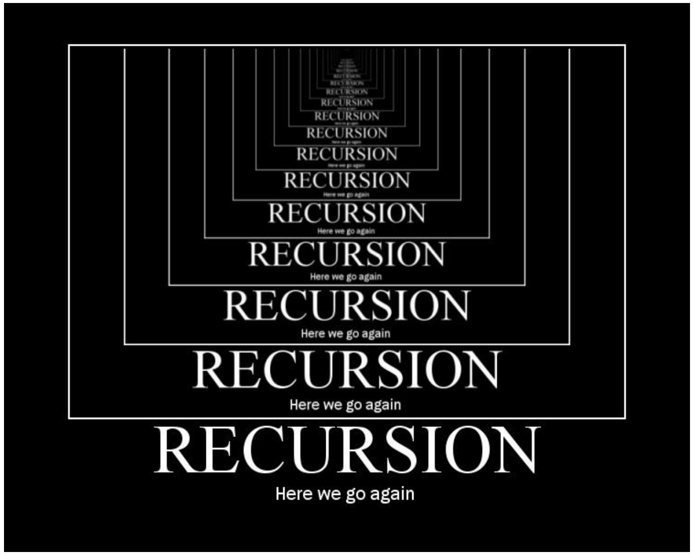
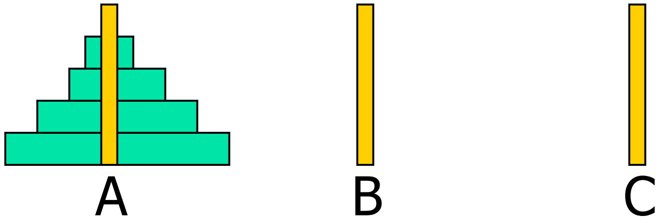
Chapter 3
Recursion and Recurrences



References:
[CLRS 4.3-4.5]
[KT 5.2]
[DPV 2.2]

Recursive Algorithms

- Recall: what is recursion?
  - Recursive algorithms: algorithms that call itself
- Example: Tower of Hanoi
  - Move n discs from A to C
  - One disc at a time
  - Bigger discs never on top of smaller ones
  - Minimise number of moves



  - Observation: if you know how to solve the problem with n-1 discs, you know how to solve it with n discs

Tower of Hanoi Algorithm

- Algorithm:
  - Recursively move the top n-1 discs to B (fixing the bottom disc at A)
  - Move bottom disc to C
  - Recursively move the n-1 discs in B to C (fixing the bottom disc at C)

- Recursion is very powerful in algorithm design: ideas and proofs are simple (once you understand it…)
- Often, we can convert recursive algorithms into non-recursive ones

Analysing Recursive Algorithms

- Efficiency of Tower of Hanoi recursive algorithm
  - Number of moves T(n) = 2 T(n-1) + 1 (why?)
  - Is it efficient?

n	1	2	3	4	5	10	100
T(n)	1	3	7	15	31	?	?

- Can we obtain a general formula for T(n)?
  - $T(n) = 2^n - 1 \leftarrow$ How to derive this?

Recurrences

- Complexity of recursive algorithms represented by *recurrence formula*
  - Another example: $T(n) = T(n/2) + 1$, $T(1) = 1$
- However, we want to express $T(n)$ as function of n directly $\rightarrow$ *"solve" the recurrence*
- Recursive algorithms always have a *base case* where the recursion stops
  - Tower of Hanoi: $T(1) = 1$
- Ways to solve recurrences
  - Iterative substitution
  - Induction
  - Master theorem

Method 1: Iterative Substitution

- $T(n) = T(n - 1) + n$, $T(1) = 1$

(replace n with $n-1$:)

$T(n-1) = T(n-1-1) + n-1$

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= (\, T(n-2) + (n-1) \,) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&= \ldots \\
&= T(1) + 2 + 3 + \ldots + n \\
&= 1 + 2 + 3 + \ldots + n \\
&= n(n+1)/2
\end{aligned}
$$

Iterative Substitutions

- Steps:
  - "Unroll" a few steps
  - Observe patterns
  - Obtain general expression for k unrollings
  - Substitute base cases
- Skills:
  - Summation of AP/GP:
  - $1 + 2 + \ldots + n = n(n+1)/2$
  - $1 + r + r^2 + \ldots + r^n = (r^{n+1} - 1) / (r - 1)$
  - $1 + r + r^2 + \ldots + r^n < 1/(1 - r)$ if $0 < r < 1$

