Let $f(n) = 3n^2 + 17n \log_2 n + 1000$. Which of the following are true?

A. $f(n)$ is $O(n^2)$.

B. $f(n)$ is $O(n^3)$.

C. Both A and B.

D. Neither A nor B.

Analysis of algorithms: quiz 1

Choose $c = 1020$, $n_0 = 1$.

Big-O: Motivation

What is the running time of this algorithm? How many “primitive steps” are executed for an input of size $n$?

```plaintext
sum = 0
for $i = 1$ to $n$
    for $j = 1$ to $n$
end for
end for
```

The running time is $T(n) = \cdot n^2 + \cdot n + \cdot$. What are the coefficients?

For large values of $n$, $T(n)$ is less than some multiple of $n^2$. We say $T(n)$ is $O(n^2)$ and typically don’t care about other terms.

Analysis of algorithms: quiz 1

Let $f(n) = 3n^2 + 17n \log_2 n + 1000$. Which of the following are true?

A. $f(n)$ is $O(n)$.

B. $f(n)$ is $O(n^3)$.

C. Both A and B.

D. Neither A nor B.

Big-O: Formal Definition

Definition: The function $T(n)$ is $O(f(n))$ (read: “is order $f(n)$”) if there exist constants $c \geq 0$ and $n_0 \geq 0$ such that

$$T(n) \leq cf(n) \text{ for all } n \geq n_0$$

We say that $f$ is an asymptotic upper bound for $T$.

Examples:

- If $T(n) = n^2 + 1000000n$ then $T(n)$ is $O(n^2)$
- If $T(n) = n^3 + n \log n$ then $T(n)$ is $O(n^3)$
- If $T(n) = 2\sqrt{\log n}$ then $T(n)$ is $O(n)$

Big-O: What it Is and Isn’t

Is: a way to categorize growth rate of (non-negative) functions relative to other functions.

Is not: “the running time of my function” (just an upper bound for growth rate, may not be tight)

Correct usage:

- The running time of my algorithm in input of size $n$ is $T(n)$.
- Statement about algorithm only.
- $T(n)$ is $O(n^3)$. Statement about the function $T(n)$ only.
- The running time of my algorithm is $O(n^3)$.
- Statement about algorithm and $T(n)$.

Incorrect usage:

- $O(n^3)$ is the running time of my algorithm (think of $O(n^3)$ as a set. Or say in words: “order of $n^3$”)
Properties of Big-O Notation

Claim (Transitivity): If $f$ is $O(g)$ and $g$ is $O(h)$, then $f$ is $O(h)$.

Proof: we know from the definition that

$\bullet$ $f(n) \leq cg(n)$ for all $n \geq n_0$
$\bullet$ $g(n) \leq c'g(n)$ for all $n \geq n'_0$

Therefore

$f(n) \leq cg(n)$ if $n \geq n_0$
$\leq c'g(n)$ if $n \geq n_0$ and $n \geq n'_0$
$= \frac{c'}{c}h(n)$ if $n \geq \max\{n_0, n'_0\}$

Know how to do proofs using Big-O definition.

Consequences of Additivity

$\bullet$ OK to drop lower order terms. E.g., if

$f(n) = 4.1n^3 + 23n + n \log n$

then $f(n)$ is $O(n^3)$

$\bullet$ Polynomials: Only highest degree term matters. E.g., if

$f(n) = a_0 + a_1n + a_2n^2 + \ldots + a_dn^d, \quad a_d > 0$

then $f(n)$ is $O(n^d)$

Other Useful Facts: Log vs. Poly vs. Exp

Fact: $\log_b(n)$ is $O(n^d)$ for all $b$ and $d$

All polynomials grow faster than logarithm of any base

Fact: $n^d$ is $O(r^n)$ when $r > 1$

Exponential functions grow faster than polynomials

Exercise: Prove these facts!

Logarithm review

Definition: $\log_b(a)$ is the unique number $c$ such that $b^c = a$

Informally: the number of times you can divide $a$ into $b$ parts until each part has size one

Properties:

$\bullet$ Log of product $\rightarrow$ sum of logs
$\bullet$ Log of power $\rightarrow$ power

$\bullet$ $\log_b(b^n) = n$
$\bullet$ $\log_b(b) = 1$

When using big-O, it’s OK not to specify base.
Assume $\log_2$ if not specified.

Big-O comparison

Which grows faster?

$n(\log n)^3$ vs. $n^{4/3}$

simplifies to

$(\log n)^3$ vs. $n^{1/3}$

simplifies to

$\log n$ vs. $n^{1/9}$

$\Rightarrow \log n$ is $O(n^{1/9})$
$\Rightarrow n(\log n)^3$ is $O(n^{4/3})$

Apply transformations (monotone, invertible) to both functions.
Try taking log.
Exponential time

An algorithm is exponential time if it is $O(2^{nk})$ for some $k > 0$

Useful fact: (Stirling’s approximation)
\[ n! \sim \sqrt{2\pi n} \left( \frac{n}{e} \right)^n \quad \text{(ratio tends to 1)} \]

Exercise: What can you claim from here for big-O (and later big-Θ)?

