Asymptotic anaysis

October 1, 2019 11:01 AM

Notes

- Lab 2 quiz this week - strings, strings func
- print notes
- Homework due Oct 11

Structure variables

- can declare and initialize a local record

```
struct Fight {
    int flightnumber;
    char source[32];
    char destination[32];
}
struct Flight AC = {101,"Vanouver, "Calgary"}
AC.flightnubmer = 101;
strcpy(AC.source, "Vanvouver");
strcpy(AC.destination, "Calgary");
```

Using a pointer to declare and assign values to struct

```
struct Flight* dynamicAC;
dynamicAC = (struct Flight*) malloc(sizeof(struct Flight));
dynamicAC ->flightnumber = 301;
strcpy(dynamicAC->source, "Montreal");
strcpy(dynamicAC->destination, "Toronto");
```

- can also dynamically allocate more than one struct with 1 pointer

```
dynamicAC = (struct Flight*) malloc(3*sizeof(struct Flight));
```

- Local array of pointers to structs, can point to 0, 1 or more dynamically allocated Flight structs

```
struct Flight* dynamicWFJ[10];
dynamicWJ[0] = NULL; // zero flights
dynamicWJ[7] = (struct Flight*) malloc(5*sizeof(struct Flight));
dynamicWJ[8] = (struct Flight*) malloc(1*sizeof(struct Flight));
```

- dynamic 2D array

```
struct Flight** dynamicAA;
dynamicAA = (struct Flight**) malloc(20*sizeof(struct Flight*));
dynamicAA[0] = (struct Flight*) malloc(5*sizeof(struct Flight));
```

Asymptotic Analysis

- Complexity theory studies algorithm efficiency
  - how well algorithm scales as problem in size increases
- Want to compare efficiency
  - Time run, space memory,other (I/O), elegance, energy, etc.
- Number of operations
  - focusing on num of operations performed by algorithm on input of given size
  - ex. num instructions executed, num comparisons
- runtime
- Running time is function of n such as T(n), no longer depends on hardware or subjective conditions
  - ex. dictionary: # of words, Restaurant: # of customers, Airline: # of flights

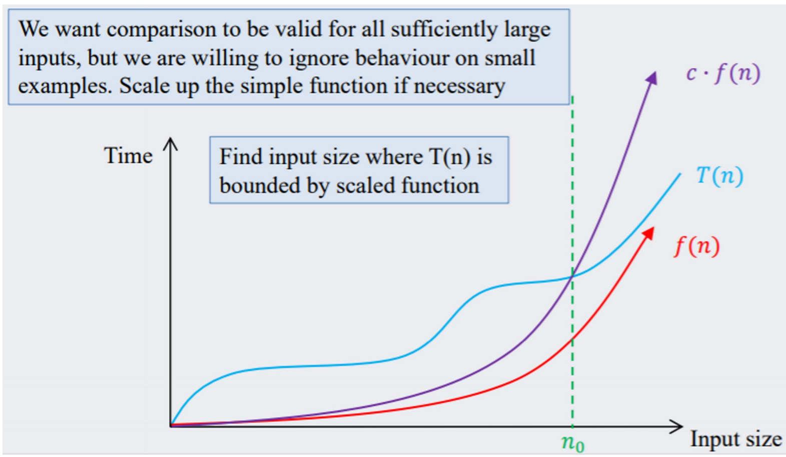
- Comparing algorithms, order notation

  Simple approximate way to make comparisons between behaviours of different algorithms' rates of growth

Order notation

O-notation

$T(n) \in O(f(n))$ if there are constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$



$T(n) \in O(f(n))$ if there are constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$

- $T(n)$ is bounded from above by $c \cdot f(n)$
- i.e. the growth of $T(n)$ is no faster than $f(n)$