Another Example: Binary Search

- To run: call Binary-Search(A,1,n,x)

```
/* search array A[i..j] for element x */
Binary-Search(A,i,j,x)
{
  if (i > j) return "not found" /* base case */
  mid := round((i+j)/2)
  if (A[mid] == x) return mid /* found */
  else if (A[mid] > x)
    Binary-Search(A,i,mid-1,x) /* lower half */
  else
    Binary-Search(A,mid+1,j,x) /* upper half */
}
```

Recurrence for Binary Search

- Let $T(n)$ be the time complexity for n elements
  - $T(n) = T(n/2) + 1$ (check middle element, then recurse)
  - $T(1) = 1$ (base case, just compare)

$T(n/2) = T((n/2)/2) + 1$

$$
\begin{aligned}
T(n) &= T(n/2) + 1 \\
&= (\ T(n/4) + 1\ ) + 1 \\
&= T(n/8) + 1 + 1 + 1 \\
&= \ldots \\
&= T(n/2^k) + k \\
&= T(1) + \log n \quad \text{(when } k = \log n\text{)} \\
&= 1 + \log n \\
&= O(\log n)
\end{aligned}
$$

Yet Another Example

- $T(n) = 2\ T(n/2) + n^2$

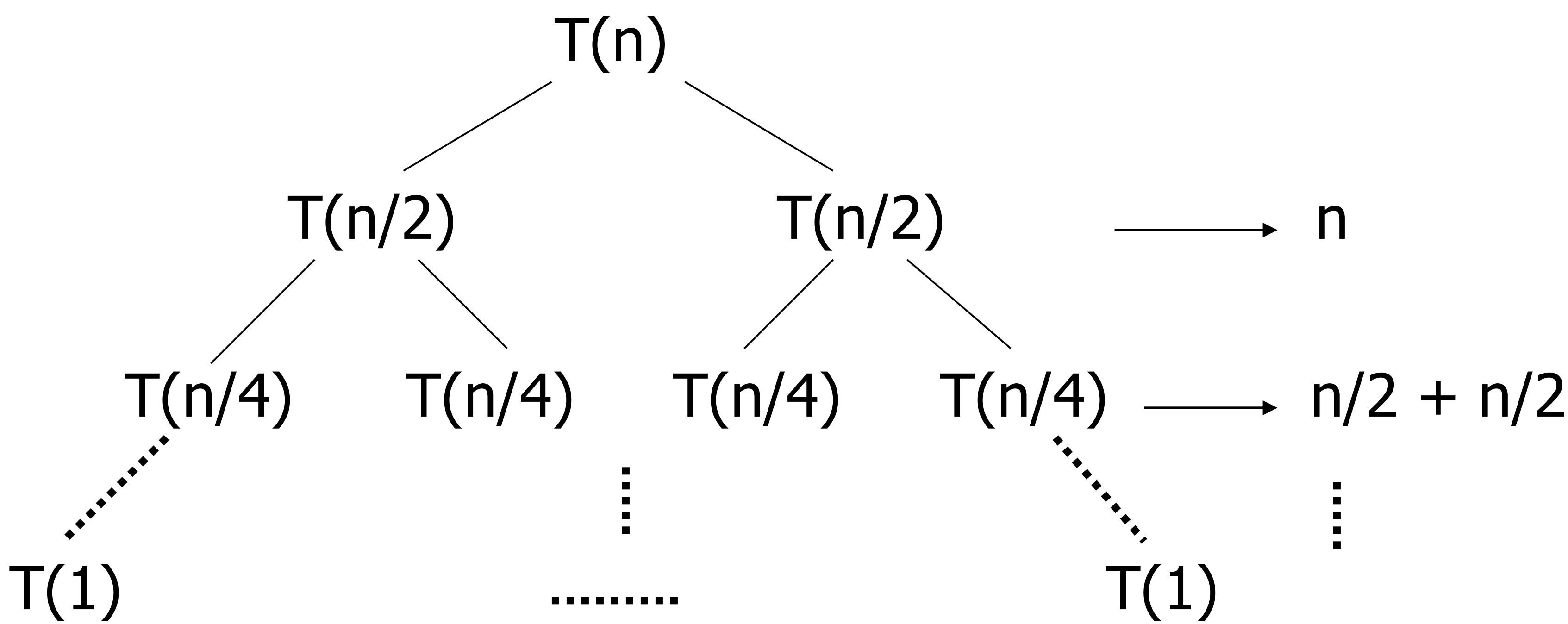
(Can omit base case if only need big-O)

