2. Searching Algorithms :: Recursive Binary Search

Remember that recursion is a problem solving process that will often yield a very compact and elegant solution to a problem. Recursion is a viable way to solve a problem if certain problem characteristics are present (see *Recursion : Section 3*):

1. The size of the problem must be reducible to a smaller, basically-equivalent subproblem.
2. The smaller, basically-equivalent subproblem must be simpler to solve than the larger problem.
3. The solution to the original problem requires repetition.
4. There must be a base case at which point the size of the problem cannot be further reduced.
5. The solution to the base case is generally easily-obtainable.
6. The solution to the smaller subproblem must be returned and used in solving the larger problem.
7. The solution to the original problem results from solving all of the subproblems.

2. Searching Algorithms :: Recursive Binary Search

All seven of these problem characteristics are present in the binary search problem:

1. In the iterative (while loop based) binary search algorithm when we determine that key is either in the bottom or top half of the list we have effectively reduced our problem to a smaller (we do not even look in the other half of the list for key), basically-equivalent subproblem (look for key in the half of the list that remains to be searched).
2. It is easier to locate key in a smaller list using binary search because there are fewer elements to search through.
3. The iterative binary search algorithm used a while loop.
4. If we have an empty list we cannot reduce the size of the list any more
5. It is very easy to determine if key is in an empty list (it is not) so we would return -1.
6. The solution to the smaller subproblem will either be -1 (determined when the base case is reached) or it will be the index of key.
7. See 6.

All of this, of course, means that we should be able to write a recursive binary search method.

2. Searching Algorithms :: Recursive Binary Search

The recursive method has four input parameters:

pList	The list to be searched (we are actually searching a sublist)
pKey	The search element
pLow	The index of the first element of the sublist
pHigh	The index of the last element of the sublist

Here is the algorithm:

```
public int recursiveBinarySearch(ArrayList<T> pList, T pKey, int pLow, int pHigh) {
  // Base case. When pLow is greater than pHigh, we essentially have an empty
  // list, and pKey cannot be in an empty list so we return -1 (not found).
  if (pLow > pHigh) {
    return -1;
  }
  int middle = (pLow + pHigh) / 2;
  if (pKey.equals(pList.get(middle))) {
    return middle;
  } else if (pKey < pList.get(middle)) {
    return recursiveBinarySearch(pList, pKey, pLow, middle - 1);
  } else {
    return recursiveBinarySearch(pList, pKey, middle + 1, pHigh);
  }
}
```