CS 344: Design & Analysis of Algorithms

Lecture 2

Sep 5, 2019

Asymptotics

- $\Omega(\cdot)$: "greater-than"
- $\Theta(\cdot)$: "equal"
- $O(\cdot)$: "less-than"

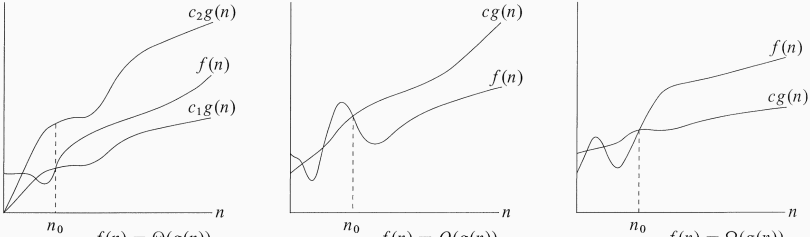
Asymptotics

$f(n) = O(g(n))$ means $\exists c, \exists n_0, \forall n > n_0, 0 \leq f(n) \leq cg(n)$

$f(n) = \Omega(g(n))$ means $g(n) = O(f(n))$

$f(n) = \Theta(g(n))$ means $f(n) = O(g(n))$
and $g(n) = O(f(n))$

Asymptotics



$f(n) = \Theta(g(n))$ $\quad$ $f(n) = O(g(n))$ $\quad$ $f(n) = \Omega(g(n))$

Asymptotics

- $7n^2 + 3 = O(n^2)$
- $7n^2 + 3 = O(n^3)$
- $7n^2 + 3 = \Omega(n^2)$
- $7n^2 + 3 = \Omega(n)$

Asymptotics

- $\omega(\cdot)$: "greater-than"
- $\Omega(\cdot)$: "greater-than or equal to"
- $\Theta(\cdot)$: "equal"
- $O(\cdot)$: "less-than or equal to"
- $o(\cdot)$: "less-than"

Asymptotics

$$f(n) = O(g(n)) \text{ means } \exists c, \exists n_0, \forall n > n_0, f(n) \leq cg(n)$$

$$f(n) = o(g(n)) \text{ means } \forall c, \exists n_0, \forall n > n_0, f(n) < cg(n)$$

Asymptotics

$f(n) = o(g(n))$ can also be interpreted as:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Asymptotics

$$f(n) = \Omega(g(n)) \text{ means } g(n) = O(f(n))$$

$$f(n) = \omega(g(n)) \text{ means } g(n) = o(f(n))$$

Asymptotics

$f(n) = \omega(g(n))$ can also be interpreted as:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

Asymptotics

- $7n^2 + 3 = O(n^2)$
- $7n^2 + 3 = O(n^3)$
- $7n^2 + 3 = o(n^3)$
- $7n^2 + 3 \neq o(n^2)$
- $7n^2 + 3 = \Omega(n^2)$
- $7n^2 + 3 = \Omega(n)$
- $7n^2 + 3 = \omega(n)$
- $7n^2 + 3 \neq \omega(n^2)$

Properties of asymptotics

These relations are transitive:

$$
\begin{aligned}
f(n) = \omega(g(n)) \text{ and } \quad & g(n) = \omega(h(n)) \Rightarrow \quad f(n) = \omega(h(n)) \\
f(n) = \Omega(g(n)) \text{ and } \quad & g(n) = \Omega(h(n)) \Rightarrow \quad f(n) = \Omega(h(n)) \\
f(n) = \Theta(g(n)) \text{ and } \quad & g(n) = \Theta(h(n)) \Rightarrow \quad f(n) = \Theta(h(n)) \\
f(n) = O(g(n)) \text{ and } \quad & g(n) = O(h(n)) \Rightarrow \quad f(n) = O(h(n)) \\
f(n) = o(g(n)) \text{ and } \quad & g(n) = o(h(n)) \Rightarrow \quad f(n) = o(h(n))
\end{aligned}
$$

Properties of asymptotics

Some are reflexive:

$$f(n) = \Omega(f(n))$$
$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$

Θ is symmetric:

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

Properties of asymptotics

And by definition, we can transpose some:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

Asymptotics as an ordering

But the "less-than", "equal", "greater-than" analogy doesn't perfectly hold.

