Alexander Kim

0,1. theArray is an array of DirectoryEntry references, sorted by name, with size the number which are currently in use. DirectoryEntry has a getName method. Implement find which returns the index of the entry which has that name, or if it is not there, it returns the index where it should be inserted. Your implementation should run in O(log n) time for n=size.

$\boxed{8.5}$ Graded by Brendan Hendricks

```
int find (String name) {

        int low = 0                int high = size

        while (  low < high                    ) {

                int middle = (low + high) /2 ;

                if (theArray[middle].getName(). compareTo(name) < 0       )
                      low = middle + 1 ;
                else

                    high = middle - 1 ;
        }


        return   high ;
}
```

5
+
4
1
3

2. Starting with an empty stack s, perform s.push(3), s.push(1), s.push(4), s.push(1), s.pop(), s.push(5), s.pop(). What will s.peek() return?

4

3,4. Write code to print out the contents of Stack<String> s from bottom to top and leave s unchanged afterwards. Use System.out.println to print each element.

```
Stack <String> helperStack = (String)(new array[INITIAL_CAP])
while (      s.empty != true) {
   helperStack.push(     s.pop())
}
while (helperStack.empty != true) {
  system.out.println (    s.push(helperStack.pop()))
}
```

1.

5. ArrayStack is implemented using an array E[] theData and an int size, the number of elements in the stack, with the BOTTOM of the stack at theData[0], implement the push method. You can call reallocate() but you do not have to implement it.

```
E push (E item) {
  if (size = theData.length()) {
    reallocate();
  }
  theData[size +1] = item;

  size ++;
  return item;
}
```

```
6. Implement ArrayStack pop.  Don't forget to throw an EmptyStackException.
   You do not have to implement empty().

        E pop () {
```

$-\frac{1}{2}$

```
          if ( theData.empty() ) {
            throw new EmptyStackException e;
          }

          size--;
          return theData[size-1] }
```

```
7. Implement push for linked stack with Node variable top pointing to the top Node
   in the stack.

        class Node {
                E data;
                Node next;
                Node (E data) { this.data = data; this.next = null; }
        }

        E push (E item) {
```

$-\frac{1}{2}$

```
            Node newTopNode = new Node(item);
            top = newTopNode;

          return newTopNode;
          }
```

```
8. ListStack stores the stack in a List<E> theList with the top of the stack as the
   last element of theList.  Implement peek using the size() and get(index) methods
   of List<E>.  You can assume empty() has already been implemented.

        E peek () {
```

$-\frac{1}{2}$

```
          if ( theList.size() != 0 ) {

          return theList.get( theList.size() - 1 );
          }
          return null;
```

```
9. If the numbers stack and operators stack below have the top on the right, show
   them after we pop two numbers and one operator, execute the operator on the
   numbers, and push the result on the numbers stack.  Numbers were pushed in order
   they appear in the expression.
                                             Top
        numbers stack:          0 1 2 3
        operators stack:        + * ^

                0 1 8

                +  *
```