Today

- Work ( = sequential runtime )
- Recurrences
  - exact and approximate solutions
- Improving efficiency

program → recurrence → work
asymptotic

- Basic operations take **constant time**
- Want to find an estimate of the **runtime** to evaluate \( f(n) \), for **large** \( n \)
  - *independent* of architecture
- We will give a **big-O** classification

\( f(n) \) is \( \mathcal{O}(g(n)) \)

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 -- $O(\sqrt{n})$
motivation

Why take exponential time when we can take quadratic time?
asymptotic

• *Ignore* additive constants
  \[ n^5 + 1000000 \text{ is } \mathcal{O}(n^5) \]

• *Absorb* multiplicative constants
  \[ 1000000n^5 \text{ is } \mathcal{O}(n^5) \]

• Be as accurate as you can
  \[ \mathcal{O}(n^2) \subset \mathcal{O}(n^3) \subset \mathcal{O}(n^4) \]

• 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
  • each operation counts as constant-time
  • work = total number of operations

• Often have a function $f$ and a notion of size for argument values, and want to find $W_f(n)$, the work of $f(v)$ when $v$ has size $n$

  May want exact or asymptotic estimate
work and evaluation

• Evaluation steps $e \Rightarrow e'$ represent basic ops, so the work for $e$ is the number of steps.

If $e \Rightarrow^{(k)} v$ then $W(e) = k$

$(2+2)+(2+2) \Rightarrow 4+(2+2)$
$\Rightarrow 4+4$
$\Rightarrow 8$

$W((2+2)+(2+2)) = 3$

$W(e_1+e_2) = W(e_1) + W(e_2) + 1$
work and application

If \( f \) is a function value and \( e \Rightarrow^{(k)} v \) then \( W(f \ e) = k + W(f \ v) \)

\[
W((\text{fn } x \Rightarrow x+x) \ (2+2)) = 1 + W((\text{fn } x \Rightarrow x+x) \ 4) = 1 + 1 + W(4+4) = 3
\]

\[
(\text{fn } x \Rightarrow x+x) \ (2+2) => (\text{fn } x \Rightarrow x+x) \ 4 => 4+4 => 8
\]
recurrences

• Given a recursive function definition for $f$ and a non-negative argument size that decreases in every recursive call

• Extract a recurrence relation for the applicative work of $f$

\[ W_f(n) = \text{work of } f \ v \ \text{on values } v \ \text{of size } n \]

Idea: express $W_f(n)$ in terms of $W_f(m)$, $0 \leq m < n$

Q: When can this method succeed?

A: If the work of $f \ v$ depends only on the size of $v$ (!)
fun \( f(x) = \) \( \begin{cases} 1 & \text{if } x=0 \\ x + f(x-1) & \text{else} \end{cases} \)

if \( x \geq 0 \), argument value is non-negative, decreases…

\[
W_f(n) = \begin{cases} k_1 & \text{if } n=0 \\ k_2 + W_f(n-1) & \text{else} \end{cases}
\]

where \( k_1, k_2 \) are constants

\[
W_f(0) = k_1 \\
W_f(n) = k_2 + W_f(n-1) \quad \text{for } n>0
\]

\[
W_f(n) = k_1 + n k_2 \quad \text{for all } n \geq 0
\]
A recurrence for the work to evaluate $\text{Fib}(n)$

$$W_{\text{Fib}(0)} = c_0$$
$$W_{\text{Fib}(1)} = c_0$$
$$W_{\text{Fib}(n)} = W_{\text{Fib}(n-1)} + W_{\text{Fib}(n-2)} + c_1$$
for some constants $c_0, c_1$
finding solutions

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

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

OR Find solution to a \textit{simplified} recurrence with the same asymptotic properties

OR Appeal to table of standard recurrences
fun exp (n:int):int = 
  if n=0 then 1 else 2 * exp (n-1)

Let M be  (fn n => if n=0 then 1 else 2 * exp(n-1))

exp 4 => M 4

=> (5) 2 * (M 3)
=> (5) 2 * (2 * (M 2))
=> (5) 2 * (2 * (2 * (M 1))))
=> (5) 2 * (2 * (2 * (2 * (M 0))))
=> (3) 2 * (2 * (2 * (2 * (2 * 1))))
=> (4) 16

exp 4 => (28) 16
It’s not hard to prove that for all $n \geq 0$,

$$\exp n \Rightarrow (5n+8) \ k,$$

where $k$ is the numeral for $2^n$

But do we need to be so accurate?

