第八次书面作业

19336035 陈梓乐
部分代码可以在 gitee.com/Czile/homework 上找到

简答题

1. 证明基于比较的排序算法时间复杂度至少是 $n\log_2 n$ 。

证明： 已知序列的所有排列可能性为$A_n^n = n!$，而给定序列是所有可能性的一种，应用信息论相关理论，给定序列包含的信息量为

$$H(\vec{X}) = \log_2(n!)$$

每一次比较，其实是确定两个元素的序，理想情况下获得的信息量为1 bit，因为每一次比较都可以否定上述可能性的一半[1]。于是需要 $\log_2(n!) \sim n\log_2 n$ 次比较才能获得足够的信息。

[1] 若比较不能否定上述可能性的一半，即比较并非等概率，则获得的信息量小于1 bit。

2. 数据序列为(12, 5, 9, 20, 6, 31, 24)，分别用插入排序、选择排序、快速排序、冒泡排序、堆排序、归并排序进行排序。

次数	插入排序							选择排序						
0	12	5	9	20	6	31	24	12	5	9	20	6	31	24

次数	插入排序							选择排序						
1	5	12	9	20	6	31	24	5	12	9	20	6	31	24
2	5	9	12	20	6	31	24	5	6	9	20	12	31	24
3	5	9	12	20	6	31	24	5	6	9	20	12	31	24
4	5	6	9	12	20	31	24	5	6	9	12	20	31	24
5	5	6	9	12	20	31	24	5	6	9	12	20	31	24
6	5	6	9	12	20	31	24	5	6	9	12	20	31	24

次数	快速排序							冒泡排序						
0	12	5	9	20	6	31	24	12	5	9	20	6	31	24
1	5	9	6	12	20	31	24	5	9	12	6	20	24	31
2	5	6	9	12	20	24	31	5	9	6	12	20	24	31
3	5	6	9	12	20	24	31	5	6	9	12	20	24	31

次数	堆排序							归并排序						
0	12	5	9	20	6	31	24	12	5	9	20	6	31	24
建堆	31	20	24	5	6	9	12	-						
1	24	20	12	5	6	9	31	5	12	9	20	6	31	24
2	20	9	12	5	6	24	31	5	9	12	20	6	24	31
3	12	9	6	5	20	24	31	5	6	9	12	20	24	31
4	9	5	6	12	20	24	31							
5	6	5	9	12	20	24	31							
6	5	6	9	12	20	24	31							

3. 已知下列初始状态，求直接插入排序至少需要多少次比较。

升序

n-1次。

降序

$$\sum_{k=1}^{n}\log k \Rightarrow n\log n$$

这里采用二分的比较法。

奇数关键码和偶数关键码分别升序

shellSort的最后一步，n-1次。

前半关键码和后半关键码分别升序

对于插入排序而言此状态优化不大，仍然是$n\log n$的时间代价，但使用归并排序的策略进行优化，可在$O(n)$的时间代价下完成，最少比较n-1次。

4. 对于n=7，给定快速排序的最好情况和最差情况实例

最好情况							最差情况						
4	1	3	2	6	5	7	7	6	5	4	3	2	1

5. 判断序列是否为堆，若不是，将它调整为堆。

(1, 5, 7, 25, 21, 8, 8, 42)

是。

(3, 9, 5, 8, 4, 17, 21, 6)

次数	序列							
0	3	9	5	8	4	17	21	6
1	3	9	5	6	4	17	21	8
1	3	4	5	6	9	17	21	8

算法设计题

1. 直接插入排序寻找插入位置可用折半搜索实现，据此写一个改进的插入排序算法。

```
template <class ForwardIterator, class T>
ForwardIterator upper_bound(
    ForwardIterator first,
    ForwardIterator last,
    const T& val
) {
    ForwardIterator it;
    iterator_traits<ForwardIterator>::difference_type count, step;
    count = std::distance(first,last);
    while (count>0) {
        it = first;
        step=count/2;
        std::advance (it,step);
        if (!(val<*it)) {
            first=++it;
            count-=step+1;
        }
        else count=step;
    }
    return first;
}
template <class InputIterator>
void insertSort(InputIterator first, InputIterator last) {
    for (auto i = first + 1; i != last; ++i) {
        auto pos = upper_bound(first, i, *i);
        auto val = *i;
        for (auto j = i; j != pos; --j)
            *j = *(j-1);
        *pos = val;
    }
}
```