---

**Analysis of algorithms: quiz 4**

Which is an equivalent definition of exponential time?

A. $O(2^n)$

B. $O(2^{cn})$ for some constant $c > 0$.

C. Both A and B.

D. Neither A nor B.

---

Big-Ω Motivation

Algorithm `foo`  
for $i = 1$ to $n$  
for $j = 1$ to $n$  
do something...  
end for  
end for  
Fact: run time is $O(n^3)$

Algorithm `bar`  
for $i = 1$ to $n$  
for $j = 1$ to $n$  
do something else..  
end for  
end for  
Fact: run time is $O(n^3)$

Conclusion: `foo` and `bar` have the same asymptotic running time. What is wrong?

---

More Big-Ω Motivation

Algorithm `sum-product`  
sum = 0  
for $i = 1$ to $n$  
for $j = 1$ to $n$  
sum += $A[i]*A[j]$  
end for  
end for  
Fact: run time is $O(n^3)$

What is the running time of `sum-product`?

Easy to see it is $O(n^2)$. Could it be better? $O(n)$?

---

Big-Ω

Informally: $T$ grows at least as fast as $f$

**Definition**: The function $T(n)$ is $Ω(f(n))$ if there exist constants $c \geq 0$ and $n_0 \geq 0$ such that
\[ T(n) \geq c f(n) \quad \text{for all} \quad n \geq n_0 \]

$f$ is an asymptotic lower bound for $T$
Which is an equivalent definition of big Theta notation?

A. $f(n)$ is $\Theta(g(n))$ iff $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$.

B. $f(n) = \frac{2n^2}{2^pM}$ $\sum_{n} a[n]$ $\Omega(n)$ $f(n)$ is $O(n)$ but limit does not exist

counterexample

Exercise: solution

Hard way
- Count exactly how many times the loop executes
  
  \[
  1 + 2 + \ldots + n = \frac{n(n+1)}{2} = \Omega(n^2)
  \]

Easy way
- Ignore all loop executions where $i > n/2$ or $j < n/2$
- The inner statement executes at least $(n/2)^2 = \Omega(n^2)$ times

Analysis of algorithms: quiz 3

Which is an equivalent definition of big Theta notation?

A. $f(n)$ is $\Theta(g(n))$ iff $f(n)$ is both $\Omega(g(n))$ and $\Theta(g(n))$.

B. $f(n)$ is $\Theta(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c < \infty$.

C. Both A and B.

D. Neither A nor B.

Big-Θ example

How do we correctly compare the running time of these algorithms?

Algorithm foo

```
for i= 1 to n do
    for j= i to n do
        do something...
    end for
end for
```

Algorithm bar

```
for i= 1 to n do
    for j= 1 to n do
        for k= 1 to n do
            do something else..
        end for
    end for
end for
```

Answer: foo is $\Theta(n^2)$ and bar is $\Theta(n^3)$.

They do not have the same asymptotic running time.
Additivity Revisited

Suppose \( f \) and \( g \) are two (non-negative) functions and \( f \) is \( O(g) \)

Old version: Then \( f + g \) is \( O(g) \)

New version: Then \( f + g \) is \( \Theta(g) \)

Example:
\[
\frac{n^2 + 42n + n \log n}{g} \text{ is } \Theta(n^2)
\]

Running Time Analysis

Mathematical analysis of worst-case running time of an algorithm as function of input size. Why these choices?

- Mathematical: describes the algorithm. Avoids hard-to-control experimental factors (CPU, programming language, quality of implementation), while still being predictive.

- Worst-case: just works. (“average case” appealing, but hard to analyze)

- Function of input size: allows predictions. What will happen on a new input?

Efficiency

When is an algorithm efficient?

Stable Matching: \( \Omega(n!) \)
Propose-and-Reject?: \( O(n^2) \)

We must have done something clever

Question: Is it \( \Omega(n^2) \)?

Polynomial Time

Definition: an algorithm runs in polynomial time if its running time is \( O(n^d) \) for some constant \( d \)

Polynomial Time: Examples

These are polynomial time:
\[
\begin{align*}
 f_1(n) &= n \\
 f_2(n) &= 4n + 100 \\
 f_3(n) &= n \log(n) + 2n + 20 \\
 f_4(n) &= 0.01n^2 \\
 f_5(n) &= n^2 \\
 f_6(n) &= 20n^2 + 2n + 3 
\end{align*}
\]

Not polynomial time:
\[
\begin{align*}
 f_7(n) &= 2^n \\
 f_8(n) &= 3^n \\
 f_9(n) &= n!
\end{align*}
\]

Why Polynomial Time?

Why is this a good definition of efficiency?

- Matches practice: almost all practically efficient algorithms have this property.

- Usually distinguishes a clever algorithm from a “brute force” approach.

- Refutable: gives us a way of saying an algorithm is not efficient, or that no efficient algorithm exists.