Lecture 6
Asymptotic Analysis
Today

- Work (sequential runtime) and span (parallel runtime)
- Recurrence relations
- Exact and approximate solutions
- Improving efficiency

program $\rightarrow$ recurrence $\rightarrow$ work/span
asymptotic

- We assume basic ops take \textit{constant time}
- Want to find running time $f(n)$, for \textit{large} $n$
  - an estimate, independent of architecture
- Give big-O classification

$f(n)$ is $O(g(n))$

if there are $N$ and $c$ such that

$\forall n \geq N, f(n) \leq c \cdot g(n)$
The graph below compares the running times of various algorithms.

- Linear -- $O(n)$
- Quadratic -- $O(n^2)$
- Cubic -- $O(n^3)$
- Logarithmic -- $O(\log n)$
- Exponential -- $O(2^n)$
- Square root -- $O(\sqrt{n})$
asymptotic

• **Ignore** additive constants

  \[ n^5 + 1000000 \text{ is } O(n^5) \]

• **Absorb** multiplicative constants

  \[ 1000000n^5 \text{ is } O(n^5) \]

• Be as accurate as you can

  \[ O(n^2) \subset O(n^3) \subset O(n^4) \]

• Use and learn common terminology

  logarithmic, linear, polynomial, exponential
work

• $W(e)$, the work of $e$, is the time needed to evaluate $e$ sequentially, on a single processor
  • count each operation as constant-time
  • work = total number of operations

• Often have a function $\text{foo}$ and a notion of size for argument values, and want to find $W_{\text{foo}}(n)$, the work of $\text{foo}(v)$ when $v$ has size $n$

  May want exact or asymptotic estimate
Analyzing \texttt{rev}

\begin{verbatim}
(* rev : int list * int list -> int list
  REQUIRES: true
  ENSURES:  trev(L, acc) == (rev L) @ acc
*)

fun rev([] : int list) : int list = []
| rev(x::xs : int list) : int list = rev(xs) @ [x]

fun @ ([]: int list, r: int list) : int list = r
| @ (x::l, r) = x :: @(l,r)
\end{verbatim}
Analyzing `append`

```haskell
fun @ ([], r) = r
    | @ (x::l, r) = x :: @l, r
```

- size of first list
- size of second list

Work of `@`\n
\[ W@ (n, m) \]

- \[ W@ (0, m) = c_0 \]
- \[ W@ (n, m) = c_1 + W@ (n-1, m) \]

Easy to prove by induction that \[ W@ (n, m) = n \cdot c_1 + c_0 \]
Analyzing \texttt{rev}

\begin{verbatim}
fun rev([]) = []
  | rev(x::xs) = rev(xs) @ [x]
\end{verbatim}

\[
W_{\text{rev}}(0) = c_0
\]

\[
W_{\text{rev}}(n) = c_1 + W_{\text{rev}}(n-1) + W @ (n-1, 1)
\]

\[
W_{\text{rev}}(n) \leq c_1 + W_{\text{rev}}(n-1) + c_2 (n-1)
\]

\[
\leq c_1 + c_2 n + W_{\text{rev}}(n-1)
\]

Can prove by induction that $O(n^2)$

@ is $O(n)$
Analyzing \texttt{trev}

\begin{verbatim}
fun trev([], acc) = acc
    | trev(x::xs, acc) = trev(xs, x::acc)
\end{verbatim}

\[ W_{trev}(0, m) = c_0 \]
\[ W_{trev}(n, m) = c_1 + W_{trev}(n-1, m+1) \]

Can prove by induction that \( W_{trev}(n, m) \) is \( O(n) \)
datatype tree = Empty | Node of tree * int * tree

(* sum : tree -> int
  REQUIRES: true
  ENSURES: sum t returns the sum of all the integers in t *)

fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
Analysis of \texttt{sum}

\begin{verbatim}
fun sum(Empty : tree) : int = 0
  | sum(Node(l,x,r)) = (sum l) + x + (sum r)
\end{verbatim}

