Recurrences
Divide and Conquer Algorithms

- Insertion Sort – *incremental*
- Sort the sub-array from $A[1...j-1]$ and insert $A[j]$ in its right position
- Divide and Conquer – *recursive* structure
- *Key idea*: Solve sub-problems recursively, then combine these solutions.
MERGE-SORT(A, p, r)

if $p < r$  // check for base case

\[ q = \lfloor (p + r)/2 \rfloor \]  // divide

MERGE-SORT(A, p, q)  // conquer

MERGE-SORT(A, q + 1, r)  // conquer

MERGE(A, p, q, r)  // combine
Divide-and-Conquer

**Divide** the problem into a number of subproblems.

**Conquer** the subproblems by solving them recursively. When the problem is small enough, solving it becomes straightforward.

**Combine** the solutions to the subproblems.
Merge Sort

• Example of divide and conquer

**Divide** the $n$-element sequence into $n/2$ elements

**Conquer** the subproblems by recursively using merge sort

**Combine** the two sorted subsequences to produce the sorted
Merge Sort (contd.)

• Recursion “bottoms out” when the sequence is of length 1.

• Key operation – **Combine** step

• We perform this using the procedure
  MERGE\((A,p,q,r)\)

• MERGE –

**INPUT**: \(A\) is an array, \(p \leq q < r\) are indices such that \(A[p..q]\) and \(A[q+1,r]\) are sorted

**OUTPUT**: Sorted merged array \(A[p,r]\)
How to Merge??

• 2 sorted piles of cards face up on the table
• Remove the smaller cards and place face down in the output pile
• Repeat until input piles are empty
• How long does this take??
\textbf{Merge}(A, p, q, r)

\[ n_1 = q - p + 1 \]
\[ n_2 = r - q \]
let \( L[1..n_1 + 1] \) and \( R[1..n_2 + 1] \) be new arrays

\textbf{for} \( i = 1 \) to \( n_1 \)
\hspace{1cm} \( L[i] = A[p + i - 1] \)
\textbf{for} \( j = 1 \) to \( n_2 \)
\hspace{1cm} \( R[j] = A[q + j] \)
\( L[n_1 + 1] = \infty \)
\( R[n_2 + 1] = \infty \)
\( i = 1 \)
\( j = 1 \)
\textbf{for} \( k = p \) to \( r \)
\hspace{1cm} \textbf{if} \( L[i] \leq R[j] \)
\hspace{1.5cm} \( A[k] = L[i] \)
\hspace{1.5cm} \( i = i + 1 \)
\hspace{1cm} \textbf{else} \( A[k] = R[j] \)
\hspace{1.5cm} \( j = j + 1 \)
Merging

- $n = r - p + 1$ time
- $\Theta(n)$ time
Figure 2.3 The operation of lines 10–17 in the call MERGE(A, 9, 12, 16), when the subarray A[9..16] contains the sequence (2, 4, 5, 7, 1, 2, 3, 6). After copying and inserting sentinels, the array \( L \) contains \( \{2, 4, 5, 7, \infty\} \), and the array \( R \) contains \( \{1, 2, 3, 6, \infty\} \). Lightly shaded positions in \( A \) contain their final values, and lightly shaded positions in \( L \) and \( R \) contain values that have yet to be copied back into \( A \). Taken together, the lightly shaded positions always comprise the values originally in \( A[9..16] \), along with the two sentinels. Heavily shaded positions in \( A \) contain values that will be copied over, and heavily shaded positions in \( L \) and \( R \) contain values that have already been copied back into \( A \). (a)-(h) The arrays \( A, L, \) and \( R \), and their respective indices \( k, i, \) and \( j \) prior to each iteration of the loop of lines 12–17. (i) The arrays and indices at termination. At this point, the subarray in \( A[9..16] \) is sorted, and the two sentinels in \( L \) and \( R \) are the only two elements in these arrays that have not been copied into \( A \).
Figure 2.4  The operation of merge sort on the array $A = (5, 2, 4, 7, 1, 3, 2, 6)$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.
Analyzing Divide-and-conquer algorithms

• So how do we analyze a recursive algorithm?
• **Recurrence Relation** – equation describing the running time of a problem of size $n$ in terms of running time of smaller inputs

• If we divide the original program into $a$ subproblems each of which is $1/b$ the size of original,

\[
T_n = \begin{cases} 
\Theta(1) & \text{if } n \leq c \\
a T(n/b) + D(n) + C(n) & \text{otherwise}
\end{cases}
\]
Analyzing Merge Sort

• Simplify analysis – assume even number of elements
• Merge sort on 1 element – constant time
• For n>1 element:
  – **Divide:** Dividing down the middle takes const time
    \[ D(n) = \theta(1) \]
  – **Conquer:** Two subproblems each of size n/2 take
    \[ 2T(n/2) \]
  – **Combine:** The merge procedure takes \( \theta(n) \)
Analyzing Merge Sort

Divide-and-Conquer

\[ T_n = \begin{cases} 
\Theta(1) & \text{if } n \leq c \\
 a \ T(n/b) + D(n) + C(n) & \text{otherwise}
\end{cases} \]

Merge Sort

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

• D(n) is \( \theta(1) \) and C(n) is \( \theta(n) \) for Merge Sort
• How do we evaluate such an expression??

