
        return;
    int key,i,j;
    for(j=p+1;j<=r; j++)
    {
        key = A[j];i=j-1;
        while(i>=p && A[i]>key)
        {
            A[i+1] = A[i];
            i = i-1;
        }
        A[i+1] = key;
    }
}
int select(int A[],int p, int r, int i)
{
    if(p==r)
        return A[p];
```

```
    int k = p+4;
    while(k<=r)
    {
        insert_sort(A, k-4, k);
        k += 5;
    }
    insert_sort(A, k-4, r);
    int num;
    num = (r-p+1)/5 + ( (r-p+1)%5 ? 1:0 );
    int *new_arr = new int[num];
    for(int j=0;j<num-1;++j)
        new_arr[j] = A[3+j*5+p-1];
    //找出最后一个中位数
        k = r-(num-1)*5 - (p - 1);
    if(k%2)k = k + 1;
    k = k/2;
    k += (num-1)*5 + p - 1;
    new_arr[num-1] = A[k];
    //对[n/5]个中位数进行插入排序，找出中位数的中位数x
    insert_sort(new_arr, 0, num-1);
    int x = new_arr[(num-1)/2];
    delete new_arr;
    //按中位数x对输入数组进行划分
    k = partition(A,p,r,x);
    if(k==i)
        return x;
    else if(k>i)
        select( A, p, k-1 + (p-1), i);
    else
        select( A, k+1 + (p-1), r, i-k);
}
void find_k(int A[],int p, int r, int k)
{
    if(k<1)return;
    int len = (r-p+1)/k;
    int i;
    i = k/2;
    if(i>0){
        select(A,p,r,i*len);
        find_k(A,p,i*len+(p-1),k/2);
        find_k(A,i*len+p,r,k%2?k/2+1:k/2);
    }
}
void test()
{
    int A[] = {0,5,15,14,13,12,9,10,7,3,2,4,6,8,1,11,17,16,18,20,19};
    find_k(A,1,20,5);
    for(int i=1;i<5; ++i)
        cout<<A[i*4]<<" ";
    cout<<endl;
}
int main()
{
    test();
    return 0;
}
```

10.1-1

操作	栈
PUSH(*S*,4)	4
PUSH(*S*,1)	4 1
PUSH(*S*,3)	4 1 3
POP(*S*)	4 1
PUSH(*S*,8)	4 1 8
POP(*S*)	4 1

10.1-4

```
QUEUE-EMPTY(Q)
    if Q.head == Q.tail
        return true
    else return false
```

```
QUEUE-FULL(Q)
    if Q.head == Q.tail + 1 or (Q.head == 1 and Q.tail == Q.length)
        return true
    else return false
```

```
ENQUEUE(Q, x)
    if QUEUE-FULL(Q)
        error "overflow"
    else
        Q[Q.tail] = x
        if Q.tail == Q.length
            Q.tail = 1
        else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
    if QUEUE-EMPTY(Q)
        error "underflow"
    else
        x = Q[Q.head]
        if Q.head == Q.length
            Q.head = 1
        else Q.head = Q.head + 1
        return x
```

10.2-5

```c
#include <stdlib.h>

typedef struct node_t {
    int key;
    struct node_t *next;
} node_t;

typedef struct {
    struct node_t nil;
} list_t;

void init_list(list_t *list) {
    list->nil.key = 0;
    list->nil.next = &(list->nil);
}

void destroy_list(list_t *list) {
    node_t *node = list->nil.next;
    node_t *next;

    while (node != &(list->nil)) {
        next = node->next;
        free(node);
        node = next;
    }
}

void insert(list_t *list, int key) {
    node_t *new = (node_t *) malloc(sizeof(node_t));
    new->key = key;
    new->next = list->nil.next;
    list->nil.next = new;
}

node_t *search(list_t *list, int key) {
    node_t *node = list->nil.next;

    list->nil.key = key;
    while (node->key != key) {
        node = node->next;
    }

    if (node == &(list->nil)) {
        return NULL;
    } else {
        return node;
    }
}

void delete(list_t *list, int key) {
    node_t *node = &(list->nil);

    while (node->next != &(list->nil)) {
        if (node->next->key == key) {
            node_t *to_be_deleted = node->next;
```

```
            node->next = node->next->next;
            free(to_be_deleted);
        } else {
            node = node->next;
        }
    }
}
```

10.3-1

$index$	1	2	3	4	5	6	7
$next$		3	4	5	6	7	/
key		13	4	8	19	5	11
$prev$		/	2	3	4	5	6

$index$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
key	13	4	/	4	7	1	8	10	4	19	13	7	5	16	10	11	/	13

10-1

	未排序的单链表	**已排序的单链表**	**未排序的双向链表**	**已排序的双向链表**
$SEARCH(L,k)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
$INSERT(L,x)$	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
$DELETE(L,x)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
$SUCCESSOR(L,x)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
$PREDECESSOR(L,x)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$
$MINIMUM(L,x)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
$MAXIMUM(L)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$