Lecture 1: Solving recurrences

The course presents techniques for solving problems algorithmically. We strive to design as efficient algorithms as possible. In order to evaluate an algorithm, i.e. to compare it to other algorithms solving the same problem, we need some measure of efficiency. In this course we will mainly analyze algorithms in terms of worst case time complexity.

Consider the problem of sorting $n$ elements stored in $A[1..n]$. A simple algorithm is Insertionsort.

```
INTRODUCTION-SORT(A, n)
if $n > 1$ then (insert $A[n]$ into sorted $A[1..n-1]$)
  JNTERVENTION-SORT(A, $n - 1$)
  key ← $A[n]$
  $i ← n - 1$
  while $i > 0$ and $A[i] > key$
    $A[i + 1] ← A[i]$
    $i ← i - 1$
  $A[i + 1] ← key$
```

Its time complexity depends on the size of input, $n$, and to what extent the input is presorted. The worst case happens when input elements are in descending order. Then the number of comparisons is $T(n) = T(n - 1) + n - 1 = \sum_{i=1}^{n-1} i = n(n - 1)/2 = \Theta(n^2)$.

Repeat definitions of $O$, $\Omega$, and $\Theta$ in the course book.

A more efficient algorithm is Mergesort, where two sorted lists are merged in linear time.

```
MERGE-SORT(A, $p$, $r$)
if $p < r$ then
  $q ← \lfloor (p + r)/2 \rfloor$
  MERGE-SORT(A, $p$, $q$)
  MERGE-SORT(A, $q + 1$, $r$)
  MERGE(A, $p$, $q$, $r$)
```

This gives the recurrence

$$T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
2T(n/2) + n - 1 & \text{if } n > 1 
\end{cases}$$

Its solution $\Theta(n \log n)$ grows slower than $\Theta(n^2)$, so Mergesort is asymptotically faster than Insertionsort.

Even faster in practice is Quicksort. It partitions the elements in two sets, one with elements less than a chosen pivot element and one with those greater, and then sorts these two sets recursively.

```
QUICKSORT(A, $p$, $r$)
if $p < r$ then
  $q ← \text{PARTITION}(A, p, r)$
  QUICKSORT(A, $p$, $q - 1$)
  QUICKSORT(A, $q + 1$, $r$)
```

A lucky partitioning produces two subproblems of equal size: $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$. If we are unlucky, all the elements (except the pivot) are placed in one of the two sets: $T(n) = T(0) + T(n - 1) + \Theta(n) = \Theta(n^2)$. This occurs when input is already sorted and the first element is chosen as pivot. However, Quicksort is $\Theta(n \log n)$ on the average.
Solving recurrences

To analyze the time complexity of a divide and conquer algorithm we express its work with a recurrence. For example, assume the algorithm divides a problem in four subproblems, each containing half of the elements; and where the split into subproblems + the combining of subsolutions takes linear time:

\[ T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
4T(n/2) + n & \text{if } n > 1 
\end{cases} \]

We want an asymptotic solution that shows how \( T(n) \) grows for sufficiently large \( n \).

Method 1. Substitution. Guess the form of the solution and verify it by induction using proper constants (often easy, if guess is correct). Applied on \( T(n) = 4T(n/2) + n \):

- Guess \( O(n^3) \): Assume \( T(k) \leq ck^3 \) for \( k < n \).
  Prove \( T(n) \leq cn^3 \) by induction.
  \( T(n) \leq 4c(n/2)^3 + n = (c/2)n^3 + n = cn^3 - ((c/2)n^3 - n) \leq cn^3 \), if \( c \geq 2 \), \( n \geq 1 \).
  Try to write expression as: (desired answer) \( - \) (something \( \geq 0 \)).
  To satisfy initial condition: choose \( c \) large enough.
- Not a sharp bound! We can prove \( O(n^2) \).
  Faulty argument for this: Assume \( T(k) \leq ck^2 \) for \( k < n \).
  \( T(n) \leq 4c(n/2)^2 + n = cn^2 + n = O(n^2) \) ?
  This is not according to the induction hypothesis: \( cn^2 + n \not\leq cn^2 \) for any \( c > 0 \).
  Instead subtract a lower order term: Assume \( T(k) \leq c_1k^2 - c_2k \) for \( k < n \).
  \( T(n) \leq 4(c_1(n/2)^2 - c_2(n/2)) + n = c_1n^2 - 2c_2n + n = c_1n^2 - c_2n - (c_2n - n) \leq c_1n^2 - c_2n \), if \( c_2 \geq 1 \).
  Choose \( c_1 \) big enough to take care of initial condition.

