Meeting 12: Asymptotic analysis 2
Brandon Myers
University of Iowa
Today’s big ideas

• big-Oh $O(g(n))$, big-omega $\Omega(g(n))$, and big-theta $\Theta(g(n))$ are used to describe the growth rate of functions

• Try using this notation to describe the running time of some real algorithms!
Never give up!

https://cs.uiowa.edu/resources/approved-tutors

Several tutors on the list for 2230 have already confirmed with me that they are indeed accepting students this semester.
Oh, Omega, Theta

• \( R(n) \in O(g(n)) \) means \( g(n) \) grows same or faster than \( R(n) \) up to constant factors
\( \exists k > 0 \cdot R(n) \leq k \cdot g(n) \) for all \( n \geq n_0 \)

what is \( n_0 \)? Just a sufficiently large number
Oh, Omega, Theta

- $R(n) \in O(g(n))$ means $g(n)$ grows same or faster than $R(n)$ up to constant factors
  \[ \exists k > 0 \cdot R(n) \leq k \cdot g(n) \text{ for all } n \geq n_0 \]

- $R(n) \in \Omega(g(n))$ means $g(n)$ grows same or slower than $R(n)$ up to constant factors
  \[ \exists k > 0 \cdot R(n) \geq k \cdot g(n) \text{ for all } n \geq n_0 \]

what is $n_0$? Just a sufficiently large number
Oh, Omega, Theta

• \( R(n) \in O(g(n)) \) means \( g(n) \) grows same or faster than \( R(n) \) up to constant factors
  \( \exists k > 0 . R(n) \leq k \times g(n) \) for all \( n \geq n_0 \)

• \( R(n) \in \Omega(g(n)) \) means \( g(n) \) grows same or slower than \( R(n) \) up to constant factors
  \( \exists k > 0 . R(n) \geq k \times g(n) \) for all \( n \geq n_0 \)

• \( R(n) \in \Theta(g(n)) \) means \( g(n) \) grows at the same rate as \( R(n) \) up to constant factors
  \( \exists k_0, k_1 > 0 . k_0 \times g(n) \leq R(n) \leq k_1 \times g(n) \) for all \( n \geq n_0 \)

what is \( n_0 \)? Just a sufficiently large number
Peer instruction

\[ f(n) = n^3 \log n + 10 \log n + 20n \]
\[ f(n) \in \Omega(g(n)) \]

Pick a the best correct \( g(n) \)

a) \( n^4 \)
b) \( n^3 \log n + n^4 \)
c) \( n^4 \log n \)
d) \( 10n^3 \)
e) \( n \)

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
Searching an array

What is $R(N)$ for searching for an integer in an unsorted array of integers?
Searching an array

What is $R(N)$ for searching for an integer in an unsorted array of integers?

$$R(N) \in O(N)$$

What if the array is sorted? Can we do better than $O(N)$?
Binary search for key 22
Binary search for key 22
Binary search for key 22
Binary search for key 22
R(N) for Binary Search?

time to compare two integers + time for binary search on half the array

let c=time to compare two integers

R(N) = c + R(N/2)

how many times can you halve N until you get 1?
$$[\log_2 N]$$

so...
R(N) =c * [log_2 N]

$$R(N) \in O(\log_2 N)$$
Nested loops

Algorithm to find if an integer appears at least twice in an array.

```
int N = a.length;
for (int i=0; i < N; i++) {
    for (int k=i+1; k < N; k++) {
        if (a[i]==a[k]) return true;
    }
}
return false;
```

when i=0, the inner loop runs N-1 times
when i=1, the inner loop runs N-2 times
...
when i=N-2 the inner loop runs 1 time

N-1 + N-2 + ... + 2 + 1 = N(N-1)/2

\[ R(N) \in O(N^2) \]
Binary recursion to compute Fibonacci numbers

```java
public int fibBad(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fibBad(n - 1) + fibBad(n - 2);
}
```

time to compute fibBad(n-1) + time to compute fibBad(n-2) + time for everything other than the recursive call (is independent of n)

\[ R(N) = R(N-1) + R(N-2) + c \]

R(N) will be exponential in N!
Linear recursion to computer Fibonacci numbers

```java
public static long[] fibGood(int n) {
    // Computes fib(n), fib(n-1) together
    if (n <= 1) {
        long[] answer = {n, 0};
        return answer;
    } else {
        long[] tmp = fibGood(n-1);
        long[] answer = {tmp[0] + tmp[1],
                         tmp[0]};
        return answer;
    }
}
```

time to compute FibGood(n-1) + time to do everything other than the recursive call (is independent of n)

\[
R(N) = R(N-1) + c
\]

\[
R(N) \in O(N)
\]
Today’s big ideas

• big-Oh $O(g(n))$, big-omega $\Omega(g(n))$, and big-theta $\Theta(g(n))$ are used to describe the growth rate of functions

• Try using this notation to describe the running time of some real algorithms!
Acknowledgements

• these slides borrow ideas from
  • http://homepage.divms.uiowa.edu/~ghosh/2116.html
  • http://datastructur.es/sp16/