Asymptotic notation

Doc distance summary

Merge-sort
- Divide & conquer
- Analysis of recurrences

Readings: CLRS Chapter 4

Asymptotics

Parameterize input size as $n$ ($m, t, \text{etc}$)
Many different inputs of size $n$

$T(n) = \text{worst case running time for input size } n$

= $\max_{x: \text{input of size } n}$ running time on $x$

How can we be precise?

don't care about $T(n)$ for small $n$

"" constant factors (diff computers, languages...)""
Suppose $T(n) = 4n^2 + 22n - 12$ MS.

only care about highest order term, without constant

Say $T(n)$ is $O(g(n))$ if $\exists n_0, \forall c$ s.t.

$0 \leq T(n) \leq c \cdot g(n)$ for all $n \geq n_0$

$0 \leq 4n^2 + 22n - 12 \leq 26n^2$ for $n \geq 1$

$\therefore 4n^2 + 22n - 12 = O(n^2)$

$O$: upper bound

read as "is" or $\in$ belongs to a set

Say $T(n) = \Omega(g(n))$ if $\exists n_0, \forall d$ s.t.

$0 \leq d \cdot g(n) \leq T(n)$ for all $n \geq n_0$

$T(n) = 4n^2 + 22n - 12 \geq n^2$ for $n \geq 1$

$\therefore T(n) = \Omega(n^2)$

Say $T(n) = \Theta(g(n))$ iff $T(n) = O(g(n))$ and $T(n) = \Omega(g(n))$

$\Omega$: lower bound

$\Theta$: high order term is $g(n)$
Optimizations

<table>
<thead>
<tr>
<th>V1</th>
<th>Initial</th>
<th>Time</th>
<th>Bobsdy vs. Lewis</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>Add profiling</td>
<td>19.5s</td>
<td>(\Theta(n^2) \rightarrow \Theta(n))</td>
</tr>
<tr>
<td>V3</td>
<td>Wordlist, extend(...)</td>
<td>84.5s</td>
<td>(\Theta(n^2) \rightarrow \Theta(n))</td>
</tr>
<tr>
<td>V4</td>
<td>Dichotomy in count-freq</td>
<td>41.5s</td>
<td>(\Theta(n) \rightarrow \Theta(n))</td>
</tr>
<tr>
<td>V5</td>
<td>Process words rather than chars in get words from string</td>
<td>13.5s</td>
<td>(\Theta(n^2) \rightarrow \Theta(n))</td>
</tr>
<tr>
<td>V6</td>
<td>Merge-sort rather than insertion sort</td>
<td>6.5s</td>
<td>(\Theta(n^2) \rightarrow \Theta(n \log n))</td>
</tr>
</tbody>
</table>

V6b: eliminate sorting altogether, \(\sim 1s\) to get \(\Theta(n)\) algorithm

Logarithmic improvement: \(\log_2 n\)
Merge - sort

Divide / conquer / Combine paradigm.

- Input array of size $n$
- 2 arrays of size $n/2$
- 2 sorted arrays of size $n/2$
- Size $n$

```
5 4 7 3 6 1 9 2
3 4 5 7 1 2 6 9
```

```
i j
1 2 3 4 5 6 7 9
```

```
Inc j Inc i Inc i Inc i Inc i Inc i (array L) (array R)
```

```
T(n) = C_1 + 2 \cdot T(n/2) + C \cdot n
```
\[ T(n) = 2T(n/2) + cn \quad \text{only keep high order terms} \]
\[ = cn + 2\left(C \cdot \left(\frac{n}{2}\right) + 2\left(C \cdot \left(\frac{n}{4}\right) + \cdots \right) \right) \]

\[ T(n) = \frac{cn(\lg n + 1)}{c \cdot n} = \Theta(n \lg n) \]

**Experiment**

- **insertion-sort** \( \Theta(n^2) \)
- **merge-sort** \( \Theta(n \lg n) \)
- **built-in sort** \( \Theta(n \lg n) \) \( n = 2^i \)
- **merge-sort takes \( \approx 2.2 n \lg n \) ms**
- **insertion-sort takes \( \approx 0.2 n^2 \) ms**
- **Built-in sort (sorted) takes \( \approx 0.1 n \lg n \) ms**

20x constant factor because it is written in C

**test-merge routine**

**test-insert**
When is merge-sort (in Python) 2n\log n better than insertion sort (in C') 0.01 n^2? 
merge-sort wins for \( n \geq 2^{12} = 4096 \)

better algorithm much more valuable than hardware or compiler even for modest \( n \)

Python cost model: tomorrow's recitation many experiments of this sort. Also PS 1 set ops.