数据结构第一次作业

软件51 沈俞霖 2151601013

2016-9-30

1 复杂度与时间估计

1.1 作业题目

程序A和程序B经过分析发现其最坏情形运行时间分别不大于 $150N\log N$ 和 N^2 。如果可能，请回答下列问题：

A 对于 N 的大值($N > 10000$)，哪一个程序的运行时间有更好的保障？

B 对于 N 的小值($N < 100$)，哪一个程序的运行时间有更好的保障？

C 对于 $N = 1000$ ，哪一个程序平均运行得更快？

D 对于所有可能的输入，程序B是否总比程序A运行得快？

1.2 问题分析

A A程序运行时间有更好的保障。当 $N = 10000$ 时，在最坏情况下，$150N\log N \approx 2 \times 10^7$ ，而 $N^2 = 10^8$，故A程序运行时间有更好的保障。

B B程序运行时间有更好的保障。当 $N = 100$ 时，在最坏情况下，$150N\log N \approx 10^5$ ，而 $N^2 = 10^4$，故B程序运行时间有更好的保障。

C 不能判断，因为题目只给出了最坏情况运行时间，没有给出平均情况运行时间。

D 不能判断，因为题目只给出了最坏情况运行时间，不能以此来判断所有的输入。

2 递归方程求复杂度

2.1 作业题目

考虑以下递归方程，定义函数 $T(n)$ ：

A

$$T(n) = \begin{cases} 1, & \text{如果} n = 1 \\ T(n-1) + n, & \text{其他情况} \end{cases} \tag{1}$$

B

$$T(n) = \begin{cases} 1, & \text{如果} n = 0 \\ 2T(n-1), & \text{其他情况} \end{cases} \tag{2}$$

请给出A和B两种递归式的大O表示，并证明。

2.2 问题分析

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

证明 (1) 令 $n = 1$，

$$T(n) = 1 = \frac{n^2 + n}{2}$$

(2) 若

$$T(k) = \frac{k^2 + k}{2}$$

则 $T(k+1) = T(k) + k + 1$，即

$$T(k+1) = \frac{(k+1)^2 + (k+1)}{2}$$

由(1)(2)可知，

$$T(n) = \frac{n^2 + n}{2}$$

故 $T(n) = O(n^2)$，证毕。

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

证明 (1) 令 $n = 0$，

$$T(n) = 1 = 2^n$$

(2) 若

$$T(k) = 2^k$$

则 $T(k+1)=2T(k)$，即

$$T(k+1)=2^{k+1}$$

由(1)(2)可知，

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

故 $T(n)=O(2^n)$，证毕。

3 测试排序算法

3.1 作业题目

实现直接插入排序、简单选择排序、希尔排序、快速排序和归并排序，以能够对给定数组的正序排序，并按照满足下列情形进行测试：

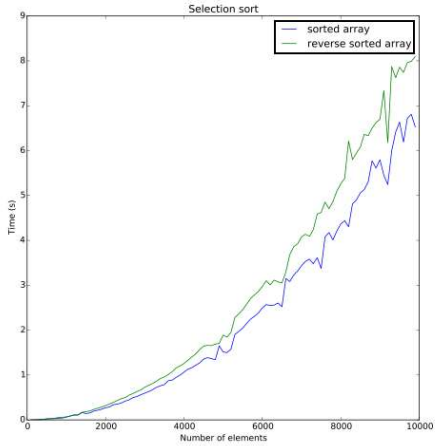
A 测试数组的大小为[100,200,300,⋯,10000] 100 种大小

B 测试数组中的元素分别为正序、逆序和随机序列

对测试的结果需要用图形的方式进行展示：

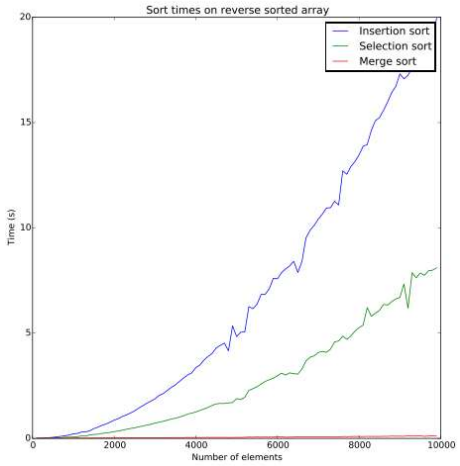
a 展示每个排序算法在满足条件 A 和条件 B 情形下的运行时间趋势变化图，如图 1 所示

图 1: 对选择排序的分析



b 将所有排序算法在正序下、逆序下和随机序列下的运行时间的对比图，如图 2 所示

图 2: 对逆序排序的分析



3.2 程序实现

插入排序

```
// InsertionSort.java 插入排序
public class InsertionSort extends SortFunction{
  InsertionSort(DataGenerator gen, int size){
    super(gen, size);
  }
  void sort(int[] arr){
    for (int cur = 1; cur < arr.length; cur++) {
      // 在当前值之前的元素都已排序
      int t = arr[cur];
      int nxt = cur;
      // 将所有大于当前值的已排序元素右移
      for (; nxt >= 1 && arr[nxt - 1] > t; nxt--) {
        arr[nxt] = arr[nxt - 1];
      }
      // 将当前值归位
```

```
      arr[nxt] = t;
    }
  }
}
```

