Data Structures and Algorithms in Python

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Instructor's Solutions Manual

WILEY

Chapter

4 Recursion

Hints and Solutions

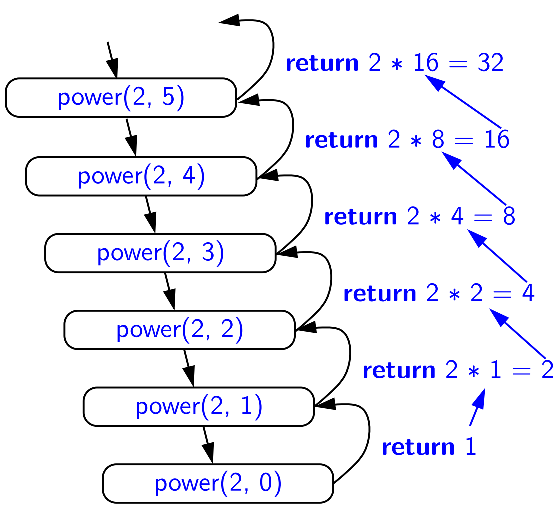
Reinforcement

R-4.1) Hint Don't forget about the space used by the function stack.

R-4.1) Solution If the sequence has 1 element, that is the maximum. Otherwise, consider the bigger of the first element or the maximum of the other $n-1$ elements. The running time and space usages is $O(n)$.

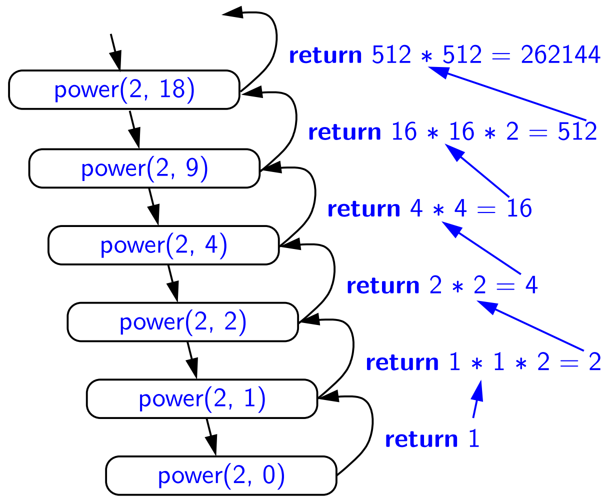
R-4.2) Hint This is probably the first power algorithm you were taught.

R-4.2) Solution



R-4.3) Hint Be sure to get the integer division right.

R-4.3) Solution



R-4.4) Hint You can model your figure after Figure 4.11.

R-4.4) Solution

0	1	2	3	4
4	3	6	2	6
6	3	6	2	4
6	2	6	3	4

R-4.5) Hint You should draw small boxes or use a big paper, as there are a lot of recursive calls.

R-4.6) Hint Start with the last term.

R-4.6) Solution The general case is $H_n = H_{n-1} + \frac{1}{n}$.

R-4.7) Hint Process the string left to right.

R-4.7) Solution Use a single-digit as the base case. For a multiple-digit string, let $s' = sd$ for digit d. We have that $value(s') = d + 10 * value(s)$.

R-4.8) Hint Look for a geometric series.

R-4.8) Solution The running time is $O(n)$, as it is $O(n + n/2 + n/4 + n/8 + \cdots)$.

Creativity

C-4.9) Hint Consider returning a tuple, which contains both the minimum and maximum value.

C-4.10) Hint The integer part of the base-two logarithm of n is the number of times you can divide by two before you get a number less than 2.

C-4.11) Hint Consider reducing the task of telling if the elements of a sequence are unique to the problem of determining if the last $n-1$ elements are all unique and different than the first element.

C-4.12) Hint You need subtraction to count down from m or n and addition to do the arithmetic needed to get the right answer.

C-4.12) 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 function ***product*** with parameters n and $m-1$.

C-4.13) Hint Define a recurrence equation.

C-4.13) Solution let $R(c)$ denote the number of dashes drawn by draw_interval(c). We prove by induction that $R(c) = 2^{c+1} - c - 2$. As a base case, we note that draw_interval(0) does not produce any output, and that $R(0) = 2^{0+1} - 0 - 2 = 0$. For $c > 0$, we note that draw_interval(c) invokes two recursive calls of draw_interval($c-1$), and a call to drawLine that produces c dashes. Therefore, $R(c) = c + 2R(c-1)$, and by the inductive hypothesis, $R(c) = c + 2\left(2^{(c-1)+1} - (c-1) - 2\right) = c + 2\left(2^{c} - c - 1\right) = c + 2^{c+1} - 2c - 2 = 2^{c+1} - c - 2$.

C-4.14) Hint 1

C-4.15) Hint Start by removing the first element x and computing all the subsets that don't contain x.

C-4.16) Hint You can use syntax print(ch, end='') to print one character ch at a time, without extraneous spaces.

C-4.16) Solution

```
def _print_recurse(s, n=None):
  if n == None:
    n = len(s) − 1
  if n >= 0:
    print(s[n], end='')
    _print_recurse(s, n−1)

def print_reverse(s):
  _print_recurse(s, len(s)−1)
  print( )                          # final newline
```

C-4.17) 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-4.18) Hint Write your recursive function to first count vowels and consonants.

C-4.19) Hint Consider whether the last element is odd or even and then put it at the appropriate location based on this and recur.

C-4.19) Solution

```
def _organize_recurse(data, low, high):
  if low < high:
    if data[high] & 1 == 0:                       # data[high] is even
      data[low],data[high] = data[high],data[low]
      _organize_recurse(data, low+1, high)        # data[low] is known to be even
    else:
      _organize_recurse(data, low, high−1)        # data[high] is known to be odd
def organize(data):
  _organize_recurse(data, 0, len(data) − 1)
```

C-4.20) Hint Begin by comparing the first and last elements in a range of indices in A.

C-4.20) Solution This problem can effectively be solved using the same technique as Exercise C-4.19.

C-4.21) Hint The beginning and the end of a range of indices in S can be used as arguments to your recursive function.

C-4.21) Solution The solution makes use of the function 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 $j' \leq j$, 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 $i==j$, no such pair was found 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-4.22) Hint You can rely on bitwise operations to interpret n in binary.

C-4.22) Solution

```
def power(x, n):
  k = 0
  while (1 << k) <= n:
    k += 1

  answer = 1.0
  for j in range(k−1, −1, −1):
    answer *= answer
    if (1 << j) & n > 0:
      answer *= x
  return answer
```

Projects

P-4.23) Hint Review use of the os module.

P-4.24) Hint Use recursion in your main solution engine.

P-4.25) Hint Consider a small example to see why the binary representation of the counter is relevant.

P-4.26) Hint Note the recursive nature of the problem.

P-4.27) Hint Review use of the other methods of the os module.