Logarithms

- Involved in many important runtime results: Sorting, binary search etc.
- Logarithms grow slowly, much more slowly than any polynomial but faster than a constant.
- Definition: \( \log_b N = K \) if \( B^K = N \). \( B \) is the base of the log.
- Examples:
  - \( \log_2 8 = 3 \) because \( 2^3 = 8 \).
  - \( \log_{10} 100 = 2 \) because \( 10^2 = 100 \).
  - \( 2^{10} = 1024 \) (K), so \( \log_{10} 1024 = 10 \).
  - \( 2^{30} = 1M \), so \( \log_{10} 1M = 20 \).
  - \( 2^{60} = 1G \), so \( \log_{10} 1G = 30 \).

Runtime Analysis

January, 2021

Thanks to Prof. Nuri Haspeli for many of these slides

Logarithms

- It requires \( \log_N K \) digits to represent \( K \) numbers in base\( N \).
- It requires approx. \( \log_N K \) multiplications by 2 to get from 1 to \( K \).
- It requires approx. \( \log_N K \) divisions by 2 to get from \( K \) to 1.
- Computers work in binary, so in order to calculate how many numbers a certain amount of memory can represent we use \( \log_2 \).
- Small example: 3 bits can represent 8 different numbers: 000, 001, 010, 011, 101, 110, and 111, and \( \log_2 8 = 3 \)

Useful Logarithm Rules

- \( \log(r/n) = \log(r) - \log(n) \)
- \( \log(n^k) = k \log(n) \)
- \( \log(b^c) = c \log(b) \)
- If the base of log is not specified, assume it is base 2 (although for runtime analysis it doesn’t matter)
- \( \log: base \ 2 \)
- \( \ln: base \ 2 \)
- \( \log_{10}: base \ 10 \)
- \( \log_{256}: base \ 256 \)
Runtime Analysis

- When we develop an algorithm we want to know how many resources it requires.
- Let $T$ and $N$ be positive numbers. $N$ is the size of the problem and $T$ measures a resource: Runtime, CPU cycles, disk space, memory etc.
- Order of growth can be important. For example, sorting algorithms can perform quadratically ($n^2$) or as $n \log(n)$. Very big difference for large inputs.
- We care less about constants, so $100N = O(N)$, $100N + 200 = O(N)$.
- Constant can be important when choosing between two similar run-time algorithms. Example: quicksort.
- * It is not always 100% clear what the "size of the problem" is. More on that later.

Runtime Analysis – Big-$O$ Notation

$T(N) = O(F(N))$ if there are positive constants $c$ and $N_0$ such that $T(N) \leq cF(N)$ for all $N \geq N_0$.
- Example: Show that $2N + 4 = O(N)$.
  - This means – find actual $c$ and $N_0$ (there is more than one correct answer).
  - Idea: $3N$ eventually beats $2N$.
  - $2N + 4 \leq 3N$ at $N = 4/3$.
  - So $2N + 4 < 3N$ for $N > 2$.
  - i.e., $c = 3$, $N_0 = 2$ (or any number > 4/3).

Runtime Analysis – $\Omega$ Notation

$T(N) = \Omega(F(N))$ if there are positive constants $c$ and $N_0$ such that $T(N) \geq cF(N)$ for all $N \geq N_0$.
- Example: Show that $2N + 4 = \Omega(N)$.
  - This means – find actual $c$ and $N_0$ (there is more than one correct answer).
  - That’s easy: $c = 2$, $N_0 = 1$.

~ Notation in S&W

- Note that although big-$O$ is the standard notation for expressing growth of functions in C.S., S&W prefer the applied mathematician’s – (tilde) notation. See pp.176-179.
- How are they related?
- Very simply. In the usual case that $O(F(N)) = O(F(N)) = O(N)$, the case of an "asymptotically tight bound", we can say $F(N) \sim \Omega(N)$.
- For our previous example, we had $T(N) = 2N + 4 = \Omega(N)$ and $T(N) = O(N)$ also, so we can say $T(N) \sim N$.

Can ~ do the whole job?

- How about an example that doesn’t qualify for ~?
- Here’s one: $T(N) = N \cdot \sin(N)$. Since $\sin(N) \leq 1$ for all $N$, we see that $T(N) = O(N)$. But $\sin(N)$ goes 0 and negative periodically, so no (positive) function qualifies as $\Omega(N)$.
- You can see that we had to reach pretty far to get an example. All polynomial functions have asymptotically tight bounds, as do other combinations of logs and powers.
- The function has to wiggle or jump around a lot to avoid having a tilde approximation.
Runtime Analysis

- When the runtime is estimated as a polynomial we care about the leading term only.
- Thus $3n^3 + n^2 + 2n + 17 = O(n^3)$ because eventually the leading cubic term is bigger than the rest.
- For a really good estimate on the runtime it's good to have both the $O$ and the $\Omega$ estimates (upper and lower bounds).
- But we often settle for the big $O$ bound. It's the most important.