S. KIMMEL

Algorithm Complexity

Worst-case asymptotic, time complexity of an algorithm

Worst-case = worst over all inputs of size n

asymptotic = Big-O

time complexity = run time = # of operations

Linear Search

- Input: $(a_1, a_2, \dots a_n)$, x ⟸ Input size is n
- Output: j if $a_j = x$, 0 otherwise

1) $i = 1$
2) while ($i \le n$ and $x \ne a_i$)
3) $i = i+1$
4) if $i \le n$:
   return i
5) else:
   return 0

This loop repeats at most n times. Each time does a constant # of ops.

worst case

constant # of ops maybe 5, 10, but independent of n!

Last time, to find worst case time complexity

- counted number of operations in worst case
  → Difficult

I suggested last time that we should use big-O notation to characterize worst-case time complexity.

One reason: easier!

Linear search: $T(n) = An + B = O(n)$

(A, B) independent of n

S.KIMMEL

But: shouldn't do just because it is easier. It is also a more useful representation.

Q: **Why?** (Constants don't matter $2x, x = O(x)$, Small x doesn't matter, only $x \geq k$)

- Computers most useful at large input size. At large input size only dominant term matters
- Clock speed / operation can be different from computer to computer; want information that will be useful no matter the computer.
- Tells you about how changing input size changes run time. If double input size, will time double? Quadruple?

$\times 2$

Input size	n	$2n$	
Runtime $O(n^2)$	n^2	$4n^2$	← 4x longer
Runtime $O(2^n)$	2^n	2^{2n}	← How many times longer?

A) $2\times$ B) $4\times$ C) $2^n \times$ C) $4^n \times$

<u>Discussion points</u>

"input size"

$$\text{input: } \overbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & & \\ \vdots & & \ddots & \\ a_{n1} & & & a_{nn} \end{bmatrix}}^{n} \Bigg\} n$$

n or n^2

S.KIMMEL

Q: What is worst-case time complexity of insertion sort? if the input is a list of n items.

A: $O(1)$ B: $O(n)$ C: $O(n^2)$ D: $O(n^3)$

procedure *insertion sort*($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j := 2$ **to** n
  $i := 1$
  while $a_j > a_i$ ← i ops
    $i := i + 1$
  $m := a_j$
  for $k := 0$ **to** $j - i - 1$ ← $j-i$ ops + constant
    $a_{j-k} := a_{j-k-1}$
  $a_i := m$
{$a_1, \ldots, a_n$ is in increasing order}

$j < n$

$\Downarrow$

n reps of n ops $\Rightarrow O(n^2)$

$$\sum_{j=2}^{n} \underset{\text{while loop}}{i} + \underset{\text{for loop}}{j - i} = \sum_{j=2}^{n} j \quad \leftarrow \text{arithmetic series}$$

Formula:

$$\sum_{j=2}^{n} j = 2+3+4+\cdots+(n-2)+(n-1)+n$$

$n-2+1$ total terms

$$= \frac{(n-2+1)}{2}(n+2)$$

$$= \frac{n^2+n-2}{2}$$

$n+2$ $n+2$ $n+2$

S.KIMMEL

ex: $\sum_{j=2}^{6} j = 2+3+4+5+6 = (6+2)\cdot\left(\frac{5}{2}\right) = \frac{8\cdot 4}{2} = 20$

✓

So $\sum_{j=2}^{n} j = \frac{1}{2}n^2 + \frac{1}{2}n - 2 = O(n^2)$

Worst case analysis

$$\sum_{j=2}^{n} j \le \sum_{j=2}^{n} n = \underbrace{n+n+n+n\cdots+n}_{n-2+1 \text{ times}} = n^2-2n+1$$

$$= O(n^2)$$

worst case
$j=n$

S. KIMMEL

Rules of Thumb for Big-O:

Loop 1 to A
 ~~~ ← complexity g
$\Big\}\ O(A\cdot g)$

Loop 1 to A
 ~~~ ← complexity g
Loop 1 to B
 ~~~ ← complexity h
$\Big\}\ O(Ag+Bh)$

Loop 1 to A
 ~~~ ← complexity g
 Loop 1 to B
  ~~~ complexity h
$\Big\}\ O(A(g+B\cdot h))$

A, B, g, h ← use worst-case. OK to round up!
  ~~~

S.KIMMEL

```
for i = 1 to n
    for j = 1 to i
        for k = 1 to j
            Print "Hello"
        for r = 1 to i
            Print "Goodbye!"
```

$$\sum_{i=1}^{n}\left[\sum_{j=1}^{i}(j+i)\right] = \sum_{i=1}^{n}\left[\left(\sum_{j=1}^{i} j\right)+\left(\sum_{j=1}^{i} i\right)\right]$$

$$= \sum_{i=1}^{n}\left((i+1)\cdot\frac{i}{2} + i^2\right) \underset{\substack{\uparrow \\ \text{for } i > \text{constant}}}{\leq} \sum_{i=1}^{n} 3i^2 \leq \sum_{i=1}^{n} 3n^2 = n\cdot 3n^2 = 3n^3$$

Time Complexity: $T(n) = O(n^3)$