Algorithm Engineering

Jens K. Mueller

jkm@informatik.uni-jena.de

Department of Mathematics and Computer Science
Friedrich Schiller University Jena

Monday 17th November, 2014

$\mathcal{O}$ Notation

Algorithm Analysis

- All basic operations take some constant amount of time
- Algorithms performance measured on ever growing instances (asymptotic analysis)
- Specify instances by fixing their input size
- Usually worst case over all instances to give strongest guarantees

Algorithm Analysis

- Time analysis (time complexity)
- Space analysis (space complexity)

- Worst Case
- Average Case
- Best Case

- Amortized
  Average time per operation over a worst-case sequence of operations.

$\mathcal{O}$ Notation

- $f(n) \in \mathcal{O}(g(n))$
  There exists a $c > 0$ for all $n > n_0$ such that $f(n) \leq c\, g(n)$.

  f is asymptotically bounded above by g (up to a constant factor).

- $f(n) \in \Omega(g(n)) :\Leftrightarrow g(n) \in \mathcal{O}(f(n))$

  f is asymptotically bounded below by g (up to a constant factor).

- $f(n) \in \Theta(g(n))$
  Shorthand for: $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$.

Pitfalls

- Worst case (max) vs. average case (mean) vs. best case (min)
- Input sizes may be small or are not large enough
- Neglects constants
- Amortized analysis

Vector and Lists

Worst Case Time Complexities

operation \ structure	vector	linked list	doubly linked list
front	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
push front	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
pop front	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
back	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
push back	$\Theta_a(1)$	$\Theta(n)$	$\Theta(1)$
pop back	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
index	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
insert	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
remove	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$

Stack and Queue

Worst Case Time Complexities

operation \ structure	stack	queue
front	N/A	$\Theta(1)$
push front	N/A	N/A
pop front	N/A	$\Theta(1)$
back	$\Theta(1)$	$\Theta(1)$
push back	$\Theta(1)$[1]	$\Theta(1)$[1]
pop back	$\Theta(1)$	N/A

[1] using a list; Using a vector gives $\Theta_a(1)$

Sorting Problem

Input: Given n comparable elements
Output: Permutation of the n elements such that the neighboring elements are in order according to a total order $\leq$.

Minimal Number of Comparisons

- There are $n!$ permutations on n elements.
- Comparing two elements at a time results in a complete binary tree with $n!$ leaves
- Tree depth is $\log n!$
- $\log n! \in \Omega(n \log n)$

$\log(n!) \in \Omega(n \log n)$

$$\begin{aligned}
\left(\frac{n}{2}\right)^{\frac{n}{2}} &= \underbrace{\frac{n}{2} \cdot \ldots \cdot \frac{n}{2}}_{\frac{n}{2}} \\
&\leq \frac{n}{2} \cdot \ldots \cdot n \\
&\leq 1 \cdot \ldots \cdot \frac{n}{2} \cdot \ldots \cdot n \\
&= n!
\end{aligned}$$

Applying the monotone increasing function log to both sides and changing sides yields

$$\log(n!) \geq \log\left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$\log(n!) \in \Omega(n \log n)$ (cont.)

$$
\begin{aligned}
\log(n!) &\geq \log\left(\frac{n}{2}\right)^{\frac{n}{2}} \\
&= \frac{n}{2}\log\frac{n}{2} \\
&= \frac{1}{2}n\log n - \frac{1}{2}n\log 2 \\
&= \frac{1}{2}n\log n - \frac{1}{2}n \\
&= \frac{1}{4}n\log n + \underbrace{\frac{1}{4}n\log n - \frac{1}{2}n}_{\geq 0 \text{ for } n \geq 4} \\
&\geq \frac{1}{4}n\log n
\end{aligned}
$$

Insertion Sort

Insertion sort

1. Assume first i elements are sorted
2. Insert new element $i + 1$ at sorted position

Algorithmic improvements

- Use less memory accesses

Quicksort

Quicksort

1. Choose pivot element
2. Partition
3. Recurse

Algorithmic improvements

- Median of three
- Random pivot
- Insertion sort for small input
- Fat partition

Error Categories