$T(n) \in \Omega(f(n))$ if $f(n) \in O(T(n))$

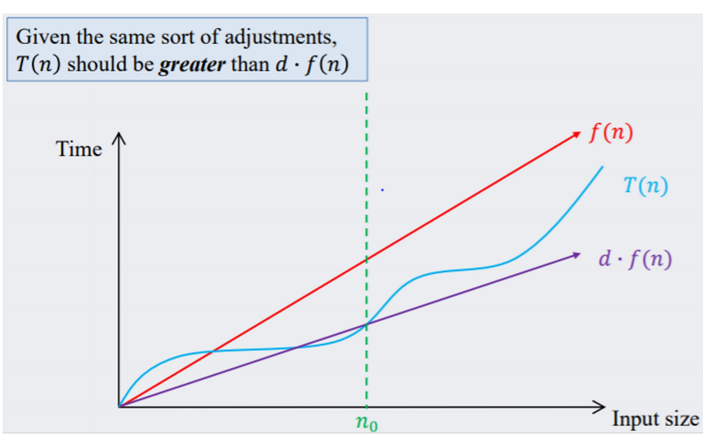
- $T(n)$ is bounded from below by $d \cdot f(n)$
- i.e. $T(n)$ grows no slower than $f(n)$

$T(n) \in \Theta(f(n))$ if $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$

- $T(n)$ is bounded from above and below by $f(n)$
- i.e. $T(n)$ grows at the same rate as $f(n)$

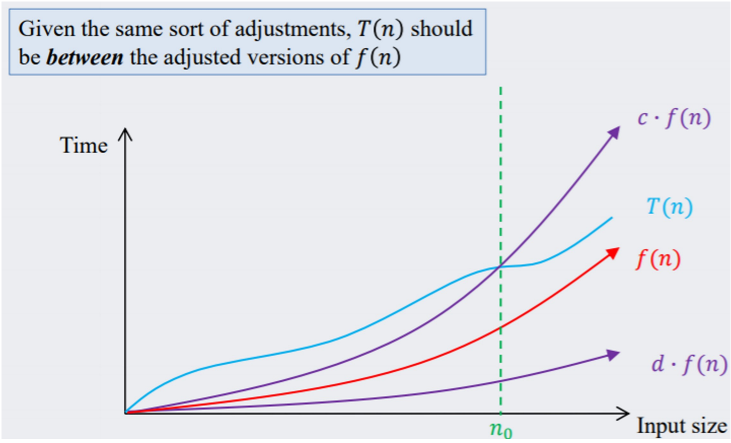
Omega Notation

$T(n) \in \Omega(f(n))$ if $\exists\, d, n_0$ such that $T(n) \geq d \cdot f(n)\ \forall\, n \geq n_0$



Theta- Notation

$T(n) \in \Theta(f(n))$ if $\exists\, c, d, n_0$ such that
$d \cdot f(n) \le T(n) \le c \cdot f(n)\ \forall\, n \ge n_0$



Asymptotic Analysis Hacks

- Running time approximation
- Eliminate low order terms
  - 4n + 5 => 4n
  - 0.5nlogn-2n+7 => 0.5nlogn
  - 2^n + n^3 + 3n => 2^n
- Eliminate constant coeffictients
  - 4n => n
  - 0.5nlogn => nlogn
  - nlog(n^2)=> 2nlogn => nlogn

Common growth rate functions

Typical growth rates in order

– Constant:	$O(1)$	
– Logarithmic:	$O(\log n)$	$(\log_k n, \log(n^2) \in O(\log n))$
– Poly-log:	$O\left((\log n)^k\right)$	
– Linear:	$O(n)$	
– Log-linear:	$O(n \log n)$	
– Superlinear:	$O(n^{1+c})$	(c is a constant, $0 < c < 1$)
– Quadratic:	$O(n^2)$	
– Cubic:	$O(n^3)$	
– Polynomial	$O(n^k)$	(k is a constant) "tractable"
– Exponential	$O(c^n)$	(c is a constant > 0) "intractable"

Dominance

- can look at dominant term to guess at a big-O growth rate

  Up to $n = 100$, the constant term dominates
  Between $n = 100$ and $n = 300$, the linear term dominates
  Beyond $n = 300$, the quadratic term dominates, $T(n) \in O(n^2)$
- which will be faster in the long run? *n^3 vs $n^3 logn$?*
  - split up, use dominance relationships *n^3 vs $n^{3.01}/logn$*

Order notation use

- for f(n) = 3nlog_2 n

$f(n) \in O(n \log n)$ and
$f(n) \in O(2^n)$
are both true

- one is more meaningful
    - "Our function $f\ n$ has growth behaviour no worse than this other pretty well-behaved function"
    - "Our function $f\ n$ has growth behaviour no worse than one of the worst functions known
- want to obtain tightest upper or lower bounding function that still satisfies the O/Om relation

