Last time

An algorithm is efficient if it does qualitatively better than brute force on every input.

2^n or worse , an^b (polynomial) = efficient

Poll:
Does every computational problem have a polynomial time algorithm?

1. yes
2. ~~no~~

432

$P = NP$

$P \neq NP$

Are we satisfied? No

How to define runtime:

1. level of detail

   → big O notation

2. which inputs?

   best case ~~X~~ → one bad input = unusable

   average case → probability distribution over inputs

   worst case → "for all inputs"

How many primitive operations does the algorithm take?

```
Algorithm 1
Input: integer array A of length n
for i = 1, 2, …, n :
    total = 0
    for j = 1, 2, …, n:
        total = total + A[i]
    B[i] = total
```

assign i 1 prim op

assign total 1 p.o.

5 prim ops n times

inner loop

assign j 1 prim op

retrive total, A[i] 2, + 1, assign total 1

retrive total assign B[i] 2 p.o.

2+5n+2 per outer loop

Assume the following

- *1 operation for basic arithmetic operations*
- *1 operation for variable assignment*
- *1 operation for variable retrieval*

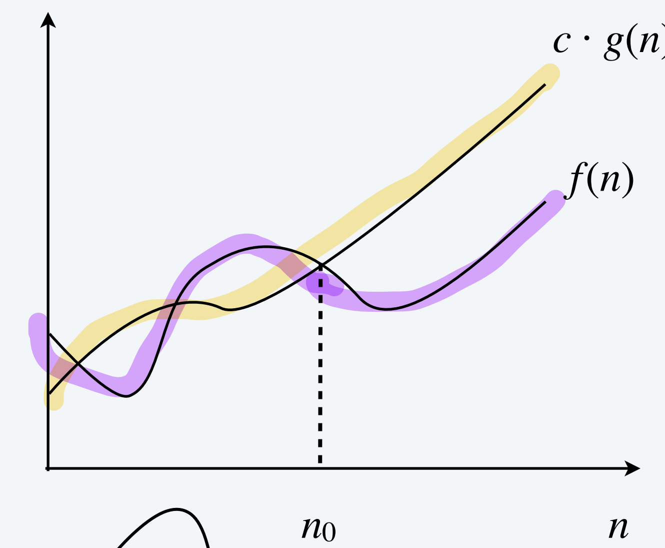
runtime = # primitive ops for input of size n

$T(n) = n(5n+4) = 5n^2 + 4n$

Big O notation

upper bound

Definition of big O: $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



ex $5n^2+4n$ is $O(n^2)$.

We need to show that there exist c and n_0 s.t.

$5n^2+4 \leq cn^2$ for all $n \geq n_0$.

how about $c=10$ and $n_0=0$?

Show that $5n^2+4n \leq 10n^2$ for $n \geq 0$.

Note that for all $n \geq 0$, $n^2 \geq n$.

So $5n^2+4n \leq 5n^2+4n^2 = 9n^2 \leq 10n^2$

could I have chosen $c=4$, $n_0=0$?

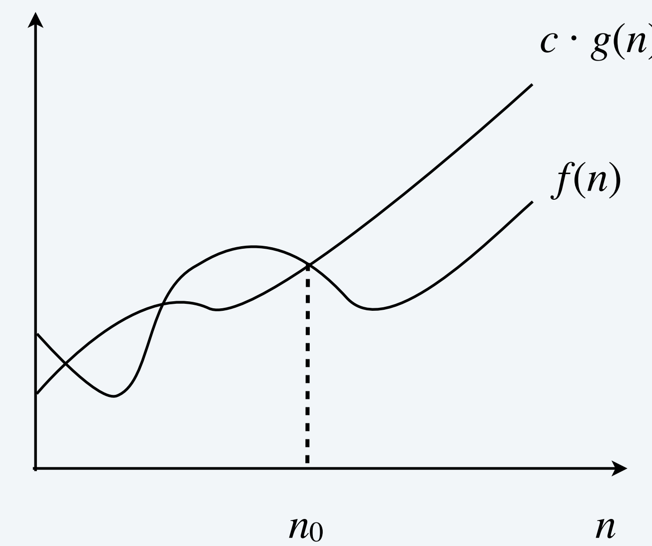
is $5n^2+4n \leq 4n^2$ for all $n \geq 0$?

could I have chosen $c=9$, $n_0=100$?

$5n^2+4n \leq 9n^2$ for all $n \geq 100$.