For any two real numbers a and b, one of these must be true:

- $a < b$
- $a = b$
- $a > b$

Asymptotics as an ordering

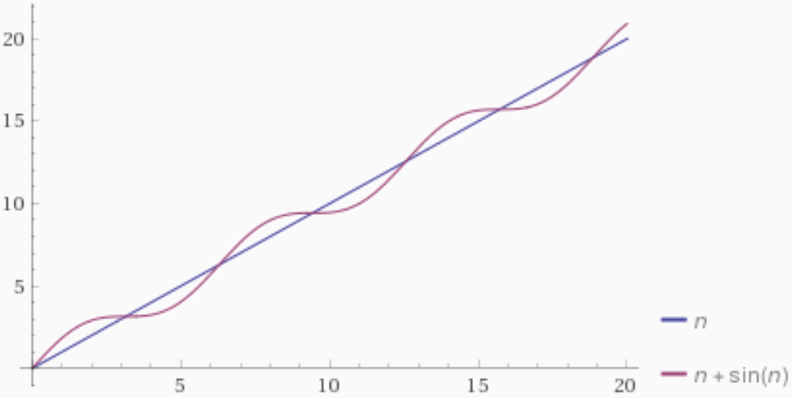
But the “less-than”, “equal”, “greater-than” analogy doesn’t perfectly hold.

For any two functions $f(n)$ and $g(n)$, is it always true that one of these holds?

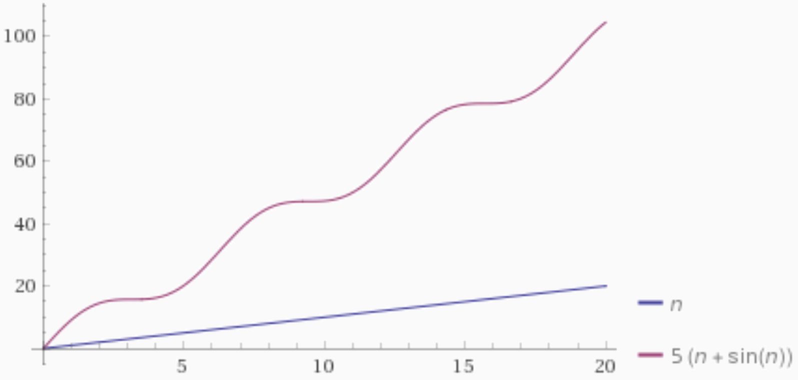
- $f(n) = O(g(n))$
- $f(n) = \Theta(g(n))$
- $f(n) = \Omega(g(n))$

Asymptotics as an ordering

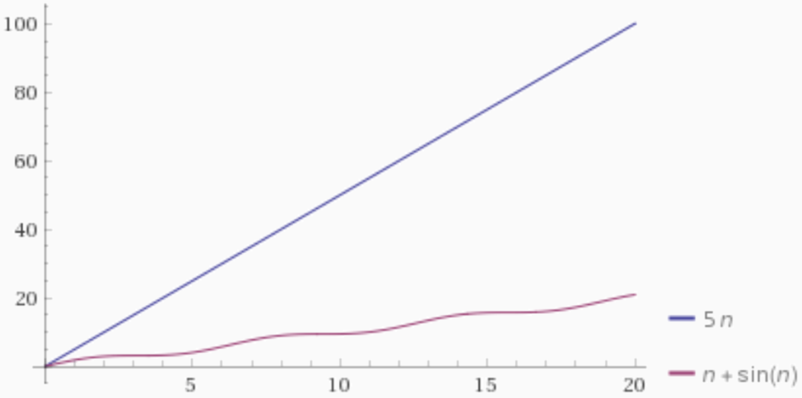
Consider $f(n) = n$ and $g(n) = n + \sin(n)$



