Exercise 2.3-5

Jack Maney

August 19, 2013

Here is the main algorithm for Binary Search (found in *BinarySearch.java*), which includes *begin* and *end* indices (note that there's an overloaded version with values of 0 and $list.size() - 1$ for *begin* and *end*, respectively):

```
public static <T extends Comparable<T>> int search(AbstractList<T> list
                ,T element,int begin,int end){

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

        int result = -1;

        if(begin == end){
                result = list.get(begin).equals(element) ? begin : -1;
        }
        else{
                int middle = (begin + end) / 2;

                int comparison = list.get(middle).compareTo(element);

                if(comparison == 0){
                        result = middle;
                }
                else if(comparison < 0){ //element is on the right (if it exists)
                        result = search(list,element,middle + 1,end);
                }
                else{ //element is on the left (if it exists)

                        /*
                         * We use upperIndex to avoid the edge case of when
                         * our beginning and ending indices are adjacent.
                         * In such a case, middle == begin, so we can't go
                         * from begin to middle - 1 == begin - 1.
                         */
```

```
                        int upperIndex = begin + 1 == end ? begin : middle - 1;

                        result = search(list,element,begin,upperIndex);
                }
        }

        return result;
}
```

Again, since this algorithm is recursive, we'll compute the cost for one iteration and use induction to view the larger picture.

cost	times
c_4	1
c_8	1
c_{10}	1
c_{11}	0 or 1
c_{14}	1
c_{16}	1
c_{18}	1
c_{19}	0 or 1
c_{21}	1
$T\left(\lfloor \frac{n}{2} \rfloor\right)$ if we reach this branch of code	1
c_{24}	1
c_{32}	1
$T\left(\lfloor \frac{n}{2} \rfloor\right)$ if we reach this branch of code	1
c_{38}	1

So, let's consider the worst-case scenario: namely that the element that we seek isn't within our list. We have a total of $\lfloor lg(n) + 1 \rfloor$ iterations of our algorithm (where n is the length of our list) since

- $\lfloor lg(n) \rfloor$ is the smallest integer power of 2 that does not exceed n–and hence the largest number of sublists that we can pick, and
- a list with only one element obviously undergoes only one iteration

Since the only source of non-constant cost in our binary search algorithm above is due to recursion, it follows that the worst case running time is

$$\Theta\left(\lfloor lg(n) + 1 \rfloor\right) = \Theta(lg(n) + 1) = \Theta(lg(n)).$$

Further, as empirical verification, here are two benchmarks (provided by http://disy.github.io/perfidix/) for worst case binary searches (the code is in *BinarySearchBenchmark.java*).

Here are the benchmark results for 100 repetitions of binary search over a list of 1000 integers:

```
|= Benchmark ==================================================================================|
| -                     | unit | sum    | min    | max    | avg    | stddev | conf95        | runs   |
|========================================= TimeMeter ==========================================|
|. BinarySearchBenchmark ......................................................................|
| binarySearchBenchmark | ms   | 17.73 | 00.11 | 01.50 | 00.18 | 00.15  | [00.12-00.23] | 100.00 |
|_ Summary for BinarySearchBenchmark __________________________________________________________|
|                       | ms   | 17.73 | 00.11 | 01.50 | 00.18 | 00.15  | [00.12-00.23] | 100.00 |
|----------------------------------------------------------------------------------------------|
|============================== Summary for the whole benchmark ================================|
|                       | ms   | 17.73 | 00.11 | 01.50 | 00.18 | 00.15  | [00.12-00.23] | 100.00 |
|========================================= Exceptions =========================================|
|==============================================================================================|
```

And here are the benchmarks when increased to a list of 10000 integers (one order of magnitude greater):

```
|= Benchmark ==================================================================================|
| -                     | unit | sum    | min    | max    | avg    | stddev | conf95        | runs   |
|========================================= TimeMeter ==========================================|
|. BinarySearchBenchmark ......................................................................|
| binarySearchBenchmark | ms   | 19.72 | 00.11 | 02.15 | 00.20 | 00.21  | [00.11-00.27] | 100.00 |
|_ Summary for BinarySearchBenchmark __________________________________________________________|
|                       | ms   | 19.72 | 00.11 | 02.15 | 00.20 | 00.21  | [00.11-00.27] | 100.00 |
|----------------------------------------------------------------------------------------------|
|============================== Summary for the whole benchmark ================================|
|                       | ms   | 19.72 | 00.11 | 02.15 | 00.20 | 00.21  | [00.11-00.27] | 100.00 |
|========================================= Exceptions =========================================|
|==============================================================================================|
```