Readings Ch. 2, 3; Sections 4.5, 5.1, 5.2

Lecture 02

Worst case running time

For algorithm A, $t(x)$ represents the number of "steps" executed by A on input x.

Worst Case Running Time of A on inputs of size n is denoted $T_A(n)$.

$$T(n) = \max\{t(x) : x \text{ is an input of size } n\}$$

```
LinkedSearch(L, k):
  z = L.head
  while z != None and z.key != k:
    z = z.next
  return z
```

Input size? $\rightarrow$ `L.length` (# of elements in L) $\rightarrow n$
Worst-case runtime?

$$T(n) = an + b \text{ for constants } a, b - T(n) \in \Theta(n)$$

What will we count as steps? $\rightarrow$ comparisons. We have 2: `z != None, z.key != key`.

Average case running time

For algorithm A, consider S_n = sample space of all inputs of size n. To define "average", we require a probability distribution over S_n specifiying the likelihood of each input.

Let $t_n(x)$ = num of steps executed by A on input x in S_n.
note: t_n is a random variable (assigns numerical value to each element of probability space).

"Average" value of $t_n(x)$ is $E[t_n]$ (expected # of steps executed on A on inputs of size n):

$$E[t_n] = \sum_{x \in S_n} t_n(x) \cdot P_r(x) = T'(n) = \text{ average case running time of } A$$

where $P_r(x)$ is probability of x (according to probability distribution).

Problem applying this to `LinkedSearch`? $\rightarrow S_n$ is infinite!
Solution - behaviour of `LinkedSearch` determined by only 1 factor - position of k inside L. Leaves exactly $n+1$ possibilities:

k occurs at position 1 in L,
k occurs at position 2 in L,
$\vdots$
k occurs at position $n-1$ in L,
k occurs at position n in L,
k does not occur in L.

Pick "representative" inputs, e.g.,

$$S_n = \{(L, k) : L = [1, 2, \ldots, n] \text{ and } k = 0, 1, 2, \ldots, n\}$$

Need probability distribution $\rightarrow$ temptation: uniform, all ks equally likely $\left(P(k_i) = \frac{1}{n+1}\right)$.
In reality, impossible to tell, depends entirely on application. In general, could leave some of this as parameter.

Need exact expression for t_n. Pick "representative" operation to count, instead of counting everything. For example, count comparisons: # of comparisions within constant factor of number of operations, so answer is correct in big O terms

$$t_n(L,k) = \begin{cases} 2n+1 & \text{if } k = 0 \\ 2k & \text{if } 1 \le k \le n \text{ (then } k \text{ occurs in position } k) \end{cases}$$

(2 comparisions for every iteration, k iterations for value k, extra comparison when $k = 0$ because of last test for `z != None`)

$$\begin{aligned} T'(n) = E[t_n] &= \sum_{(L,k)\in S_n} t_n(L,k) \cdot P_r[L,k] \\ &= t_n(L,0) \cdot \frac{1}{n+1} + \sum_{k=1}^{n} \left(t_n(L,k) \cdot \frac{1}{n+1} \right) \\ &= \frac{2n+1}{n+1} + \frac{1}{n+1} \cdot \sum_{k=1}^{n} 2k \\ &= \frac{2n+1}{n+1} + \frac{2}{n+1} \cdot \frac{n(n+1)}{2} \\ &= \frac{2n+1}{n+1} + n \end{aligned}$$

Notice: worst-case $T(n) = 2n+1$; average-case $T'(n) \approx n+2$, half of worst case. Consistent with intuition: when elements equally likely to be anywhere in a list, on average examine roughly half the list to find element. Average slightly higher because of case when element not in list.

Best case running time

$$\min\{t(x) : x \text{ is an input of size } n\}$$

For example, $\Theta(1)$ for `LinkedSearch`. Mostly useless.

Upper Bound

The upper bound is usually expressed in Big O notation, $T(n) \in O(g(n))$. Can apply to best-case, worst-case, or average-case. For worst-case we want to prove

$$T(n) = \max\{t(x) : x \text{ of size } n\} \le cg(n)\, \forall n \ge B$$

Usually, exact expression for $t(x)$ or $T(n)$ is unknown. In practice, argue that algorithm executes no more than $cg(n)$ steps on *every* input of size n. In particular, *cannot* prove upper bound by arguing about only one input, unless we also prove this input is worst - back to proving something for every input!

Lower Bound

The lower bound is usually expressed in Ω notation, T_n in $\Omega(f(n))$. Can apply to best-case, worst-case, or average-case. For worst case, we want to prove

$$T(n) = \max\{t(x) : x \text{ has size } n\} \ge cf(n)\, \forall n \ge B$$

Practice: exhibit one input for which algorithm take at least $cf(n)$ steps. (Not every input).