Exercise 2.3-4

Jack Maney

August 19, 2013

For the practice, let's break down the overall running time of the recursive insertion sort, as found in

com.jackmaney.IntroductionToAlgorithms.sort.RecursiveInsertionSort.java

First, we look at the running time of the *insert* method. Recall that this method assumes that the elements of *list* from index 0 to $index - 1$ are sorted, and inserts the element *index* into the sorted array $list[0..(index - 1)]$.

```
public static <T extends Comparable<T>> void insert(AbstractList<T> list, int index){

        if(index < 0 || index >= list.size()){
                throw new IllegalArgumentException();
        }

        if(index > 0){
                T elementToInsert = list.get(index);

                for(int i = index - 1; i >= 0; i--){

                        if(elementToInsert.compareTo(list.get(i)) <= 0){
                                list.set(i+1,list.get(i));

                                if(i == 0){
                                        list.set(0,elementToInsert);
                                }
                        }
                        else{
                                list.set(i+1,elementToInsert);
                                break;
                        }
                }
        }
}
```

As before, we will denote by c_i a constant running time for line i. Let t denote the number of times that the loop in lines 10–23 is run (note that $1 \leq t \leq index$)

cost	times
c_3	1
c_7	1
c_8	1
c_{10}	$t+1$
c_{12}	t
c_{13}	Either t or $t-1$
c_{15}	Either t or $t-1$
c_{16}	Either t or $t-1$
c_{19}	t
c_{20}	1
c_{21}	1

Therefore, there exist constants a, and b such that the total running time is

$$T(n, index) = at + b$$

and hence the *insert* method has a running time anywhere from $\Theta(1)$ to $\Theta(index)$.

We now look at the *sort* method, which has as arguments not only a list, but an index upon which to use recursion:

```
1  public static <T extends Comparable<T>> void sort(AbstractList<T> list, int index){
2          if(index < 0 || index >= list.size()){
3                  throw new IllegalArgumentException();
4          }
5
6          if(index > 0){
7                  sort(list, index - 1);
8                  insert(list,index);
9          }
10
11 }
```

We will compute the total running time–denoted by $T(n, index)$ for running this method once. We will also denote by t_i the running time for the *insert* method in line 8 ($1 \leq t_i \leq index$).

cost	times
c_2	1
c_6	1
$T(n, index-1)$	1
t_i	1

So, there exist constants a, b, and c such that

$$T(n, index) = aT(n, index - 1) + bt_i + c.$$

For the sake of notational convenience, let us denote *index* by i for the moment. Expanding the recurrence relation above–and using a simple argument by induction–one can show that there exist constants $a_0, a_1, a_2, \cdots, a_{i-1}$ such that

$$T(n, i) = a_0 + a_1 t_1 + a_2 t_2 + \cdots + a_i t_i$$

Finally, we consider the overloaded *sort* method that recursively sorts the entire list:

```
public static <T extends Comparable<T>> void sort(AbstractList<T> list){
        sort(list,list.size() - 1);
}
```

And we can write the running time of this method as

$$T(n) = a_0 + a_1 t_1 + a_2 t_2 + \cdots + a_n t_n$$

where the a_j are constants and t_j are as defined above.

Now, to specifically address the question, the worst case scenario is when the list that we're provided is in reverse sorted order. In that case, $t_j = j$ for all $1 \leq j \leq n$, and we have a running time of $\Theta(n^2)$.