• \( T_n = \theta(n \ lg n) \)

• We will look at a formal method (Master Theorem)
• But let's try and reason this
Reasoning $\theta(n \lg n)$

• Let’s rewrite

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

• As

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

• How do we solve this?
Figure 2.5 The construction of a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part (a) shows $T(n)$, which is progressively expanded in (b)–(d) to form the recursion tree. The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated), and each level contributes a total cost of $cn$. The total cost, therefore, is $cn \lg n + cn$, which is $\Theta(n \lg n)$. 
Reasoning $\Theta(n \lg n)$

• Each level has a total cost $cn$
• In general, level $i$ has $2^i$ nodes each contributing a cost of $cn/2^i$ for a total cost of $cn$
• How many levels does this tree have?
  • Levels = $\lg n$
• Total cost = $cn \lg n$
How fast?

• So how does $\theta(n \lg n)$ for merge sort compare to $\theta(n^2)$ for insertion sort?

• Way better for large $n$!
Solving Recurrences

• The approach we just took – *Recursion Tree*
• Sum the costs within each level to obtain a set of per-level costs.
• Sum these to get total cost
Another example

- \( T(n) = T(n/3) + T(2n/3) + cn \)
Another Example

Figure 4.2  A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$. 

Total: $O(n \log n)$
Master Method

• “Cookbook” for solving recurrences of the form:
  \[ T(n) = a \times T(n/b) + f(n) \]
Where,

a ≥ 1 and b > 1 and f(n) is an asymptotically positive function
Master Theorem

\[ T(n) = a \, T\left(\frac{n}{b}\right) + f(n) \quad \text{where} \quad a \geq 1, \, b > 1. \]

Case 1

\[ f(n) = \mathcal{O} \left( n^{\log_b(a) - \varepsilon} \right) \quad \text{then} \quad T(n) = \Theta \left( n^{\log_b a} \right). \]

Case 2

\[ f(n) = \Theta \left( n^{\log_b a \log^k n} \right) \quad \text{then} \quad T(n) = \Theta \left( n^{\log_b a \log^{k+1} n} \right). \]

Case 3

\[ f(n) = \Omega \left( n^{\log_b a + \varepsilon} \right) \quad \& \quad a f \left( \frac{n}{b} \right) \leq c f(n) \quad \text{then} \quad T(n) = \Theta \left( f(n) \right). \]

For all cases, \( \varepsilon > 0 \) and \( c < 1 \).
Examples of the Master Method

Example

\[ T(n) = 8T \left( \frac{n}{2} \right) + 1000n^2 \]

As one can see in the formula above, the variables get the following values:

\[ a = 8, b = 2, f(n) = 1000n^2, \log_b a = \log_2 8 = 3 \]

Now we have to check that the following equation holds:

\[ f(n) = \mathcal{O} \left( n^{\log_b a - \varepsilon} \right) \]
\[ 1000n^2 = \mathcal{O} \left( n^{3 - \varepsilon} \right) \]

If we choose \( \varepsilon = 1 \), we get:

\[ 1000n^2 = \mathcal{O} \left( n^{3-1} \right) = \mathcal{O} \left( n^2 \right) \]

Since this equation holds, the first case of the master theorem applies to the given recurrence relation, thus resulting in the conclusion:

\[ T(n) = \Theta \left( n^{\log_b a} \right) . \]

If we insert the values from above, we finally get:

\[ T(n) = \Theta \left( n^3 \right) . \]

Thus the given recurrence relation \( T(n) \) was in \( \Theta(n^3) \).

(This result is confirmed by the exact solution of the recurrence relation, which is \( T(n) = 1001n^3 - 1000n^2 \), assuming \( T(1) = 1 \)).
Examples of the Master Method

Example

\[ T(n) = 2T \left( \frac{n}{2} \right) + 10n \]

As we can see in the formula above the variables get the following values:

\[ a = 2, b = 2, k = 0, f(n) = 10n, \log_b a = \log_2 2 = 1 \]

Now we have to check that the following equation holds (in this case k=0):

\[ f(n) = \Theta \left( n^{\log_b a} \right) \]

If we insert the values from above, we get:

\[ 10n = \Theta \left( n^1 \right) = \Theta \left( n \right) \]

Since this equation holds, the second case of the master theorem applies to the given recurrence relation, thus resulting in the conclusion:

\[ T(n) = \Theta \left( n^{\log_b a} \log n \right). \]

If we insert the values from above, we finally get:

\[ T(n) = \Theta \left( n \log n \right). \]

Thus the given recurrence relation \( T(n) \) was in \( \Theta(n \log n) \).
Examples of the Master Method

Example

\[ T(n) = 2T\left(\frac{n}{2}\right) + n^2 \]

As we can see in the formula above the variables get the following values:

\[ a = 2, \ b = 2, \ f(n) = n^2, \ \log_b a = \log_2 2 = 1 \]

Now we have to check that the following equation holds:

\[ f(n) = \Omega\left(n^{\log_b a + \epsilon}\right) \]

If we insert the values from above, and choose \( \epsilon = 1 \), we get:

\[ n^2 = \Omega\left(n^{1+1}\right) = \Omega\left(n^2\right) \]

Since this equation holds, we have to check the second condition, namely if it is true that:

\[ af\left(\frac{n}{b}\right) \leq cf(n) \]

If we insert once more the values from above, we get:

\[ 2\left(\frac{n}{2}\right)^2 \leq cn^2 \iff \frac{1}{2}n^2 \leq cn^2 \]

If we choose \( c = \frac{1}{2} \), it is true that:

\[ \frac{1}{2}n^2 \leq \frac{1}{2}n^2 \forall n \geq 1 \]

So it follows:

\[ T(n) = \Theta(f(n)) \]

If we insert once more the necessary values, we get:

\[ T(n) = \Theta(n^2) \]

Thus the given recurrence relation \( T(n) \) was in \( \Theta(n^2) \), that complies with the \( f(n) \) of the original formula.