$T(n/2) = 2T((n/2)/2)+(n/2)^2$

$$
\begin{aligned}
T(n) &= 2\ T(n/2) + n^2 \\
&= 2\ (\ 2\ T(n/4) + (n/2)^2\ ) + n^2 \\
&= 2^2\ T(n/2^2) + n^2/2 + n^2 \\
&= 2^2\ (2T(n/2^3) + (n/2^2)^2) + n^2/2 + n^2 \\
&= 2^3\ T(n/2^3) + n^2/2^2 + n^2/2 + n^2 \\
&= \ldots \\
&= 2^k\ T(n/2^k) + (1/2^{k-1} + \ldots + 1)n^2 \\
&= n\ T(1) + (1 - (1/2)^k)/(1 - 1/2)n^2 \\
&\qquad\qquad \text{when } n/2^k = 1\text{, i.e. } k = \log n \\
&= cn + 2(1 - 1/n)n^2 \\
&= 2n^2 + cn - 2n \\
&= O(n^2)
\end{aligned}
$$

Recursion Tree

- We can view the "unrolling" in the form of a tree
  - Example: $T(n) = 2T(n/2) + n$



Method 2: Induction

- If we can guess the solution, we can prove it by *induction*
  - Recall: What is mathematical induction?
- Example: $T(n) = 2\,T(n-1) + 1$, $T(1) = 1$ (Tower of Hanoi)
  - Guess: $T(n) = 2^n - 1$
  - Reason: $T(n)$ "roughly" doubles for every n, and $T(1) = 2^1 - 1 = 1$
  - Proof: base case n=1: trivial
  - Induction step:

$$\begin{aligned} T(n) &= 2\,T(n-1) + 1 \\ &= 2\,(2^{n-1} - 1) + 1 && \text{(induction hypothesis)} \\ &= 2^n - 2 + 1 \\ &= 2^n - 1 && \text{(done)} \end{aligned}$$

Induction: Wrong Use

- Be careful when using induction with Big-O!
- Example: T(n) = 2 T(n/2) + n
  - We "prove" that T(n) = O(n)
  - "Proof": base case is trivial. Induction step:

$$
\begin{aligned}
T(n) &= 2\,T(n/2) + n \\
&= 2\,O(n/2) + n \quad \text{(induction hypothesis)} \\
&= 2\,O(n) + n \quad (\,O(n/2) = O(n)\,) \\
&= O(n) + n \quad (\text{ big-O absorbs constants}) \\
&= O(n)
\end{aligned}
$$

  - Wrong because the constant hidden in big-O is no longer a constant

Wrong Use (cont'd)

- Try assuming $T(n) \le cn$ for some c:
  - $T(n) = 2T(n/2) + n \le 2(cn/2) + n = (c+1)n > cn$ for any c!
- Try $T(n) \le c\, n \log n$ for some c:

$$
\boxed{
\begin{aligned}
T(n) &= 2\,T(n/2) + n \\
&\le 2c(n/2)\log(n/2) + n && \text{(induction hypothesis)} \\
&= cn(\log n - 1) + n && (\log(n/2) = \log n - \log 2) \\
&= cn \log n - cn + n \\
&\le cn \log n && \text{for } c > 1
\end{aligned}
}
$$

  - Note: we use $\le$ because it is big-O

Method 3: Master Theorem

- Many recurrences are of the form

  $$T(n) = a\,T(n/b) + O(n^d)$$

  for some constants a, b, d
- Then
  - $$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } d < \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^d) & \text{if } d > \log_b a \end{cases}$$

Master Theorem: Examples

- $T(n) = 9\ T(n/3) + n$
  - $a = 9$, $b = 3$, $d = 1$, $\log_b a = \log_3 9 = 2$
  - Apply case 1: $T(n) = O(n^2)$
- $T(n) = 3\ T(n/3) + n$
  - $a = b = 3$, $d = 1$, $\log_b a = \log_3 3 = 1$
  - Apply case 2: $T(n) = O(n \log n)$
- $T(n) = 3\ T(n/3) + n^2$
  - $a = b = 3$, $d = 2$, $\log_b a = \log_3 3 = 1$
  - Apply case 3: $T(n) = O(n^2)$

Use of Master Theorem

- Still, you need to know elementary ways of solving recurrence
  - Not all recurrences can be solved by master theorem
  - It only gives big-O answers
  - You will be asked to solve "manually"

Floors and Ceilings

- In $T(n) = 2\ T(n/2) + n$: what if n is odd?
- Some notations:
  - *Floor* of n, $\lfloor n \rfloor$ = largest integer smaller than or equal to n
  - *Ceiling* of n, $\lceil n \rceil$ = smallest integer larger than or equal to n
  - E.g. $\lfloor 3.7 \rfloor = 3$, $\lceil 5.2 \rceil = 6$
- To be precise, it should be $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$
- We often omit these complications
  - E.g. assume n is power of 2 (always divisible)
  - Does not affect result