第十章

10.3

希尔排序 快速排序 堆排序是不稳定的

{4, 3, 4, 2}

10.7

1 2 3 4 5 6 7 序列 第一次选4 比较6次 第二次选2和6 比较4次 一共10次

10.25

```
void sort(SqList &L){
    int k=L.length;
    L.r[0].next=1;
    L.r[1].next=0;
    int i,j,pre;
    for(i=2;i<k;i++){
        j=L.r[0].next;
        if(L.r[i].data<L.r[j].data){
            L.r[i].next=L.r[0].next;
            L.r[0].next=i;
        } else{
            pre=j;
            while(L.r[i].data>L.r[j].data&&j!=0){
                pre=j;
                j=L.r[j].next;
            }
            L.r[pre].next=i;
            L.r[i].next=j;
        }
    }
}
```

10.27

```
void sort(vector<int> &a) {
    int dir = 1;
    int n = a.size();
    int st = 0, ed = n, st2 = n - 1, ed2 = -1;
    for (int i = 1; i < n; i++) {
        for (int j = st; j != ed; j += dir) {
            if ((a[j] - a[j + dir]) * dir > 0) swap(a[j], a[j + dir]);
        }
        swap(st, st2);
        swap(ed, ed2);
    }
}
```

10.28

```cpp
void sort(vector<int> &a) {
    int st = 0, ed = a.size() - 1;
    while (st != ed) {
        int id = getMaxElement(a, st, ed);
        swap(a[id], a[ed]);
        ed--;
        if (st == ed) break;
        id = getMinElement(a, st, ed);
        swap(a[id], a[st]);
        st++;
    }
}
```

10.33

```cpp
ListNode* insertionSortList(ListNode* head) {
    if (head == nullptr) return head;
    ListNode* newHead = new ListNode();
    newHead->next = head;
    ListNode* now = head->next;
    ListNode* lst = head;
    while (now != nullptr) {
        if (lst->val <= now->val) {
            lst = lst->next;
            now = lst->next;
            continue;
        }
        ListNode* pre = newHead;
        while (pre->next != now && pre->next->val < now->val) {
            pre = pre->next;
        }
        lst->next = now->next;
        now->next = pre->next;
        pre->next = now;
        now = lst->next;
    }
    return newHead->next;
}
```

10.35

```cpp
void HeapAdjust(HeapType &h, int s, int m) {
    int j;
    RedType rc;
    rc.key = h.r[s].key;
    for (j = s * 3 - 1; j <= m; j = 3 * j - 1) {
        if (j < m && h.r[j] < h.r[j + 1]) {
            j++;
            if (j < m && h.r[j] < h.r[j + 1]) {
                j++;
            }
        }
        swap(h.r[s], h.r[j]);
        s = j;
    }
```

```
    h.r[s] = rc;
}
void HeapSort (HeapType  &h ) {
     int i;
     if (h. length ) {
         for (i = (h. length + 1 ) / 3 ;i > 0 ; --i ) {
            HeapAdjust (h,i,h. length );
         }
         for (i =h. length ;i > 1 ; --i ) {
            Swap (h. r [ 1 ],h. r [i ] );
            HeapAdjust (h, 1,i - 1 );
         }
     }
}
```

10.37

```
ListNode* sortList(ListNode* head, ListNode* tail) {
    if (head == nullptr) return head;
    if (head->next == tail) {
        head->next = nullptr;
        return head;
    }
    ListNode *slow = head, *fast = head;
    while (fast != tail && fast->next != tail) {
        fast = fast->next->next;
        slow = slow->next;
    }
    return merge(sortList(head, slow), sortList(slow, tail));
}
ListNode* merge(ListNode* head1, ListNode* head2) {
    ListNode *newHead = new ListNode(0);
    ListNode *now = newHead, *temp1 = head1, *temp2 = head2;
    while (temp1 != nullptr && temp2 != nullptr) {
        if (temp1->val <= temp2->val) {
            now->next = temp1;
            temp1 = temp1->next;
        } else {
            now->next = temp2;
            temp2 = temp2->next;
        }
        now = now->next;
    }
    if (temp1 != nullptr) now->next = temp1;
    if (temp2 != nullptr) now->next = temp2;
    return newHead->next;
}
ListNode* sortList(ListNode* head) {
    return sortList(head, nullptr);
}
```

10.38

```
void merge(LinkList &lt1,LNode* order[],int i,int j,int k,int length){
    LinkList lt2;
```

```
    lt2=(LNode*)malloc(sizeof(LNode));lt2->next=NULL;
    LNode *p,*q,*post1,*post2,*r,*t;
    p=order[i];q=order[j];post1=q;
    if(j+length<k) post2=order[j+length];
        else post2=NULL;
    r=lt2;

    while(p!=post1&&q!=post2){
        t=(LNode*)malloc(sizeof(LNode));t->next=NULL;
        if(p->data<q->data){
            t->data=p->data;
            r->next=t;r=t;
            p=p->next;
        }else{
            t->data=q->data;
            r->next=t;r=t;
            q=q->next;
        }
    }

    while(p!=post1){
        t=(LNode*)malloc(sizeof(LNode));t->next=NULL;
        t->data=p->data;
        r->next=t;r=t;
        p=p->next;
    }
    while(q!=post2){
        t=(LNode*)malloc(sizeof(LNode));t->next=NULL;
        t->data=q->data;
        r->next=t;r=t;
        q=q->next;
    }

    LNode *w,*e;

    if(i==0) lt1->next=lt2->next;
    else{
        e=lt1->next;
        while(e->next->data!=order[i]->data) e=e->next;
        e->next=lt2->next;
    }
    if(j<k-1){
        r->next=order[j+length];
    }

    w=lt1->next;
    while(w->data!=lt2->next->data) w=w->next;
    order[i]=w;
}

```