R-5.1) Hint When the algorithm finds a match, does it know where?

R-5.1) Solution

    public static int binarySearch(int[] data, int target, int low, int high) {
        if (low > high) {
            return -1;                     // failed search
        } else {
            int mid = (low + high) / 2;
            if (target == data[mid]) {
                return mid;                 // index of found match
            } else if (target < data[mid]) {
                return binarySearch(data, target, low, mid - 1);
            } else {
                return binarySearch(data, target, mid + 1, high);
            }
        }
    }

R-5.2) Hint This is probably the first power algorithm you were taught.

R-5.2) Solution
Chapter 5. Recursion

R-5.3) Hint Be sure to get the integer division right.

R-5.3) Solution

R-5.4) Hint You can model your figure after Figure 5.11.

R-5.4) Solution

R-5.5) Hint You should draw small boxes or use a big paper, as there are a lot of recursive calls.
R-5.6) **Hint** Start with the last term.

**R-5.6) Solution** The general case is $H_n = H_{n-1} + \frac{1}{n}$.

R-5.7) **Hint** Process the string from right to left.

**R-5.7) Solution** Use a single-digit as the base case. For a multiple-digit string, let $s' = sd$ for digit $d$. We have that $value(s') = d + 10 \times value(s)$.

R-5.8) **Hint** You can rely on bitwise operations to interpret $n$ in binary.

**R-5.8) Solution**

```java
public static double power(double x, int n) {
    int k = 0;
    while ((1 << k) <= n)
        k++;

    double answer = 1;
    for (int j=k-1; j >= 0; j--) {
        answer *= answer;
        if ((1 << j) & n) > 0)
            answer *= x;
    }
    return answer;
}
```

R-5.9) **Hint** You can use two recursive methods that look like Binary-Sum.

---

**Creativity**

C-5.10) **Hint** Consider reducing the task of telling if the elements of an array are unique to the problem of determining if the last $n - 1$ elements are all unique and different than the first element.

C-5.11) **Hint** You need subtraction to count down from $m$ or $n$ and addition to do the arithmetic needed to get the right answer.

C-5.11) **Solution** The recursive algorithm, product($n$, $m$), for computing product using only addition and subtraction, is as follows: If $m = 1$ return $n$. Otherwise, return $n$ plus the result of a recursive call to the method product with parameters $n$ and $m - 1$.

C-5.12) **Hint** Start by removing the first element $x$ and computing all the subsets that don’t contain $x$.

C-5.13) **Hint** Output to System.out one character at a time.

C-5.13) **Solution**
void printReverse(String s, int n) {
    if (n >= 0) {
        System.out.print(s.charAt(n))
        printReverse(s, n−1);
    }
}

void printReverse(String s) {
    printReverse(s, s.length()−1);
}

C-5.14) Hint Check the equality of the first and last characters and recur (but be careful to return the correct value for both odd- and even-length strings).

C-5.15) Hint Write your recursive method to first count vowels and consonants.

C-5.16) Hint Consider whether the last element is odd or even and then put it at the appropriate location based on this and recur.

C-5.16) Solution
void organize(int[] data, int low, int high) {
    if (low < high) {
        if (data[high] & 1 == 0) { // even
            int temp = data[high];
            data[high] = data[low];
            data[low] = temp;
            organize(data, low+1, high); // data[low] is known to be even
        } else {
            organize(data, low, high−1); // data[high] is known to be odd
        }
    }
}

void organize(int[] data) {
    organize(data, 0, data.length − 1);
}

C-5.17) Hint Begin by comparing the first and last elements in a range of indices in A.

C-5.17) Solution This problem can effectively be solved using the same technique as Exercise C-5.16.

C-5.18) Hint The beginning and the end of a range of indices in A can be used as arguments to your recursive method.
C-5.18) Solution  The solution makes use of the method FindPair(A, i, j, k) below, which given the sorted subarray A[i..j] determines whether there is any pair of elements that sums to k. First it tests whether A[i] + A[j] < k. Because A is sorted, for any j' ≤ j, we have A[i] + A[j'] < k. Thus, there is no pair involving A[i] that sums to k, and we can eliminate A[i] and recursively check the remaining subarray A[i+1..j]. Similarly, if A[i] + A[j] > k, we can eliminate A[j] and recursively check the subarray A[i..j−1]. Otherwise, A[i] + A[j] = k and we return true. If no such pair is ever found, eventually all but one element is eliminated (i = j), and we return false.

Algorithm FindPair(A, i, j, k):

Input: An integer subarray A[i..j] and integer k
Output: Returns true if there are two elements of A[i..j] that sum to k

if i == j then
  return false
else
  if A[i] + A[j] < k then
    return FindPair(A, i + 1, j, k)
  else
    if A[i] + A[j] > k then
      return FindPair(A, i, j − 1, k)
    else
      return true

C-5.19) Hint Check the last element and then recur on the rest of A.

C-5.20) Solution  The running time is O(n), as it is O(n + n/2 + n/4 + n/8 + ···).

C-5.21) Hint Recur on the first n − 1 positions.

C-5.21) Solution  Let us define a method reverse(L, n), which reverses the first n ≤ L.size() nodes in L, and returns a pointer end to the node just after the nth node in L (end = null if n = L.size()). If L.size() ≤ 1, we are done, so let us assume L has at least 2 nodes. If n = 1, then we return L.first().next(). Otherwise, we recursively call reverse(L, n − 1), and let end denote the returned pointer to the nth node in L. We then set ret to end.next() if n < L.size(), and to null otherwise. We then insert the node pointed to by end at the front of L and we return ret. The total running time is O(n).

C-5.22) Hint View the chain of nodes following the head node as forming themselves another list.
Chapter 5. Recursion

Projects

**P-5.23) Hint** Use recursion in your main solution engine.

**P-5.24) Hint** Consider a small example to see why the binary representation of the counter is relevant.