

37181 DISCRETE MATHEMATICS

©Murray Elder, UTS

Lecture 11: Big O, complexity of algorithms

PLAN

- Big O
- comparing speed of algorithms

COMPARING SPEEDS OF ALGORITHMS

```
power_slow(a real; n positive integer)
x = 1.0
for i = 1 to n do
    x = x*a
return x
```

COMPARING SPEEDS OF ALGORITHMS

```
power_slow(a real; n positive integer)
x = 1.0
for i = 1 to n do
    x = x*a
return x
```

i	x
1	3
2	9
3	27
4	81
5	243
6	729

power_slow(3,6):

[illegible]

COMPARING SPEEDS OF ALGORITHMS

```
power_slow(a real; n positive integer)
x = 1.0
for i = 1 to n do
    x = x*a
return x
```

i	x
1	3
2	9
3	27
4	81
5	243
6	729
7	2187
8	6561
9	19683
10	59049
11	177147
12	531441
13	1594323
14	4782969
15	14348907
16	43046721
17	129140163
18	387420483
19	1162261443
20	3486804489
21	10460413467
22	31381240395
23	94143721183
24	282431163549
25	847293490647
26	2541880471941
27	7625641415823
28	22876924247469
29	68630772742407
30	205892318227221
31	617676954681663
32	1853030864045089
33	5559092592135267
34	16677277776405801
35	50031833329217403
36	150095499987652209
37	450286499962956627
38	1350859499888869881
39	4052578499666609643
40	12157735498999828929
41	36473206496999486787
42	109419619490998460361
43	328258858472995381083
44	984776575418986143249
45	2954329726256958429747
46	8862989178770875289241
47	26588967536312625867723
48	79766892608937877603169
49	239290677826813632809507
50	717872033480440898428521
51	2153616090441322695285563
52	6460848271323968085856689
53	19382544813971904257570067
54	58147634441915712772710201
55	174442903325747138318130603
56	523328709977241414954391809
57	1569986129931724244863175427
58	4709958389795172734589526281
59	14129875169385518203768578843
60	42389625508156554611305736529
61	127168876524469663833917209587
62	381506629573408991501751628761
63	1144519888720226974505254886283
64	3433559666160680923515764658849
65	10290678998482042770547293976547
66	30872036995446128311641881929641
67	92616110986338384934925645788923
68	277848332958015154804776937366769
69	833544998874045464414330812090307
70	2500634996622136393242992436270921
71	7501904989866409179728977308812763
72	22505714969599227539186931926438289
73	67517144908797682617560795779314867
74	202551434726393047852682387337944601
75	607654304179179143558047162013833803
76	1822962912537537430674141486041491409
77	5468888737612612292022424458124474227
78	16406666212837836876067273374373422681
79	49219998638513510628201819123120268043
80	147659995915540531884605457369360804129
81	442979987746621595653816372108082412387
82	1328939963239864786961449116324247237161
83	3986819889719594360884347348972741711483
84	11960459669158783082653042046918225134449
85	35881378907476349247959126140754675403347
86	107644136722429047743877378422263926200041
87	322932409167287143231632135266791778600123
88	968797227501861429694896405800375335800369
89	2906391682505584289084689217401125907401107
90	8719175047516752867254067652203377722203321
91	26157525142550258601762202956609133166609963
92	78472575427650775805286608869827399499829889
93	

[illegible]

COMPARING SPEEDS OF ALGORITHMS

```
power_fast(a real; n positive integer)
x = 1.0
i = n
while i > 0 do
    if i is odd then
        x = x*a
    i = floor(i/2)
    if i > 0 then
        a = a*a
return x
```

COMPARING SPEEDS OF ALGORITHMS

```
power_fast(a real; n positive integer)
x = 1.0
i = n
while i > 0 do
    if i is odd then
        x = x*a
    i = floor(i/2)
    if i > 0 then
        a = a*a
return x

power_fast(3,6):
```

i	x	a

COMPARING SPEEDS OF ALGORITHMS

```
power_fast(a real; n positive integer)
x = 1.0
i = n
while i > 0 do
    if i is odd then
        x = x*a
    i = floor(i/2)
    if i > 0 then
        a = a*a
return x

power_fast(3,100):
```

i	x	a

How can we *formally* capture the idea that the second algorithm is *faster*? Can we estimate that on input of size n , the algorithm will take roughly some function of n steps?

Look back at the two algorithms and try to roughly guess a function for each one.

COMPARING SPEEDS OF ALGORITHMS

Luckily, someone already thought of how to capture the idea of algorithm speed/complexity, using this definition.

