12/05/2022

→ ALGORITHM ANALYSIS

ALGORITHM ANALYSIS

Algorithms are clearly very important in CS. You'll probably take an entire course about them within the next year, but we'll do some intro stuff now.

We're very interested in the "behavior" of algorithms. The specific behavior is context-dependent, but common examples include...

- how many times does a certain operation occur?
- how does the number of operations grow relative to the "size" of our input?
- does our algorithm always require the same amount of computation for an input of a given size? If not, what is the best, worst, and average case performance?

Example: Find how many additions/multiplications occur for a given n (consider subtraction as addition).

```
int func(n):
  if n < 3:
    return n+1
  return (n*3) + func(n-2)
```

n:	0	1	2	3	4	5	6	...	n
+:	1	1	1	3	3	5	5	...	$2\lfloor n/2 \rfloor + 1$
×:	0	0	0	1	1	2	2	...	$\lceil n/2 \rceil - 1$

For this disc, give answers that holds as n goes to infinity. Notationally,

$$(\exists N \in \mathbb{N}, \forall n \geq N)[\text{formula holds for } n]$$

PRACTICE

```
boolean is_even(n):
  if n==0:
    return true
  if n==1:
    return false
  return is_even(n-2)
```

How many comparisons for an input n?

n:	0	1	2	3	4	5	6	7	n
=:	1	2	3	4	5	6	7	8	n+1

Our inputs to this function are two sorted arrays of length 10, A & B. Our goal is a singular sorted array of length 20.

```
ans = merge(A, B, 0, 0, [])

arr merge(A, B, a, b, ans):
  if a==10:
    return ans.append(B[b:])
  if b==10:
    return ans.append(A[a:])

  if A[a] < B[b]:
    return merge(A, B, a+1, b, ans.append(A[a]))
  else:
    return merge(A, B, a, b+1, ans,append(B[b]))
```

How many array-comparison operations will this algorithm perform in the best and worst case? What inputs yield this output?

Best Case: 10 comparisons
Occurs when the largest element in A is smaller than the smallest element in B, or vice-versa.

Worst Case: 19 comparisons
Occurs when the largest and second-largest elements are in different arrays.

What is the probability of the worst/best case, assuming all variations of 20 unique elements are equally likely?

Best Case: $\frac{2}{\binom{20}{10}}$ Worst Case: $\frac{2 \cdot \binom{18}{9}}{\binom{20}{10}}$

```
arr mystery(n):
  if n.size == 0:
    return []
  if is_even(n.size):
    return mystery(n[1:]).append(n[0])
  else:
    return mystery(n[1:])
```

What does this algorithm do?

Reverse a list and keep all the elements at even indicies.

How many append-operations for the input [1,2,3,4]?

2

How many append-operations for an input of size n?

$\lfloor \frac{n}{2} \rfloor$

How many subarray-operations for the input [1,2,3]?

3

How many subarray-operations for an input of size n?

n

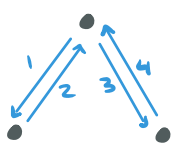
You are interested in traversing a perfect-binary tree of height n in a particular way:

We're interested in the number of edges crossed (go in the psuedo-code):

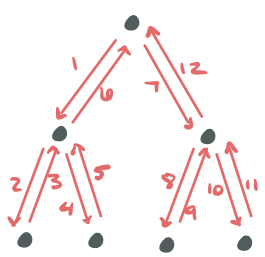
```
traverse(node):
  if is_leaf(node):
    do nothing

  go node.left
  traverse(node.left)
  go node
  go node.right
  traverse(node.right)
  go node
```

So, for $n = 0$: • (nothing happens)

$n = 1$:



Draw the traversal pattern for $n = 2$:



Give a recursive formula for the number of traversals on a tree of height n:

$T(0) = 0$ $\quad\quad$ $T(n) = 4 + 2 \cdot T(n-1)$

What is the closed-form version of this formula?

$T(n) = 4(2^n - 1)$