2. 设待排序的记录序列用单链表作为储存结构，试写出直接插入排序算法。

由于第一题使用了模板函数，故也适用于本题。

3. 设待排序的记录序列用单链表作为储存结构，试写出简单选择排序算法。

```
template <class ForwardIterator>
ForwardIterator min_element(
    ForwardIterator first,
    ForwardIterator last
) {
    if (first==last) return last;
    ForwardIterator smallest = first;
    while (++first!=last)
        if (*first<*smallest)    // or: if (comp(*first,*smallest)) for version (2)
            smallest=first;
    return smallest;
}
template <class InputIterator>
void selectSort(InputIterator first, InputIterator last) {
    for (auto i = first; i != last; ++i)
        swap(*i, *min_element(i, last));
}
```

4. 找出序列中第j小的位置。

```
template <class InputIterator, class T>
InputIterator jth_element(
    InputIterator first,
    InputIterator last,
    int j
) {
    InputIterator i = first + 1, j = last - 1;
    while (i < j) {
        while (i < j && *j >= *first) --j;
        while (i < j && *i < *first) ++i;
        if (i < j)
            swap(i, j);
    }
    while (*i > *first) --i;
    swap(*i, *first);
    if (std::distance(first, i) == k)
        return i;
    else if (std::distance(first, i) > k)
        return jth_element(first, i, k);
    else
        return jth_element(i+1, last, j - std::distance(first, i));
}
```

5. 写出快速排序的非递归调用法

```
template <class InputIterator>
void quickSort(InputIterator first, InputIterator last) {
    stack<pair<InputIterator, InputIterator>> bound;
    bound.push(make_pair(first, last));
    while (bound.size()) {
        InputIterator i = bound.top().first, j = bound.top().last;
        if (j - i < 2) {
            bound.pop();
            continue;
        }
        ++i, --j;
        while(i < j) {
            while (i < j && *i < *bound.top().first) ++i;
            while (i < j && *j > *bound.top().first) --j;
            if (i < j) swap(*i, *j);
        }
        while (i >= first && *i > *bound.top().first) --i;
        swap(*bound.top().first, *i);
        bound.push(make_pair(first, i));
        bound.push(make_pair(i+1, last));
        bound.pop();
    }
}
```

6. 将线性表中的正整数与负整数分开，使得前一半为负、后一半为正。

```
template <class InputIterator, class T>
void split(
    InputIterator first,
    InputIterator last,
    const T& bound = 0
) {
    auto i = first;
    auto j = last - 1;
    while (i < j) {
        while (*i <= bound) ++i;
        while (*j > bound) --j;
        if (i < j)
            swap(*i, *j);
    }
}
```

7. 在堆中插入一元素

```
template <class T>
void appendHeap(vector<T>& heap, const T& val) {
    heap.push_back(val);
    auto i = heap.size() - 1;
    while (i) {
        if (heap[(i-1)/2] > val) {
            swap(heap[i], heap[(i-1)/2]);
            i = (i-1)/2;
        } else break;
    }
}
```

8. 给定有序序列A, B，请归并为有序序列C。

```
template <class T>
vector<T> merge(const vector<T> & A, const vector<T> & B) {
    vector<T> ans;
    auto i = A.begin();
    auto j = B.begin();
    while (i != A.end() || j != B.end())
        ans.push_back((j == B.end() || i != A.end() && *i < *j)?
            *(i++) : *(j++));
    return ans;
}
```