Definition

Let $f, g : \mathbb{N}_+ \rightarrow \mathbb{R}$. We say that g *dominates* f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that

$$|f(n)| \leq m|g(n)|$$

for all $n \in \mathbb{N}, n \geq k$.

COMPARING SPEEDS OF ALGORITHMS

Luckily, someone already thought of how to capture the idea of algorithm speed/complexity, using this definition.

Definition

Let $f, g : \mathbb{N}_+ \rightarrow \mathbb{R}$. We say that g *dominates* f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that

$$|f(n)| \leq m|g(n)|$$

for all $n \in \mathbb{N}, n \geq k$.

We use the notation $f \in O(g)$ and read this as “ f is in Big O of g ”.

COMPARING SPEEDS OF ALGORITHMS

Luckily, someone already thought of how to capture the idea of algorithm speed/complexity, using this definition.

Definition

Let $f, g : \mathbb{N}_+ \rightarrow \mathbb{R}$. We say that g *dominates* f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that

$$|f(n)| \leq m|g(n)|$$

for all $n \in \mathbb{N}, n \geq k$.

We use the notation $f \in O(g)$ and read this as “ f is in Big O of g ”.

$O(g)$ is the set of all functions from $\mathbb{N}_+$ to $\mathbb{R}$ which are dominated by g .

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Ex: who dominates who: $f(n) = \log_2 n + 5$ versus $g(n) = n$:

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Ex: who dominates who: $f(n) = \log_2 n + 5$ versus $g(n) = n$:

Here is my prepared solution in case my “live” one is too messy:

Claim 1: $n \leq 2^n$. Let $P(n)$ be the statement $n \leq 2^n$. $P(1)$ true. Assume $P(k)$ then $n + 1 \leq n + n = 2n \leq 2 \cdot 2^n = 2^{n+1}$ so $P(k + 1)$ is implied, so by PMI true for all $n \geq 1$.

Take $\log_2$ both sides gives $\log_2 n \leq n$, so

$$\log_2 n + 5 \leq n + 5 \leq n + n$$

(if $n \geq 5$)

$$= 2n$$

so $m = 2, k = 5$ gives the result.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Some careful analysis of the two algorithms above shows that, up to some constants, `power_fast` takes about $\log_2 n$ steps (because each step divides i by 2, roughly) and `power_slow` takes n steps.

Because of Big O, if you count steps slightly differently, and/or count the initial and final lines of the code, the time complexity function changes a bit but is **the same** up to Big O.

RECALL

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Recall the formula

$$\log_b y = \frac{\log_a y}{\log_a b}.$$

¹

¹Proof: $y = a^{\log_a y}$, so sub this into the LHS.

RECALL

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Recall the formula

$$\log_b y = \frac{\log_a y}{\log_a b}. \quad ^1$$

If $b = 2$ and your calculator only has a button for $\log_{10}$ or $\log_e = \ln$ then you can use this formula:

$$\log_2 n = \frac{\log_e n}{\log_e 2}.$$

¹Proof: $y = a^{\log_a y}$, so sub this into the LHS.

RECALL

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Recall the formula

$$\log_b y = \frac{\log_a y}{\log_a b}.$$
¹

If $b = 2$ and your calculator only has a button for $\log_{10}$ or $\log_e = \ln$ then you can use this formula:

$$\log_2 n = \frac{\log_e n}{\log_e 2}.$$

Since $\log_e 2 \approx 4.6$ is a fixed number, it doesn't matter which log we use because of the m in the definition.

¹Proof: $y = a^{\log_a y}$, so sub this into the LHS.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Let $f(n) = 6n$ and $g(n) = n^2$. Show that g dominates f , that is, $6n \in O(n^2)$.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Let $f(n) = 6n$ and $g(n) = n^2$. Show that g dominates f , that is, $6n \in O(n^2)$.

Here is my prepared solution. It might be different that the one I just did “live” – many choices for m, k can work.

There exist $m = 1, k = 6$ so that

$$6n \leq n \cdot n = n^2$$

for all $n \geq 6$.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $g(n) = n^2$ is not dominated by $f(n) = 6n$. That is, $n^2 \notin O(6n)$.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $g(n) = n^2$ is not dominated by $f(n) = 6n$. That is, $n^2 \notin O(6n)$.

We need the *negation* of the Big O definition.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $g(n) = n^2$ is not dominated by $f(n) = 6n$. That is, $n^2 \notin O(6n)$.

We need the *negation* of the Big O definition.

$$\forall m \in \mathbb{R}_+ \forall k \in \mathbb{N}_+ \exists n \in \mathbb{N}_+ [(n \geq k) \wedge (|f(n)| > m|g(n)|)].$$

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $g(n) = n^2$ is not dominated by $f(n) = 6n$. That is, $n^2 \notin O(6n)$.