And does $5n+8$ tell us about actual runtime in milliseconds?

No! But it does tell us runtime is linear.
big-O

- It’s useful to classify runtimes asymptotically
- This abstracts away from additive and multiplicative constants
  (which may be machine-dependent, so not very significant)
- And ignores runtime on small inputs
  (which may be special-cased in the code, so don’t imply much)
Let $W_{\text{exp}}(n)$ be the runtime for $\text{exp}(n)$

- $W_{\text{exp}}(0) = c_0$
- $W_{\text{exp}}(n) = W_{\text{exp}}(n-1) + c_1$ for $n > 0$

for some constants $c_0$ and $c_1$

- $c_0$ cost of $n=0$
- $c_1$ cost of $n=0$, $n-1$, mult by 2
solution

• Can prove by induction on \( n \) that

\[
W_{\text{exp}}(n) = c_0 + n c_1 \quad \text{for } n \geq 0
\]

the work of \( \exp(n) \) is linear in \( n \)
• $W_{\text{exp}}(n) = c_0 + n c_1$
• $W_{\text{exp}}(n)$ is $O(n)$

O-class for $W_{\text{exp}}(n)$ is independent of $c_0, c_1$

Let $N=42$, $c = \max(c_0, c_1) + 1$. For all $n \geq N$,

$W_{\text{exp}}(n) \leq c \ n$

(would also work with $N=1$)

(would also work with an even bigger $c$)
summary

- We’ve shown that for $n \geq 0$, $\exp n$ computes the value of $2^n$ in $O(n)$ steps.

- This fact is independent of machine details (provided basic operations are constant time).

- Can we do better?
faster \text{ exp}?

- The definition of exp relies on the fact that
  \[ 2^n = 2 \ (2^{n-1}) \]

- Everybody knows that
  \[ 2^n = (2^{n \ \text{div} \ 2})^2 \ \text{ if } n \ \text{is even} \]
fun square(x:int):int = x * x

fun fastexp (n:int):int =
  if n=0 then 1 else
  if n mod 2 = 0 then square(fastexp (n div 2))
  else 2 * fastexp(n-1)

fastexp 4 = square(fastexp 2)
  = square(square (fastexp 1))
  = square(square (2 * fastexp 0))
  = square(square (2 * 1))
  = square 4 =16
is it faster?

fun fastexp (n:int):int = 
  if n=0 then 1 else 
    if n mod 2 = 0 then square(fastexp (n div 2)) 
    else 2 * fastexp(n-1)

Let $W_{\text{fastexp}}(n)$ be the work for fastexp(n)

$W_{\text{fastexp}}(0) = k_0$

$W_{\text{fastexp}}(n) = W_{\text{fastexp}}(n \text{ div } 2) + k_1$ for $n>0$, even

$W_{\text{fastexp}}(n) = W_{\text{fastexp}}(n-1) + k_2$ for $n>0$, odd

for some constants $k_0, k_1, k_2$
is it faster?