选择排序

```
// SelectionSort.java 选择排序
public class SelectionSort extends SortFunction{
  SelectionSort(DataGenerator gen, int size){
    super(gen, size);
  }
  void sort(int[] arr){
    for (int i = 0; i < arr.length; i++) {
      // 初始化当前最小元素及其位置
      int curm = arr[i];
      int curmi = i;
      // 遍历数组，寻找最小元素
      for (int j = i + 1; j < arr.length; j++) {
        if(arr[j] < curm){
          curm = arr[j];
          curmi = j;
        }
      }
      // 将当前元素与最小元素交换
      int t = arr[i];
      arr[i] = arr[curmi];
      arr[curmi] = t;
    }
  }
}
```

希尔排序

```java
// ShellSort.java 希尔排序
public class ShellSort extends SortFunction{
  ShellSort(DataGenerator gen, int size){
    super(gen, size);
  }
  void sort(int[] arr){
    // 使用Marcin间隔序列
    int[] gaps = new int[]{701, 301, 132, 57, 23, 10, 4, 1};
    for(int i = 0; i < gaps.length; i++){
      // 对每个间隔进行一次插入排序
      for(int j = gaps[i]; j < arr.length; j++){
        int t = arr[j];
        int k = j;
        for( ; k >= gaps[i] && arr[k - gaps[i]] > t; k -= gaps[i]){
          arr[k] = arr[k - gaps[i]];
        }
        arr[k] = t;
      }
    }
  }
}
```

快速排序

```java
// QuickSort.java 快速排序
import java.util.Random;

public class QuickSort extends SortFunction{
  QuickSort(DataGenerator gen, int size){
    super(gen, size);
  }
  void sort(int[] arr){
    // 调用递归的排序方法
    rsort(arr, 0, arr.length);
```

```
}
// 递归快速排序
void rsort(int[] arr, int l, int r){
  // 在元素个数小于等于16时调用插入排序
  if(r - l <= 16){
    isort(arr, l, r);
    return;
  }
  // 随机选取轴值并与第一个元素交换
  Random ran = new Random();
  int pivot = ran.nextInt(r - l);
  int t = arr[l + pivot];
  arr[l + pivot] = arr[l];
  arr[l] = t;
  int cur = l;
  // 选取所有小于轴值的元素放到数组左半部分
  for (int i = l + 1; i < r; i++) {
    if(arr[i] < arr[l]){
      cur++;
      t = arr[cur];
      arr[cur] = arr[i];
      arr[i] = t;
    }
  }
  // 将轴值放到数组中间
  t = arr[cur];
  arr[cur] = arr[l];
  arr[l] = t;
  // 递归对左右两部分进行排序
  rsort(arr, l, cur);
  rsort(arr, cur + 1, r);
}
// 插入排序
```

```
  void isort(int[] arr, int l, int r){
    for (int cur = l + 1; cur < r; cur++) {
      int t = arr[cur];
      int nxt = cur;
      for (; nxt >= l + 1 && arr[nxt - 1] > t; nxt--) {
        arr[nxt] = arr[nxt - 1];
      }
      arr[nxt] = t;
    }
  }
}
```

归并排序

```
// MergeSort.java 归并排序
public class MergeSort extends SortFunction{
  MergeSort(DataGenerator gen, int size){
    super(gen, size);
  }
  void sort(int[] arr){
    // 调用递归的排序方法
    rsort(arr, 0, arr.length);
  }
  // 递归归并排序
  void rsort(int[] arr, int l, int r){
    // 在元素个数小于等于16时调用插入排序
    if(r - l <= 16){
      isort(arr, l, r);
      return;
    }
    // 把数组均分，递归进行排序
    rsort(arr, l, (l + r) / 2);
    rsort(arr, (l + r) / 2, r);
```

```
    // 申请临时空间
    int[] arr2 = new int[arr.length];
    int arr2p = 0;

    // 归并两个有序数组
    int ls = l, le = (l + r) / 2, es = (l + r) / 2, ee = r;
    while(ls < le && es < ee){
      if(arr[ls] < arr[es]){
        arr2[arr2p++] = arr[ls++];
      }
      else{
        arr2[arr2p++] = arr[es++];
      }
    }
    while(ls < le){
      arr2[arr2p++] = arr[ls++];
    }
    while(es < ee){
      arr2[arr2p++] = arr[es++];
    }

    // 将临时结构拷贝回原数组
    System.arraycopy(arr2, 0, arr, 0, arr.length);
  }
  // 插入排序
  void isort(int[] arr, int l, int r){
    for (int cur = l + 1; cur < r; cur++) {
      int t = arr[cur];
      int nxt = cur;
      for (; nxt >= l + 1 && arr[nxt - 1] > t; nxt--) {
        arr[nxt] = arr[nxt - 1];
      }
      arr[nxt] = t;
```

```
    }
  }
}
```