Let \( n \) be the number of integers in a tree \( t \)

\[
W_{\text{sum}}(0) = c_0
\]

\[
W_{\text{sum}}(n) = c_1 + W_{\text{sum}}(n_l) + W_{\text{sum}}(n_r)
\quad \text{O}(n)
\]

number of ints in the left subtree of \( t \)

number of ints in the right subtree of \( t \)
Opportunity for parallelism

```haskell
fun sum(Empty : tree) : int = 0
| sum(Node(l,x,r)) = (sum l) + x + (sum r)
```

Let $n$ be the number of integers in a tree $t$

$$S_{\text{sum}}(0) = c_0$$
$$S_{\text{sum}}(n) = c_1 + \max(S_{\text{sum}}(n_l), S_{\text{sum}}(n_r))$$

- number of ints in the left subtree of $t$
- number of ints in the right subtree of $t$
Opportunity for parallelism

```
fun sum(Empty : tree) : int = 0
| sum(Node(l, x, r)) = (sum l) + x + (sum r)
```

Let $n$ be the number of integers in a tree $t$

$$
S_{\text{sum}}(0) = c_0
$$

$$
S_{\text{sum}}(n) = c_1 + \max(S_{\text{sum}}(n_l), S_{\text{sum}}(n_r))
$$

If tree is balanced span is $O(\log n)$
Without that assumption it is $O(n)$
Sorting
comparison

(compare : int * int -> order)

datatype order = LESS | EQUAL | GREATER

fun compare(x:int, y:int):order =
  if x<y then LESS else
  if y<x then GREATER else EQUAL

compare(x,y) = LESS             if x<y
compare(x,y) = EQUAL         if x=y
compare(x,y) = GREATER     if x>y

Pre-defined as Int.compare
A list of integers is $\leq$-sorted (or just "sorted") if each item in the list is $\leq$ all items that occur later.

```ocaml
fun sorted [] = true
| sorted [x] = true
| sorted (x::y::L) =
  (Int.compare(x,y) <> GREATER) andalso sorted(y::L)
```
For all sorted integer lists L, 
ins(x, L) is a sorted permutation of x::L.
**isort**

isort : int list -> int list

(* REQUIREES: true *)

(* ENSURES: isort(L) is a sorted perm of L *)

fun isort [] = []

| isort (x::L) = ins (x, isort L)

For all values L: int list,

isort L is a sorted permutation of L.
work

- Let $W_{\text{ins}}(n)$ be the work for $\text{ins}(x, L)$ when $x, L$ are values and $L$ has length $n$
  
  $W_{\text{ins}}(n)$ is $O(n)$

- Let $W_{\text{isort}}(n)$ be the work for $\text{isort}(L)$ when $L$ is a list of length $n$

  $W_{\text{isort}}(0) = 1$
  $W_{\text{isort}}(n) = 1 + W_{\text{ins}}(n-1) + W_{\text{isort}}(n-1)$
  
  for $n > 0$

  $W_{\text{isort}}(n)$ is $O(n^2)$
finding solutions

Try to find a closed form solution for $W(n)$ (usually, by guessing and induction)

OR Code the recurrence in ML, test for small $n$, look for a common pattern

OR Find solution to a simplified recurrence with the same asymptotic properties

OR Appeal to table of standard recurrences
standard results

• $T(n) = c + T(n-1)$ \quad O(n)
• $T(n) = c + n + T(n-1)$ \quad O(n^2)
• $T(n) = c + T(n \text{ div } 2)$ \quad O(\log n)
• $T(n) = c + 2T(n \text{ div } 2)$ \quad O(n)
• $T(n) = c + n + T(n \text{ div } 2)$ \quad O(n)
• $T(n) = c + n + 2T(n \text{ div } 2)$ \quad O(n \log n)
• $T(n) = c + kT(n-1)$ \quad O(k^n)$