We need the *negation* of the Big O definition.

$$\forall m \in \mathbb{R}_+ \forall k \in \mathbb{N}_+ \exists n \in \mathbb{N}_+ [(n \geq k) \wedge (|f(n)| > m|g(n)|)].$$

Here is my prepared solution.

Given m, k fixed numbers (positive), there is a number $n = \max\{6m + 1, k\}$ so that $n > 6m$ so

$$n^2 > m6n.$$

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Let $f(n) = 6n^2 + 5n + 2$ and $g(n) = n^2$. Show that $f \in O(g)$ and $g \in O(f)$. So they are (up to Big O equivalence) the “same”.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $f(n) = n^3$ is dominated e^n .

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $f(n) = n^3$ is dominated e^n .

My prepared solution:

Let $P(n)$ be the statement $n^3 \leq e^n$. Need to find a value k so that $P(k)$ is true, then

assume for $s \geq k$ that $P(s)$ is true, then $P(s+1)$:

$$(n+1)^3 = n^3 + 3n^2 + 3n + 1 \leq n^3 + n^3$$

(by a Lemma I will prove below)

$$= 2n^3 \leq 2e^n < e \cdot e^n = e^{n+1}$$

since $e = 2.7... > 2$.

Lemma: $3n^2 + 3n + 1 \leq n^3$ for n big enough.

CHALLENGE QUESTION

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $f(n) = n^c$ is dominated e^n where c is any positive integer.

So polynomials are “slower” than exponentials.

CHALLENGE QUESTION

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $f(n) = n^c$ is dominated e^n where c is any positive integer.

So polynomials are “slower” than exponentials.

Your proof might use the Binomial Theorem:

$$(x + y)^c = x^c + \binom{c}{1}x^{c-1}y + \binom{c}{2}x^{c-2}y^2 + \cdots + \binom{c}{c-1}x^1y^{c-1} + y^c$$

CHALLENGE QUESTION

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Show that $f(n) = n^c$ is dominated e^n where c is any positive integer.

So polynomials are “slower” than exponentials.

Your proof might use the Binomial Theorem:

$$(x + y)^c = x^c + \binom{c}{1}x^{c-1}y + \binom{c}{2}x^{c-2}y^2 + \cdots + \binom{c}{c-1}x^1y^{c-1} + y^c$$

$$(n + 1)^c = n^c + \binom{c}{1}n^{c-1} + \binom{c}{2}n^{c-2} + \cdots + \binom{c}{c-1}n + 1$$

In your proof, c is a fixed number, so all the $\binom{c}{i}$ terms are bounded above by some fixed number.

Defn: g dominates f if there exist constants $m \in \mathbb{R}^+$ and $k \in \mathbb{N}_+$ such that $|f(n)| \leq m|g(n)|$ for all $n \in \mathbb{N}$, $n \geq k$.

Which is faster (who dominates who?) – exponentials or factorials?

That is, show that one of $f(n) = c^n$, $g(n) = n!$ dominates the other (for any $c \in \mathbb{R}_+$ say).

COMPLEXITY OF ALGORITHMS

If you do more computer science, you will use Big O. You count (roughly, since constant multiples don't matter) the number of steps an algorithm takes on inputs of size $n \in \mathbb{N}$.

The algorithm is “fast” if the number of steps is $O(n)$ or maybe $O(n \log n)$, and “bad” if it takes $O(c^n)$ or worse steps.

Issues: is it bad if it takes this many steps on *all* inputs, or some inputs, or most inputs.

COMPLEXITY OF ALGORITHMS

If you do more computer science, you will use Big O. You count (roughly, since constant multiples don't matter) the number of steps an algorithm takes on inputs of size $n \in \mathbb{N}$.

The algorithm is “fast” if the number of steps is $O(n)$ or maybe $O(n \log n)$, and “bad” if it takes $O(c^n)$ or worse steps.

Issues: is it bad if it takes this many steps on *all* inputs, or some inputs, or most inputs.

See Worksheet 6 for a table of standard functions and their names used in time complexity.

Exercise: prove that the worst case running time for the Euclidean algorithm is when the input is two of the Fibonacci numbers (worksheet 3)

COMPLEXITY OF ALGORITHMS

Exercise: prove that the worst case running time for the Euclidean algorithm is when the input is two of the Fibonacci numbers (worksheet 3)

Standard exercise in comp sci: compare running times for algorithms which sort a list of numbers: bubble sort versus merge sort.

Recall from the Canvas “Welcome” page:

<https://www.youtube.com/watch?v=kVgy1GSDHG8>

And here is another video that mentions Big O:

<https://www.youtube.com/watch?v=xFFs9Ug0ALE>

Next time:

- pigeonhole principle