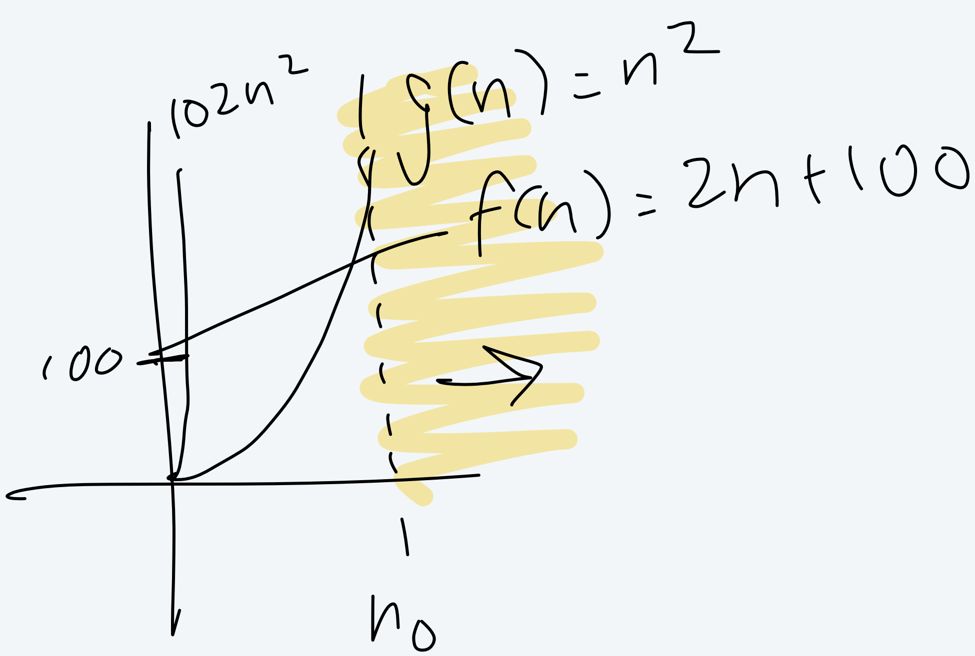
Big O notation: another example

Definition of big O: $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



Claim: $2n + 100$ is $O(n^2)$.

To show $2n+100$ is $O(n^2)$, we have to show that there exist constants $c>0$ and $n_0 \geq 0$ s.t. $2n+100 \leq cn^2$ for all $n \geq n_0$.



try $c = 102$

$n_0 = 1$

need to show:

$2n+100 \leq 102n^2$ for all $n \geq 1$

when $n=1$:

$2+100 \leq 102 \cdot 1^2$ ✓

$c=1$?