Asymptotics as an ordering



Asymptotics as an ordering



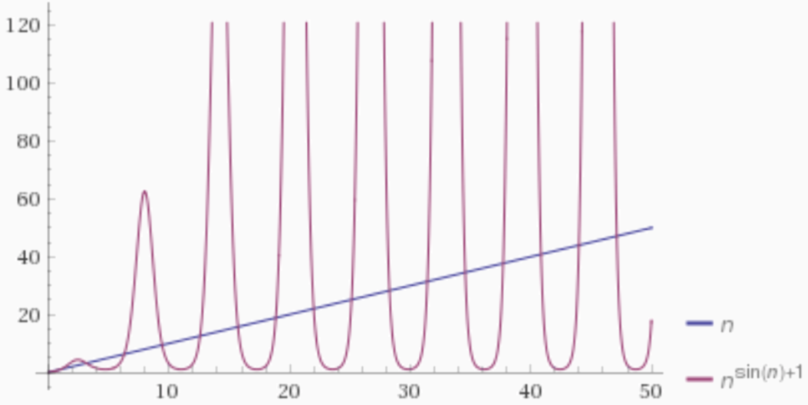
Asymptotics as an ordering

Let $f(n) = n$ and $g(n) = n^{1+\sin(n)}$.

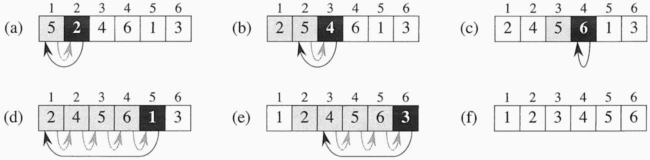
Does one of these hold?

- $f(n) = O(g(n))$
- $f(n) = \Theta(g(n))$
- $f(n) = \Omega(g(n))$

Asymptotics as an ordering



Insertion sort



INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

Insertion sort

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$i = j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
$i = i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

Insertion sort

$$
\begin{aligned}
T(n) \quad = \quad & c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
& + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) \; .
\end{aligned}
$$

Insertion sort

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
&\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .
\end{aligned}
$$

In the best case, $t_j = 1$:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) .
\end{aligned}
$$

Insertion sort

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
&\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .
\end{aligned}
$$

In the worst case, $t_j = j$:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1 \right) \\
&\quad + c_6 \left(\frac{n(n-1)}{2} \right) + c_7 \left(\frac{n(n-1)}{2} \right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8) .
\end{aligned}
$$

Selection sort

The idea of selection sort is to find the smallest element and swap it with the first element. Then find the second smallest element and swap it with the second element, and repeat till the whole array is sorted.

```
for i = 1 to n:
    minInd = i
    for j = i+1 to n:
        if a[j] < a[minInd]:
            minInd = j
    temp = a[i]
    a[i] = a[minInd]
    a[minInd] = temp
```

Common complexities

Big-O notation	Description	Notes
$O(1)$	constant	independent of input size
$O(\log n)$	logarithmic	examines subset of input
$O(n)$	linear	may examine all of input
$O(n^2)$	quadratic	generally considered feasible
$O(n^k)$	polynomial	may be feasible, depending on k
$O(2^n)$	exponential	generally considered infeasible

Algorithm design

We can design algorithms a few ways:

- incremental – sort up to $j - 1$, then sort up to j
- divide and conquer

Divide and conquer

- divide: split problem into subproblems of the same type
- conquer: solve the subproblems recursively
- combine: combine the results into a solution for the original problem

Merge sort

Given an unsorted array, if we could somehow separate it into two sorted arrays, we could then merge these two to get a final sorted array.

