Lecture 5: Master Method and Quick-Sort
(CLRS 4.3-4.4 (read this note instead), 7.1-7.2)

May 22nd, 2002

1 Master Method (recurrences)

- We have solved several recurrences using substitution and iteration.
- Last time we solved several recurrences of the form $T(n) = aT(n/b) + n^c$ ($T(1) = 1$).
  - Strassen’s algorithm $T(n) = 7T(n/2) + n^2$ ($a = 7, b = 2, c = 2$)
  - Merge-sort $T(n) = 2T(n/2) + n$ ($a = 2, b = 2, c = 1$).
- It would be nice to have a general solution to the recurrence $T(n) = aT(n/b) + n^c$.
- We do!

\[
T(n) = aT\left(\frac{n}{b}\right) + n^c \quad a \geq 1, b \geq 1, c > 0
\]
\[
\begin{cases}
\Theta(n^{\log_b a}) & a > b^c \\
\Theta(n^c \log_b n) & a = b^c \\
\Theta(n^c) & a < b^c
\end{cases}
\]

Proof (Iteration method)

\[
T(n) = aT\left(\frac{n}{b}\right) + n^c \\
= n^c + a\left(\left(\frac{n}{b}\right)^c + aT\left(\frac{n}{b^2}\right)\right) \\
= n^c + \left(\frac{a}{b^c}\right)n^c + a^2T\left(\frac{n}{b^2}\right) \\
= n^c + \left(\frac{a}{b^c}\right)n^c + a^2\left(\left(\frac{n}{b^2}\right)^c + aT\left(\frac{n}{b^4}\right)\right) \\
= n^c + \left(\frac{a}{b^c}\right)n^c + a^2\left(\left(\frac{n}{b^2}\right)^c + aT\left(\frac{n}{b^4}\right)\right) \\
= \cdots \\
= n^c + \left(\frac{a}{b^c}\right)n^c + \left(\frac{a}{b^c}\right)^2n^c + \left(\frac{a}{b^c}\right)^3n^c + \cdots + \left(\frac{a}{b^c}\right)^{\log_b n - 1}n^c + a^{\log_b n}T(1) \\
= n^c \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^k + a^{\log_b n} \\
= n^c \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^k + n^{\log_b a}
\]

Recall geometric sum $\sum_{k=0}^{n-1} x^k = \frac{x^{n+1} - 1}{x-1} = \Theta(x^n)$

- $a < b^c$ $\iff$ $\frac{a}{b^c} < 1$ $\implies$ $\sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^k \leq \sum_{k=0}^{+\infty} \left(\frac{a}{b^c}\right)^k = \frac{1}{1-(\frac{a}{b^c})} = \Theta(1)$
- $a < b^c$ $\iff$ $\log_b a < \log_b b^c = c$
- $T(n) = n^c \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^k + n^{\log_b a} \\
= n^c \cdot \Theta(1) + n^{\log_b a} \\
= \Theta(n^c)$
\[ a = b^c \]

\[ a = b^c \iff \frac{a}{b} > 1 \Rightarrow \sum_{k=0}^{\log_b n} \left( \frac{a}{b} \right)^k = \sum_{k=0}^{\log_b n-1} 1 = \Theta(\log_b n) \]

\[ a = b^c \iff \log_b a = \log_b b^c = c \]

\[ T(n) = \sum_{k=0}^{\log_b n} \left( \frac{a}{b} \right)^k + n^{\log_b a} = n^c \Theta(\log_b n) + n^{\log_b a} = \Theta(n^c \log_b n) \]

\[ a > b^c \iff \frac{a}{b^c} > 1 \Rightarrow \sum_{k=0}^{\log_b n} \left( \frac{a}{b^c} \right)^k = \Theta \left( \left( \frac{a}{b^c} \right)^{\log_b n} \right) = \Theta \left( \frac{a^{\log_b n}}{n^c} \right) \]

\[ T(n) = n^c \cdot \Theta \left( \frac{a^{\log_b n}}{n^c} \right) + n^{\log_b a} = \Theta \left( n^{\log_b a} \right) + n^{\log_b a} = \Theta \left( n^{\log_b a} \right) \]

- Note: Book states and proves the result slightly differently (don’t read it).

### 1.1 Other recurrences

Some important/typical bounds on recurrences not covered by master method:

- **Logarithmic:** \( \Theta(\log n) \)
  - Recurrence: \( T(n) = 1 + T(n/2) \)
  - Typical example: Recurse on half the input (and throw half away)
  - Variations: \( T(n) = 1 + T(99n/100) \)

- **Linear:** \( \Theta(N) \)
  - Recurrence: \( T(n) = 1 + T(n-1) \)
  - Typical example: Single loop
  - Variations: \( T(n) = 1 + 2T(n/2), T(n) = n + T(n/2), T(n) = T(n/5) + T(7n/10 + 6) + n \)

- **Quadratic:** \( \Theta(n^2) \)
  - Recurrence: \( T(n) = n + T(n-1) \)
  - Typical example: Nested loops