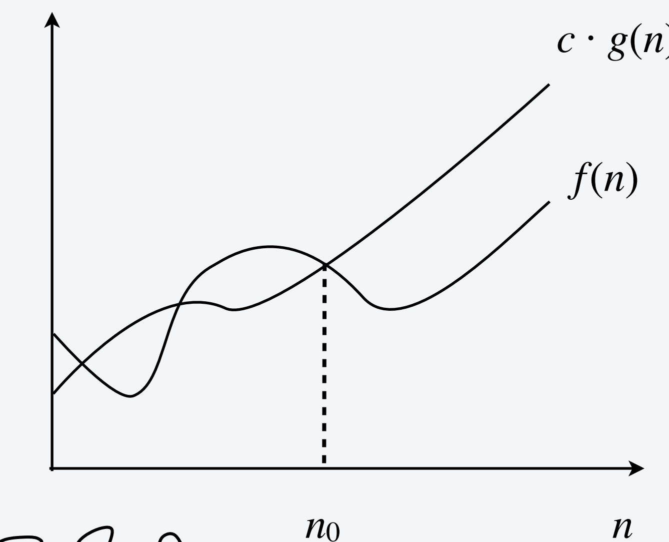
140 400

need $2n+100 \leq n^2$ for all $n \geq 20$

Big O notation: another example

$\leq$

Definition of big O: $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

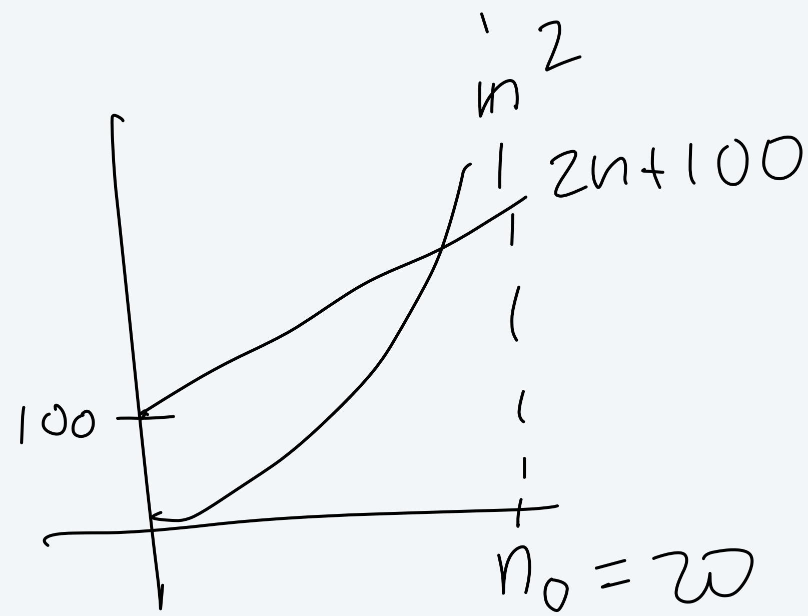


Claim: $2n + 100$ is $O(n^2)$.

Proof:

choose $c = 1$ and $n_0 = 20$. Then, we have $2n + 100 \leq n^2$ for all $n \geq 20$.

see picture.



Claim: 10 is $O(2^n)$.

is $5n+4$ $O(n)$?

how do I prove that

$5n^2+4$ is not $O(n)$?

To prove true: give $c>0$, $n_0 \geq 0$ s.t.

$5n^2+4 \leq cn$ for all $n \geq n_0$.

To prove false ($5n^2+4$ is not $O(n)$):

show that there are no $c>0$, $n_0 \geq 0$

s.t. $5n^2+4 \leq cn$ for all $n \geq n_0$.

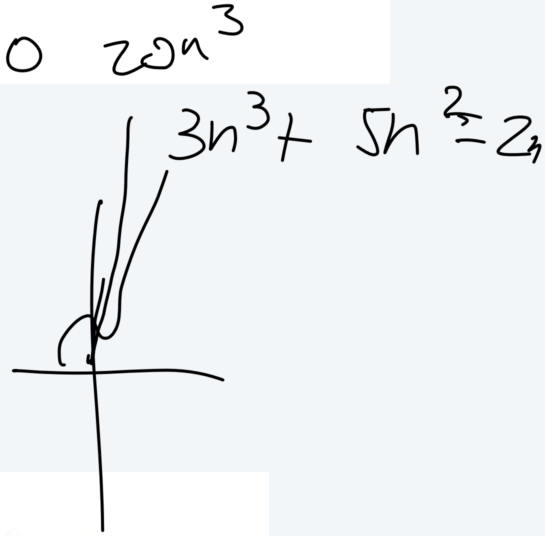
Let's do some examples together

1. Prove that $3n^3 + 5n^2 - 2n = O(n^3)$ using the definition of big O. That is, you must give a c and an n_0 and show that they fit the definition, either by reasoning in words or by drawing a graph.

$3n^3 + 5n^2 - 2n \le cn^3$ for all $n \geq n_0$

$c = 20$, $n_0 = 0$.

$3n^3 + 5n^2 - 2n \le 20n^3$ for all $n \geq 0$.



2. How would you prove that $3n^3 + 5n^2 - 2n \neq O(n^2)$? Don't do it, just tell me how you would.

Def of big O: $f(n)$ is $O(g(n))$ if there exist $c > 0$, $n_0 \geq 0$ s.t. $f(n) \le c\,g(n)$ for all $n \geq n_0$.

1. (8 points) Suppose you have algorithms with the following runtimes. (Assume these are exact running times as a function of the input size n, not a asymptotic running times.) How much slower do these algorithms get when you double the input size?

   (a) n

   (b) $n \log n$

2. (7 points) Recall the definition of big O:

$f(n)$ is $O(g(n))$ if there exist positive constants n_0, c such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Using this definition, prove that $2n + 5$ is $O(n^2)$.

