CSC263

sean.ryan

January 2023

1 Review

Abstract Data Types	**Data Structures**
specification	implementation
objects	data
operations	algorthims

Analysis - Runtime/complexity

- Worst-Case — Upper Bounds - O
- Best-Case — Lower Bounds - Ω
- ??? — Tight Bounds - Θ

When analyzing an algorithm, we are counting by steps. Steps are represented by any constant time operations, such that

$$t_A(x) = Number\, of\, constant\, operations$$

To be able to prove an upper bound, you need to compare two functions: usually, it's $t_A(x)$ compared to runtime. but runtime can have different values depending on the input size, and the order of the input, and that's where worst and best-case scenarios come into play. For a worst-case analysis, you take the largest possible value of runtime for a given input size, and the best case takes the smallest. can we use this fact in proving the tight bound?

Average-case running time
for each n, $s_n = all\, inputs\, of\, size\, n$ if we consider inputs to be random, S_n = a sample space.

we'll want a discrete probability distribution on S_n. For each x$\in$S$_n$, pr(x)

$$t(x) = \; steps\, on\, input\, x$$

Where x is a random variable.

Average-case: T(n) = E(t) = $\Sigma_{x \in S_n} t(x) * Pr(x)$

EX: LinSearch(L, x):

number L is a linked list (precondition).

z =L.head (the first node)

while z != None and z.data != x:

    z = z.next

return z

Average-case runtime? S_n = ? with probability distribution? (aka Pr(x) t(x)?

Here, we need an exact expression for t(x). There is a lot of different ways to count steps, but they would all be within a constant factor of each other. The trick we're going to do here is *choose some key operations s.t counting only these operation is within a constant factor of total time then set t(x) = number of these key operations*

For LinSearch, the key operation is `z.data != x`