Let $W_{\text{fastexp}}(n)$ be the work for fastexp(n)

<table>
<thead>
<tr>
<th>$W_{\text{fastexp}}(0)$</th>
<th>$= c_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{\text{fastexp}}(1)$</td>
<td>$= c_1$</td>
</tr>
<tr>
<td>$W_{\text{fastexp}}(n)$</td>
<td>$= W_{\text{fastexp}}(n \div 2) + c_2$ for $n&gt;1$, even</td>
</tr>
<tr>
<td></td>
<td>$= W_{\text{fastexp}}(n \div 2) + c_3$ for $n&gt;1$, odd</td>
</tr>
</tbody>
</table>

for some constants $c_0, c_1, c_2, c_3$

**fun** fastexp (n:int):int =
  if n=0 then 1 else
  if n mod 2 = 0 then square(fastexp (n div 2))
    else 2 * fastexp(n-1)
solution?

- Not so obvious how to solve for $W_{\text{fastexp}}(n)$
- A closed form would involve $c_0, c_1, c_2, c_3$
- But we only care about \textit{asymptotic} behavior
- So we can work with a \textit{simpler} recurrence that has the \textit{same} asymptotic properties

\textbf{simplification:} choose each constant to be 1
simplified
recurrence

Let $T_{\text{fastexp}}(n)$ be given by

\[
T_{\text{fastexp}}(0) = 1 \\
T_{\text{fastexp}}(1) = 1 \\
T_{\text{fastexp}}(n) = T_{\text{fastexp}}(n \div 2) + 1 \quad \text{for } n > 1
\]

$W_{\text{fastexp}}(n)$ and $T_{\text{fastexp}}(n)$ are asymptotically equivalent
(belong to the same big-O class)
solution

• For n > 1, $T_{\text{fastexp}}(n)$ is defined like $\log(n)$

  ```
  fun log n = 
    if n=1 then 0 else log(n div 2) + 1
  ```

• We know that $\log(n) = \log_2(n)$ for all $n > 0$

• Can show that there is a constant $c$ such that

  $$T_{\text{fastexp}}(n) \leq c \log_2(n)$$

  for all large enough $n$
• \( T_{\text{fastexp}}(n) \) is \( O(\log_2 n) \)

• \( W_{\text{fastexp}}(n) \) depends on \( c_0, c_1, c_2, c_3 \)

• We can find constants \( c_{\text{low}} \) and \( c_{\text{high}} \) such that

\[
c_{\text{low}} \cdot T_{\text{fastexp}}(n) \leq W_{\text{fastexp}}(n) \leq c_{\text{high}} \cdot T_{\text{fastexp}}(n)
\]

and this implies that

\( W_{\text{fastexp}}(n) \) is also \( O(\log_2(n)) \)
really, faster

• Work of $\exp(n)$ is $O(n)$
• Work of $\text{fastexp}(n)$ is $O(\log n)$
• $O(\log n)$ is a proper subset of $O(n)$
• $\text{fastexp}$ is *asymptotically faster* than $\exp$
even faster?

- The definition of \texttt{fastexp} relies on

  \[
  2^n = (2^{n \ \text{div} \ 2})^2 \quad \text{if } n \text{ is even}
  \]

  \[
  2^n = 2 \cdot (2^{n-1}) \quad \text{if } n \text{ is odd}
  \]

- A moment’s thought tells us that

  \[
  2^n = 2 \cdot (2^{(n \ \text{div} \ 2)})^2 \quad \text{if } n \text{ is odd}
  \]
fun pow (n:int):int =
  case n of
   0 => 1
  | 1 => 2
  | _ => let
    val k = pow(n div 2)
    in
    if n mod 2 = 0 then k*k else 2*k*k
  end
work of \( \text{pow}(n) \)

\[
\begin{align*}
W_{\text{pow}}(0) &= c_0 \\
W_{\text{pow}}(1) &= c_1 \\
W_{\text{pow}}(n) &= c_2 + W_{\text{pow}}(n \div 2) \text{ for } n > 1
\end{align*}
\]

Same recurrence as \( W_{\text{fastexp}} \)

Same asymptotic behavior

\( \text{pow}(n) \) is \( O(\log n) \)
• $\text{fastexp}(n)$ and $\text{pow}(n)$ have $O(\log n)$ work.