- **Exponential:** \( \Theta(2^n) \)
  - Recurrence: \( T(n) = 2T(n-1) \)

### 2 Quick-sort

- We previously saw how divide-and-conquer can be used to design sorting algorithm—Merge-sort
  - Partition \( n \) elements array \( A \) into two subarrays of \( n/2 \) elements each
  - Sort the two subarrays recursively
  - Merge the two subarrays

Running time: \( T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n) \)
Another possibility is to use the “opposite” version of divide-and-conquer—Quick-sort

- Partition \( A[1...n] \) into subarrays \( A' = A[1..q] \) and \( A'' = A[q+1...n] \) such that all elements in \( A'' \) are larger than all elements in \( A' \).
- Recursively sort \( A' \) and \( A'' \).
- (nothing to combine/merge. \( A \) already sorted after sorting \( A' \) and \( A'' \))

If \( q = n/2 \) and we divide in \( \Theta(n) \) time, we again get the recurrence \( T(n) = 2T(n/2) + \Theta(n) \) for the running time \( \Rightarrow T(n) = \Theta(n \log n) \)

The problem is that it is hard to develop partition algorithm which always divide \( A \) in two halves

- Pseudo code for Quick-sort:

```
QUICKSORT(A, p, r)
IF p < r THEN
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
FI
```

Sort using \( \text{QUICKSORT}(A, 1, n) \)

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
FOR j = p TO r - 1 DO
    IF A[j] \leq x THEN
        i = i + 1
        Exchange A[i] and A[j]
    FI
OD
Exchange A[i + 1] and A[r]
RETURN i + 1
```

- \( \text{Partition} \) runs in time \( \Theta(n) \)
• Correctness:
  – Clear if \textsc{Partition} divides correctly
  – Example:

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
2 & 8 & 7 & 1 & 3 & 5 & 6 & 4 \\
2 & 8 & 7 & 1 & 3 & 5 & 6 & 4 \\
2 & 8 & 7 & 1 & 3 & 5 & 6 & 4 \\
2 & 8 & 7 & 1 & 3 & 5 & 6 & 4 \\
2 & 1 & 7 & 8 & 3 & 5 & 6 & 4 \\
2 & 1 & 3 & 8 & 7 & 5 & 6 & 4 \\
2 & 1 & 3 & 8 & 7 & 5 & 6 & 4 \\
2 & 1 & 3 & 8 & 7 & 5 & 6 & 4 \\
\hline
i=0, j=1 & i=1, j=2 & i=1, j=3 & i=1, j=4 & i=2, j=5 & i=3, j=6 & i=3, j=7 & i=3, j=8 & q=4
\end{array}
\]

– \textsc{Partition} can be proved correct (by induction) using the loop invariant:
  * \( A[k] \leq x \) for \( p \leq k \leq i \)
  * \( A[k] > x \) for \( i + 1 \leq k \leq j - 1 \)
  * \( A[k] = x \) for \( k = r \)

• Running time depends on how well \textsc{Partition} divides \( A \).
  – In the example it does reasonably well.
  – In the worst case \( q \) is always \( p \) and the running time becomes \( T(n) = \Theta(n) + T(1) + T(n-1) \Rightarrow T(n) = \Theta(n^2) \).

  * and what is maybe even worse, the worst case is when \( A \) is already sorted.

• So why is it called "quick"-sort? Because it "often" performs very well—can we theoretically justify this?

  – Even if all the splits are relatively bad, we get \( \Theta(n \log n) \) time:

   * Example: Split is \( \frac{9}{10}n, \frac{1}{10}n \).

   \[
   T(n) = T\left(\frac{9}{10}n\right) + T\left(\frac{1}{10}n\right) + n
   \]

   Solution?
   Guess: \( T(n) \leq cn \log n \)

Induction

\[
T(n) = T\left(\frac{9}{10}n\right) + T\left(\frac{1}{10}n\right) + n
\]

\[
\leq \frac{9cn}{10} \log\left(\frac{9n}{10}\right) + \frac{cn}{10} \log\left(\frac{n}{10}\right) + n
\]

\[
\leq \frac{9cn}{10} \log n + \frac{9cn}{10} \log\left(\frac{9}{10}\right) + \frac{cn}{10} \log n + \frac{cn}{10} \log\left(\frac{1}{10}\right) + n
\]

\[
\leq cn \log n + \frac{9cn}{10} \log 9 - \frac{9cn}{10} \log 10 - \frac{cn}{10} \log 10 + n
\]

\[
\leq cn \log n - n(c \log 10 - \frac{9c}{10} \log 9 - 1)
\]

\[T(n) \leq cn \log n \text{ if } c \log 10 - \frac{9c}{10} \log 9 - 1 > 0 \text{ which is definitely true if } c > \frac{10}{\log 10}\]
– So, in other words, if just the splits happen at a constant fraction of $n$ we get $\Theta(n \lg n)$—or, its almost never bad!

• Next time we will further justify the good practical performance by looking at average case running time.