Asymptotic analysis proofs

- Use definitions of O and/or Om to determine either a witness pair (c, n_0) satisfying the definition or show that no such witness pair is possible
    - ex. prove that for *$f(n) = 2log_6 n$* and *$g(n) = 3n$*, *f(n) E O(g(n))*
      prove *$2log_6 n$ E O(3n)*
      *$2log_6 n$ <= c*3n* for *$n >= n_0$*
      let c=1, *$2log_6 n$ <= 3n*
      choose *$n_0 = 6$, $2log_6$ <= 3*6, 2 <= 8*
      Therefore, valid solution is (c=1, n_0 = 6)

> There are constants $c > 0$ and $n_0 > 0$ such that $2\log_6 n \leq c \cdot 3n$ for all $n \geq n_0$
> Choose $c = 1, n_0 = 6$, it can be seen that LHS $\leq$ RHS and remains so as n increases.

-
    - ex. Prove that for f(n) = $2log_6 n$ and g(n) = 3n, g(n) !E O(f(n))
      Assume (contradiction) g(n) E O(f(n))
      There exists consts c>0, n_0>0, f(n) <= c*f(n) for all n >= n_0
      3n <= c*$2log_6 n$, rearrange inequality for c, *c>= $3n/2log_6 n$*
      as n --> inf, increases to inf, no constant that can remain larger than RHS for all increasing vales of n
      Contradiction, initial assumption g(n) E O(f(n)) is false

Input size

- described the number of operations as a function of a given input size *n*
- how are n items organized?

Analysing code

- Boudn flavour
    - Upper bound O
    - Lowerbound Om
    - Asympotically tight T
- Analysis case
    - Best case (lucky)
    - Worst case (adversary)
    - Average case
    - "common" case
- Analysis quality
    - Loose bound
    - Tight bound
- Steps
    - Step 1: what is the input size n?
    - Step 2: what kind of analysis should we perform? Worst? Best? Average?
    - Step 3: How much does each line cost? correct unit?
    - Step 4: What is T(n) in its raw form?
    - Step 5: Simplify T(n) and convert to order notation? Which notation O? Om? T?
    - Step6: Prove asymptotic bound by finding constants c and n_0 satisfying the required

inequalities

- ex. pseudo code:
    - each loop runs n times, const amount of work done inside

```
for i = 1 to n do        1
  for j = 1 to n do      1   } n times   } n times
    sum = sum + 1        1
```

$$T(n)=\sum_{i=1}^{n}\left(1+\sum_{j=1}^{n}2\right)=\sum_{i=1}^{n}(1+2n)=n+2n^2=O(n^2)$$

    - count number of times sum = sum + 1 is executed

$$T(n)=\sum_{i=1}^{n}\sum_{j=1}^{n}1=\sum_{i=1}^{n}n=n^2=O(n^2)$$

- ex.

```
i = 1
while i < n do
  for j = i to n do
    sum = sum + 1
  i++
```

$$T(n)=1+\sum_{i=1}^{n-1}(n-i+2)$$

$$=\frac{n^2}{2}+\frac{3n}{2}-1$$

$$T(n)\in\Theta(n^2)$$

    - $T(n^2)$
    - max range i: $O(n)$ * j: $O(n)$ = $O(n^2)$
    - or = sum of all iterations, 1+2+3... + n-1 + n
    - $\sum^{n}_{k} k = n(n+1)/2 = O(n^2)$

- ex.

```
for (i = 1; i <= n; i++)
  for (j = 1; j <= n; j = j*2)
    sum = sum + 1
```

Visual aid for loop executions

- determine range of loop variable
- determine how many elements within that range will be hit
- Complexities of nested loops usually multiplied
- Complexities of separate loops usually added
- $\log_2 9n$ -> O(logn
- $O(n_2)$
- loop var mltiplied/divided by some const c --> $\log_c$range
- loop var increented/decremented by some const c --> linear in range

```
int i, j;
for (i = 1; i < 9*n; i = i*2) {
  for (j = n*n; j > 0; j--) {
    ...
  }
}
```