• For $n \geq 0$, $\text{fastexp}(n) = \text{pow}(n)$.

• For $n < 0$, $\text{fastexp}(n)$ and $\text{pow}(n)$ fail to terminate.

• So $\text{fastexp}$ and $\text{pow}$ are extensionally equivalent and have the same asymptotic work classification.
fun badpow (n:int):int =

  case n of
    0 => 1
  | 1 => 2
  | _ => let
    val k2 = badpow(n div 2)*badpow(n div 2)
  in
    if n mod 2 = 0 then k2 else 2*k2
  end
work of badpow(n)

\[ W_{\text{badpow}}(0) = c_0 \]
\[ W_{\text{badpow}}(1) = c_1 \]
\[ W_{\text{badpow}}(n) = c_2 + 2 W_{\text{badpow}}(n \text{ div } 2) \quad \text{for } n > 1 \]

Same asymptotic class as

\[ T_{\text{badpow}}(0) = 1 \]
\[ T_{\text{badpow}}(1) = 1 \]
\[ T_{\text{badpow}}(n) = 1 + 2 T_{\text{badpow}}(n \text{ div } 2) \quad \text{for } n > 1 \]
examples

\[ T_{\text{badpow}}(2^0) = 1 \]

\[ T_{\text{badpow}}(2^1) = 1 + 2 \times T_{\text{badpow}}(2^0) \]
\[ = 1 + 2 \times 1 = 3 \]

\[ T_{\text{badpow}}(2^2) = 1 + 2 \times T_{\text{badpow}}(2^1) \]
\[ = 1 + 2 \times 3 = 7 \]

\[ T_{\text{badpow}}(2^m) = 2^{m+1} - 1 \]
analysis

- $T_{\text{badpow}}(2^m)$ is $O(2^m)$
- $W_{\text{badpow}}(2^m)$ is $O(2^m)$
- This implies that $W_{\text{badpow}}(n)$ is $O(n)$

$W_{\text{pow}}(n)$ is $O(\log n)$

$O(\log n) \subset O(n)$

**pow** is asymptotically faster than **badpow**
list reversal

fun rev [ ] = [ ]
  \ |
  \ | \[x\]
  |\rev (x::L) = (rev L) @ [x]

For list values A and B, \( W_\@(A, B) \) is linear in the length of A

For all L, length(rev L) = length(L)

Runtime of rev(L) depends only on length of L
work of rev

fun rev [ ] = [ ]
    | rev (x::L) = (rev L) @ [x]

- $W_{rev}(n) = \text{work to reverse a list of length } n$

\[
W_{rev}(0) = 1 \\
W_{rev}(n) = W_{rev}(n-1) + (n-1) + 1
\]
solution

\[ W_{\text{rev}}(n) = n + W_{\text{rev}}(n-1) \]
\[ = n + (n-1) + W_{\text{rev}}(n-2) \]
\[ = n + (n-1) + ... + 1 + W_{\text{rev}}(0) \]
\[ = \frac{1}{2} n(n+1) + 1 \]

\[ W_{\text{rev}}(n) \text{ is } O(n^2) \]

quadratic runtime
faster rev

• Use an extra argument to accumulate the reversed list
  
  \[
  \text{revver} : \text{int list} \times \text{int list} \rightarrow \text{int list}
  \]

• Instead of \textit{append} after the recursive call, do a \textit{cons} before the recursive call
  
  \[
  \text{fun revver}([\ ], \text{A}) = \text{A} \\
  \quad | \quad \text{revver}(\text{x}::\text{L}, \text{A}) = \text{revver}(\text{L}, \text{x}::\text{A})
  \]
fun revver([], A) = A
  | revver(x::L, A) = revver(L, x::A)

fun Rev L = revver(L, [])

For all L, A, revver(L, A) = (rev L) @ A

For all L, Rev L = rev L
analysis

• Explain why $W_{\text{revver}}(n)$ is $O(n)$
standard results

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