- Break it into two arrays
- An array of size 1 is already sorted

Merge sort

Suppose we have the following unsorted array:

45	-6	73	28	5	-1	23	42

Merge sort

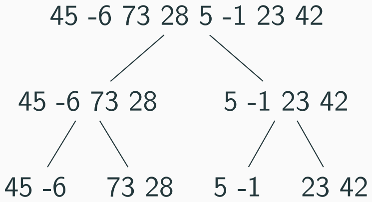
Repeatedly divide

45 -6 73 28 5 -1 23 42

45 -6 73 28   5 -1 23 42

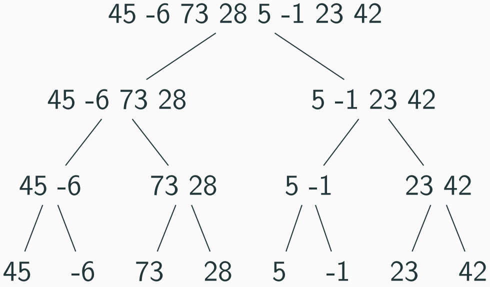
Merge sort

Repeatedly divide



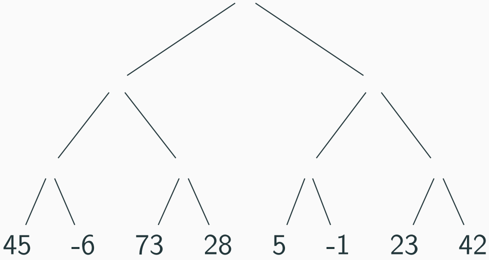
Merge sort

Repeatedly divide



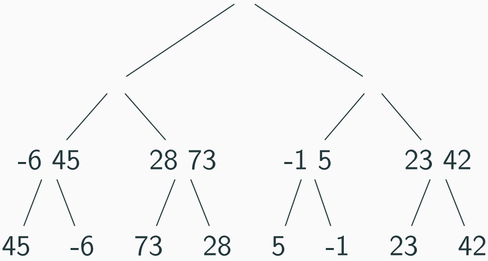
Merge sort

Now merge:



Merge sort

Now merge:



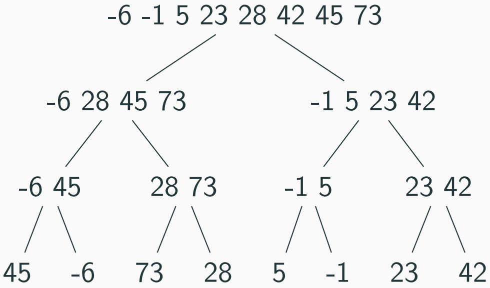
Now merge:

-6 28 45 73 | -1 5 23 42

-6 45 | 28 73 | -1 5 | 23 42

45 | -6 | 73 | 28 | 5 | -1 | 23 | 42

Now merge:


-6 -1 5 23 28 42 45 73
-6 28 45 73
-1 5 23 42
-6 45
28 73
-1 5
23 42
45
-6
73
28
5
-1
23
42

Merge sort

What is the running time of merge sort?

- Divide n elements into two sets of $n/2$ elements and solve
- Merge the results

Merge sort

How long does it take to merge two sorted $n/2$ lists?

1	5	10	20

3	4	7	42

1	3	4	5	7	10	20	42

Merge sort

What is the running time of merge sort?

$$
\begin{aligned}
T(n) &= 2T(n/2) + n = 2(2T(n/4) + n/2) + n \\
&= 4T(n/4) + 2n = 4(2T(n/8) + n/4) + 2n \\
&= 8T(n/8) + 3n \\
&= \ldots \\
&= 2^k T(n/2^k) + kn
\end{aligned}
$$

Merge sort

What is the running time of merge sort?

$$T(n) = 2^k T(n/2^k) + kn$$

Let $n = 2^k$:

$$\begin{aligned} T(n) &= nT(n/n) + kn \\ &= nT(1) + kn \\ &= n \cdot 1 + kn \\ &= n + kn \end{aligned}$$

Merge sort

What is the running time of merge sort?

$$T(n) = n + kn$$

To get rid of k, observe that $n = 2^k$ implies $k = \log n$:

$$\begin{aligned} T(n) &= n + (\log n)n \\ &= O(n) + O(n \log n) \\ &= O(n \log n) \end{aligned}$$