Test 1 Sample Questions

Robin Dawes

September 30, 2021

Overview

You can expect questions on the following topics:

- complexity
- binary search
- binary trees
- binary search trees

Complexity

Here are some sample questions:

1. Prove that $f(n) = n^3$ is not in $\Omega(n^4)$

Solution:

If n^3 were in $\Omega(n^4)$ there would exist constants $c > 0$ and n_0 such that $n^3 \geq c * n^4 \quad \forall\, n \geq n_0$

$$\Rightarrow \frac{1}{n} \geq c \quad \forall\, n \geq n_0$$
$$\Rightarrow \frac{1}{c} \geq n \quad \forall\, n \geq n_0$$

But $\frac{1}{c}$ is constant so $n > \frac{1}{c} \quad \forall\, n \geq 1 + \left\lceil \frac{1}{c} \right\rceil$, which gives us a contradiction. Therefore the constants c and n_0 do not exist, and therefore $n^3 \notin \Omega(n^4)$

2. Prove that if $f(n)$ is a polynomial function in which all coefficients are positive and k is the largest exponent, then $f(n) \in \Theta(n^k)$

SOLUTION:

Let $f(n) = a_0 + a_1 * n + a_2 * n^2 + \cdots + a_k * n^k$ and let $m = max(a_0, a_1, \ldots a_k)$

$$\begin{aligned} f(n) &\leq m * 1 + m * n + m * n^2 + \cdots + m * n^k \\ &= m * (1 + n + n^2 + \cdots + n^k) \\ &\leq m * (n^k + n^k + \cdots + n^k) \\ &= m * (k+1) * n^k \end{aligned}$$

Since m and k are constants, $m * (k+1)$ is also a constant. Thus $f(n) \in O(n^k)$

Because all the a_i values are > 0, $f(n) > a_k * n^k \quad \forall n \geq 1$. Thus $f(n) \in \Omega(n^k)$

Thus $f(n) \in \Theta(n^k)$

3. Consider this function. Determine its big-O class, its Ω class, and (if possible) its Θ class

```
Complexity(n):
     i = 1
     while (i <= n):
         print(i)
         i = i*2
```

SOLUTION:

Actually, the question should have said "Determine the big-O class (etc.) of this function's **running time**." But people (including me) often just refer to "complexity of the algorithm" when we really mean "complexity of the algorithm's running time". Anyway, it's the running time that we are interested in.

Bye examining the function, we see the interior of the loop takes constant time, and that the loop executes with i = 1, then i = 2, then i = 4, until it reaches i = 2^k where $2^{k+1} > n$. This means that when n doubles, the number of executions of the loop increases by 1. We recognize that this corresponds to $O(\log n)$ complexity. (More formal arguments are possible, but this appeal to established knowledge is acceptable.)

Similarly, analysis of the loop reveals that it always executes $\lceil 1 + \log_2 n \rceil$ times. Thus the function is in the $\Omega(\log n)$ class.

Thus it is in $\Theta(\log n)$

Binary Search

Here is a sample question:

A BITONIC array is one in which all the values are in ascending order until a certain point (called the peak), after which all the values are in descending order. For example

$$A = [3, 7, 8, 12, 7, 2]$$

is a bitonic array in which the peak value is 12.

Write an algorithm that finds the peak value of a bitonic array. Write your algorithm in clear pseudocode.

SOLUTION:

WLOG we can assume that the peak is not in the first position or the last position (we can ensure this by attaching an extremely low value at the beginning and end of A).

```
find_peak(A):   # A is indexed from 1 to n
   first = 1
   last = n
   while True:
      mid = (first+last) // 2 # integer division
              if A[mid-1] < A[mid]   AND   A[mid+1] < A[mid] :
                 # A[mid] is the peak
                 return A[mid]
              elsif A[mid-1] < A[mid]:
                 # peak lies to the right of A[mid]
                 first = mid + 1
              else:
                 # peak lies to the left of A[mid]
                 last = mid - 1
```

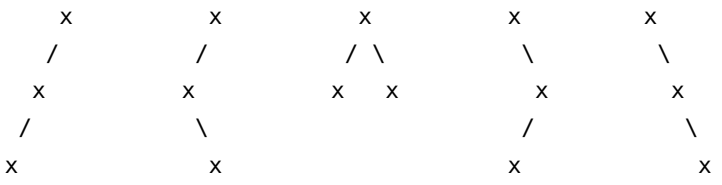
Binary Trees

Here are two sample questions:

1. How many structurally different binary trees are there on 3 vertices? Show them.

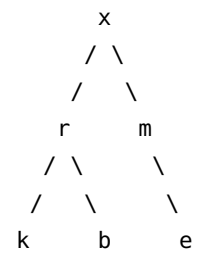
I realize that showing trees is difficult in a typed answer ... that's why this is a practice question rather than a real one!

SOLUTION: There are 5.



2. Write an algorithm that will print the values stored in a binary tree, one level at a time, **from the bottom up**

For example, on this tree



the algorithm could print "k, b, e, r, m, x" ... or any other order in which the "k", "b" and "e" are printed in some order, then the "r" and "m" in some order, and finally the "x"

SOLUTION:

What follows is a simple modification of the Breadth-First Traversal. Instead of printing each vertex's value as it comes off the queue, we push it onto a stack. Once all the vertices are on the stack, we pop them off and print their values - this will print the bottom level of the tree first, then the next-to-bottom level, and so on back up to the root.

```
Bottom_up():         # This function belongs to a tree, so "self.root" means
                     # root of the tree that owns this function
   if self.root == nil:
      return
   else:
    let Q be a queue
    let S be a stack
    Q.append(self.root)          # queues support two operations: append and remove-head
    while Q not empty:
       current = Q.remove_head()
       if current.left_child != nil:
          Q.append(current.left_child)
       if current.right_child != nil:
          Q.append(current.right_child)
       S.push(current)
    # now all vertices are on the stack
    while S.not_empty():
       vertex_to_print = S.pop()
       print(vertex_to_print.value)
```

Binary Search Trees

Here is a sample question:

Write an algorithm for a binary search tree class (with standard definitions) that will take a value x as a parameter, and return the number of times x occurs in the tree.

SOLUTION:

I'm offering a recursive solution, based on the pattern used in class and in the notes: a "starter" function that is called to set up and initiate the recursive process.

```
Count(x):
     return Count_recursive(root,x)

Count_recursive(v, x):
   if v == nil:
      return 0
   elsif v.value == x:
      return 1 + Count_recursive(v.left_child, x)
   elsif v.value > x:
      return Count_recursive(v.left_child, x)
   else:
      return Count_recursive(v.right_child, x)
```

Non-recursive solutions are perfectly acceptable.