Note: We rarely need lower order terms in the induction proofs.

Similarly show a lower bound \( \Omega(n^2) \): Assume \( T(k) \geq c \cdot k^2 \) for \( k < n \). Prove \( T(n) \geq c \cdot n^2 \).

\[ T(n) \geq 4c \cdot (n/2)^2 + n = c \cdot n^2 + n \quad \text{[desired answer + something} \geq 0] \geq c \cdot n^2. \]

2. Recursion tree sums the costs on each level in the recursion. In our example:

\[
T(n) = n + 4T(n/2) \\
= n + 4(n/2 + 4T(n/2^2)) \\
= \cdots + \frac{4^{i-1}}{2^{i-1}}n + 4^i T(n/2^i) \\
= n + 2n + 4n + \cdots + 4^{\log_2 n} T(1) \\
= n \sum_{i=0}^{\log_2 n - 1} 2^i + \Theta(4^{\log n}) \\
= n(2^{\log_2 n} - 1)/(2 - 1) + \Theta(2^{\log n})^2, \text{ since } (2^2)^{\log n} = (2^{\log n})^2 \\
= \Theta(n^2) + \Theta(n^2) = \Theta(n^2) \]
3. Master method for $T(n) = aT(n/b) + f(n)$, when $n \geq b$, and $\Theta(1)$ otherwise.

Three cases compare $f(n)$ to $n^\log_b a$, here with a more general Case 2 than in the book. We let the cases be formulated (equivalently) as fractions instead of bounds on $f$.

<table>
<thead>
<tr>
<th>Case 1</th>
<th>$O(n^{-\epsilon})$, constant $\epsilon &gt; 0$</th>
<th>$\Theta(1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 2</td>
<td>$\Theta(\log^k n)$, constant $k \geq 0$</td>
<td>$\Theta(\log^k \log^{k+1} n)$</td>
</tr>
<tr>
<td>Case 3</td>
<td>$\Omega(n^\epsilon)$, constant $\epsilon &gt; 0$ and $a f(n/b) &lt; f(n)$</td>
<td>$\Theta(f(n))$</td>
</tr>
</tbody>
</table>

In our example: $a = 4$, $b = 2$, $f(n) = n$ give $n/n^\log_2 4 = 1/n \Rightarrow$ Case 1: $T(n) = \Theta(n^2)$.

Note that the three cases are not complete: the master method cannot solve all recurrences of the given form with $a \geq 1$, $b > 1$, and $f$ asymptotically positive.

4. Akra-Bazzi (optional)

More general than the master method for solving recurrences:

$$T(x) = \begin{cases} 
\Theta(1) & \text{if } 1 \leq x \leq x_0 \\
\Theta(x^{p}) + \sum_{i=1}^{k} a_i T(b_i x) & \text{if } x > x_0 
\end{cases}$$

where

- $x \geq 1$ is a real number and $k \geq 1$ is a constant
- constant $x_0$, the break point of recursion, such that $x_0 \geq 1/b_i$ and $x_0 \geq 1/(1 - b_i)$ for $1 \leq i \leq k$
- $a_i > 0$ and $b_i \in (0, 1)$ are constants for $1 \leq i \leq k$
- $g(x)$ grows polynomially: there are positive constants $c_1, c_2$ such that for all $x \geq 1$, $1 \leq i \leq k$, and $u \in [b_i x, x]$, $c_1 g(x) \leq g(u) \leq c_2 g(x)$.

Let $p$ be the unique real number for which $\sum_{i=1}^{k} a_i b_i^p = 1$. Then

$$T(x) = \Theta(x^p (1 + \int_{1}^{x} \frac{g(u)}{u^{p+1}} du))$$

Justification for this is given in an article available from the course website.

Our example: $T(n) = 4T(n/2) + n$, gives $4(1/2)^p = 1 \Rightarrow p = 2$, and since $\int_1^n u/u^3 \, du = \int_1^n 1/u^2 \, du = 1 - 1/n$, we get $T(n) = \Theta(n^2(2 - 1/n)) = \Theta(n^2)$.

Summary: We show how the following three recurrences can be solved by the above methods:

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

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

$T(n) = T(n/2) + T(n/4) + n$