Summary so far

When we talk about the runtime of an algorithm, we always mean:

(1) The # primitive operations on input of size n is f(n) (ex $5n\log n + 6n + 25$)

(2) f(n) is O(g(n)) (ex $5n\log n + 6n + 25$ is $O(n\log n)$)

"the algorithm is n log n"

For $i = 1, 2, \ldots, n$
  For $j = i+1, i+3, \ldots, n$
    Add up array entries $A[i]$ through $A[j]$
    Store the result in $B[i, j]$
  Endfor
Endfor

n-1 time 7 prim op

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

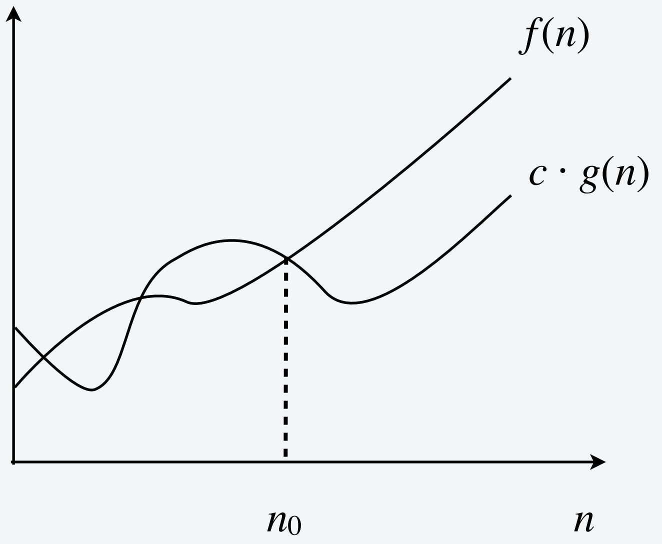
$= \frac{(n-1)(n-2)}{2}$

inner loop: $n-1 + (n-1)(n-2) + n$

overall $f(n) = n(n-1 + (n-1)(n-2) + n)$

Big Omega notation

Lower bounds. $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \ge 0$ such that $f(n) \ge c \cdot g(n)$ for all $n \ge n_0$.

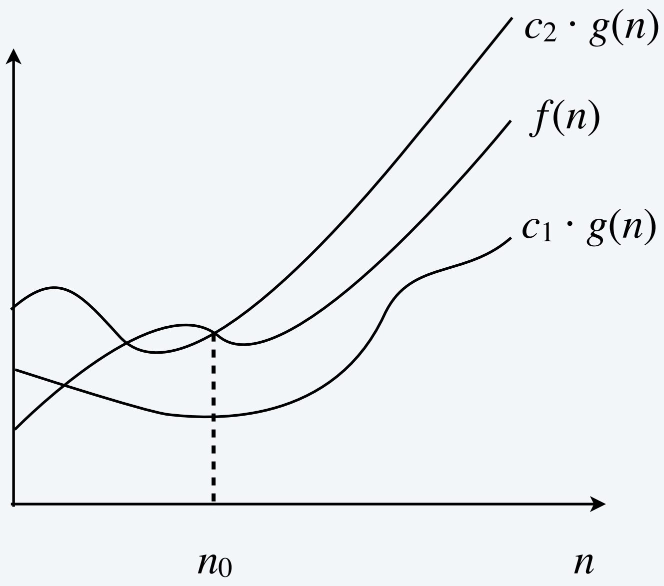


Ex. $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is both $\Omega(n^2)$ and $\Omega(n)$.
- $f(n)$ is not $\Omega(n^3)$.

Big Theta notation

Tight bounds. $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.



Ex. $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $\Theta(n^2)$.
- $f(n)$ is neither $\Theta(n)$ nor $\Theta(n^3)$.

Is $f(n) = 32n^2 + 17n + 5$...

1. $\Theta(n)$
2. $\Theta(n^2)$
3. $\Theta(n^3)$
4. All three
5. $\Theta(n^2)$ and $\Theta(n^3)$

Multiplication by a constant

Suppose I have runtime $f(n)$ and I know it is $O(g(n))$.

Is $b \cdot f(n) = O(g(n))$ for all constants b?

Asymptotically different functions

c

$\log n$

n

$n \log n$

n^2

n^3

...

n^a

2^n

3^n

...

b^n

$n!$

Summary

Big O is ______ for functions

Big Omega is ________ for functions

Big Theta is _________ for functions

We use Big O/Big Omega/Big Theta in order to:

Is there a worst-case input here? (Or best case?)

```
Algorithm 1
Input: integer array A of length n
for i = 1, 2, …, n :
    total = 0
    for j = 1, 2, …, n:
        total = total + A[i]
    B[i] = total
```

Summary

When we talk about the runtime of an algorithm, we always mean: