divide-and-conquer algorithm to sort a list of numbers:

\[\text{procedure mergesort}(L);\]
\[\text{if } |L| = 1 \text{ then return } L \quad \text{ /* base case */}\]
\[\text{else } \{\]
\[L_1 \leftarrow \text{any } \lfloor |L|/2 \rfloor \text{ elements of } L; \quad \text{ /* divide step */}\]
\[L_2 \leftarrow \text{the remaining } \lceil |L|/2 \rceil \text{ elements of } L;\]
\[S_1 \leftarrow \text{mergesort}(L_1); \quad \text{ /* recurse */}\]
\[S_2 \leftarrow \text{mergesort}(L_2);\]
\[\text{merge } S_1 \text{ with } S_2 \text{ and return the result; } \}\quad \text{ /* combine */}\]

note that any integer \( n \) satisfies \( n = \lfloor n/2 \rfloor + \lceil n/2 \rceil \)

**Example 1.**

input: 1 12 8 10 4 7 2 9 → recursively sort: 1 8 10 12 2 4 7 9 → merge: 1 2 4 7 8 9 10 12

Example 1 illustrates the 1st of 2 good ways to visualize recursive algorithms:

*The Magic View of Recursion:*

think of recursive calls as “magically” returning the correct answer
do not worry about the details of lower levels of recursion!

**Theorem.** Mergesort sorts a list of \( n \) numbers in time \( O(n \log n) \) and space \( O(n) \).

we’ll prove this twice (in this handout and next) illustrating 2 basic techniques

**Simple analysis/Iteration method**

define \( T(n) = \) the worst-case time to execute \( \text{mergesort} \) on a list of \( n \) elements

proceed in 2 steps (\( i \)) – (\( ii \)):

\( i \) assume \( n = 2^k \) for an integer \( k \)

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

**Remark.** this recurrence is well-defined, and avoids floors and ceilings

iterate the recurrence:

\[
T(n) = n + 2T(n/2) \\
= n + 2(n/2) + 4T(n/4) \\
= n + 2(n/2) + 4(n/4) + 8T(n/8) \\
= n + 2(n/2) + 4(n/4) + \ldots + 2^{i-1}(n/2^{i-1}) + 2^iT(n/2^i) \\
= n + 2(n/2) + \ldots + 2^i(n/2^i) + \ldots + 2^{k-1}(n/2^{k-1}) + 2^k \\
= n(1+k) \\
= n(1 + \log n)
\]
thus \( T(n) = O(n \log n) \), for \( n \) a power of 2

this calculation is the iteration method

(ii) let \( n \) be arbitrary

\textit{Fact.} There is a power of 2 between \( n \) and \( 2n \) (specifically \( p = 2^\lceil \log n \rceil \)).

for the above power \( p \),
\[
T(n) \leq T(p) = p(1 + \log p) \leq 2n(1 + \log 2n) \implies \text{in general, } T(n) = O(n \log n) \quad \square
\]

\textit{Remarks}

1. the recurrence for general \( n \) is too messy to analyze
2. usually we omit step (ii)! (see CLRS 4.4.2)
3. the “simple analysis” often corresponds to a “simple algorithm”
   the algorithm makes \( n \) a power of 2 by padding with dummy numbers (e.g., see FFT)
4. divide-and-conquer algorithms recurring on 2 equal-sized problems are the most common
F Master Theorem

the F Master Theorem generalizes our timing calculation to any number of equal-sized problems
it solves recurrences by inspection!
here’s a summary; see Handout #49 for details

consider a recurrence
\[ T(n) = \begin{cases} 
1 & n = 1 \\
 aT(n/b) + D(n) & n > 1, \ n \text{ a power of } b 
\end{cases} \]

where \( a, b \) are real numbers, \( a > 0, \ b > 1 \)
\( D(n) \) is called the “driving function”

the “homogeneous solution” (h.s.) is \( n^h \) for \( h = \log_b a \)

intuitively “\( T(n) = \max \{ \text{homogeneous solution, driver} \} \)”
more precisely:

(i) if \( D(n) = O(n^d) \) with \( d < h \) then \( T(n) = \Theta(n^h) \)

if (i) doesn’t apply suppose \( D(n) = n^d f(n) \) where \( d \geq 0 \ & \ f \) is a nondecreasing function
(intuitively \( f \) is a small function like \( \log n \), but that’s not required)

(ii) if \( d > h \) then \( T(n) = \Theta(D(n)) \)

(iii) if \( d = h \) then \( T(n) = \Theta(D(n) \log n) \) if \( f(n) \) is a small function like any power of \( \log n \)
more precisely if \( f(n) \) satisfies this “flatness condition”:
(F) \( \exists c > 0 \ \exists f(\sqrt{n}) \geq cf(n) \)

Example.

(i) \( T(n) = 8T(n/2) + n^2 \implies T(n) = \Theta(n^3) \)
(ii) \( T(n) = 2T(n/2) + n^2 \implies T(n) = \Theta(n^3) \)
(iii) \( T(n) = 4T(n/2) + n^2 \implies T(n) = \Theta(n^2 \log n) \)

Question. How do the answers change when the driver increases to \( n^2 \log n \)?
1. Divide-and-conquer recurrences

suppose a divide-and-conquer algorithm divides the given problem into equal-sized subproblems say a subproblems, each of size \( n/b \)

\[
T(n) = \begin{cases} 
1 & n = 1 \\
aT(n/b) + D(n) & n > 1, n \text{ a power of } b
\end{cases}
\]

the driving function

assume a and b are real numbers, \( a > 0, b > 1 \)

Remarks
1. usually \( a \) is integral!
2. fractional \( b \) is useful, e.g., \( T(n) = 3T(2n/3) + 1 \)
   here \( T \) is defined on a set of rational numbers, \( (3/2)^i \)
   the related function on integers, \( T(n) = 3T([2n/3]) + 1 \),
   behaves exactly the same way – CLRS 4.4.2

2. Solving the recurrence

let \( n = b^k \), \( k = \log b n \) (\( n \) not necessarily integer)

iterate the recurrence:

\[
T(b^k) = D(b^k) + aT(b^{k-1})
= D(b^k) + aD(b^{k-1}) + a^2T(b^{k-2})
= \sum_{i=0}^{k-1} a^iD(b^{k-i}) + a^kT(1)
\]

second term \( a^kT(1) \) is the solution when \( D(\cdot) = 0 \), called the homogeneous solution (h.s.)

\[
a^kT(1) = a^{\log b n} = n^{\log a}
\]

let \( h = \log b a \), so h.s. = \( n^h \)

usually \( h \geq 0 \) since \( a \geq 1 \)

An important special case

a common driving function is \( D(n) = n^d, d \geq 0 \) (\( d \) is real)
the sum becomes \( n^d \sum_{i=0}^{k-1} (a/b^d)^i \), a geometric progression

Sum of a geometric progression

let \( r \) be a constant and \( k \) tend to \( \infty \)

\[
\sum_{i=0}^{k} r^i = \begin{cases} 
\frac{r^{k+1} - 1}{r - 1} & r \neq 1 \\
k + 1 & r = 1
\end{cases} = \begin{cases} 
\Theta(1) & 0 < r < 1 \\
\Theta(k) & r = 1 \\
\Theta(r^k) & r > 1
\end{cases}
\]
for \( D(n) = n^d \), \( T(n) = \begin{cases} 
 \Theta(n^d) & a < b^d, \ i.e., \ h < d \\
 \Theta(n^h \log n) & a = b^d, \ i.e., \ h = d \\
 \Theta(n^h) & a > b^d, \ i.e., \ h > d 
\end{cases} \)

More generally
it’s fairly common to have drivers like \( n \log n \) or even \( n^2 \log n \log \log n \), etc.
we’ll assume our driver has the form \( n^d f(n) \), where \( f \) is nondecreasing
intuitively \( f \) is a small function like \( \log n \)

**F Master Theorem.** For any nondecreasing function \( f(n) \) and any \( d \geq 0 \),
\[
T(n) = \begin{cases} 
\Theta(D(n)) & D(n) = \Theta(n^d f(n)) \ h < d \\
O(D(n) \log n) & D(n) = \Theta(n^h f(n)) \\
\Theta(n^h) & D(n) = O(n^d) \ h > d 
\end{cases}
\]

Remarks
1. informally, “\( T(n) = \max \{ \text{homogeneous solution, driver} \} \)”
2. F Master Theorem is proved similar to special case above
3. the middle case is tight, i.e., \( T(n) = \Theta(D(n) \log n) \) for \( D(n) = \Theta(n^h f(n)) \),
   if \( f(n) \) satisfies this “flatness condition”:
   (F) \( f(\sqrt{n}) = \Omega(f(n)) \)
e.g., \( f(n) = \log n \) satisfies (F), \( f(n) = n \) doesn’t
the set of \( f \)'s satisfying (F) is closed under product, powers, logs
e.g., \( \log^2 n, \sqrt{\log n}, \log \log n \) satisfy (F)
we can also relax (F), requiring it only for sufficiently large \( n \)
4. the CLRS Master Theorem (p.73) has weaker 2nd & 3rd cases

3. Examples
1. \( T(n) = 3T(2n/3) + 1 \) (Stooge-sort, Pr.7-3)
h.s. : \( T(n) = 3T(2n/3) \); iterating gives h.s. = \( n^h \), \( h = \log_{3/2} 3 \approx 2.7 \)
\( h > d (\log_{3/2} 3 > 0) \implies T(n) = h.s. = \Theta(n^h) = \omega(n^2) \) (!)
2. \( T(n) = T(n/2^d) + d^2 n^{1/d} \) (recursion on \( d \)-dimensional mesh)
h.s. : \( T(n) = T(n/2^d) \); h.s. = 1
\( h < d (0 < 1/d) \implies T(n) = \text{driver} = \Theta(d^2 n^{1/d}) \)
this illustrates the case \( h = 0 \) when \( a = 1 \)
3. \( T(n) = T(n/2) + \log n \) (PRAM mergesort)
h.s. = 1, driver = (h.s.) \times \log n
\implies T(n) = \text{driver} \times \log n = \Theta(\log^2 n) \)
Master Theorem for Unequal-size Subproblems

**Example 1.** $T(n) = n + T(n/2)$  

**Example 2.** $T(n) = n + 2T(n/2)$

in general for $T(n) = n + aT(n/b)$,

$$T(n) = \begin{cases} 
\Theta(n) & a < b, \text{ i.e., total problem size decreases each level} \\
\Theta(n \log n) & a = b, \text{ i.e., total problem size stays same each level}
\end{cases}$$

this generalizes to unequal-size subproblems:

**Example 3.** $T(n) = n + T(n/2) + T(n/3)$

**Example 4.** $T(n) = n + T(n/2) + T(n/3) + T(n/6)$

**Example 5.** $T(n) = n + T(n/2) + 2T(n/4)$

**Example 6.** $T(n) = \Theta(n) + T([n/2]) + T([n/3]) + T([n/6] + 21)$

**Example 7.** $T(n) = n + T(n-1)$

**Theorem.** For real numbers $a_i, A_i, 0 < a_i < 1, i = 1, \ldots, k$ let

$$T(n) = \begin{cases} 
c & n < N \\
n + \sum_{i=1}^{k} T([a_i n + A_i]) & n \geq N
\end{cases}$$

Then

$$T(n) = \begin{cases} 
\Theta(n) & \sum_{i=1}^{k} a_i < 1 \\
\Theta(n \log n) & \sum_{i=1}^{k} a_i = 1
\end{cases}$$

**Proof.** use method of substitution (CLRS 4.1):

we guess the answer using intuition from equal-size subproblems

(CLRS p.191) illustrates 1st case  

**Remarks**

1. other ways to guess the solution:
   (i) make a table of values (perhaps computer-generated)
   (ii) guess form of solution, introducing unknown constants
   the inductive proof reveals the values of the constants
   see CLRS p.65

2. sometimes subproblem sizes can vary
e.g., **Planar Separator Theorem.** Any planar graph has a set of \( \leq \sqrt{8n} \) vertices whose removal leaves 2 disconnected subgraphs, each with \( \leq 2n/3 \) vertices. The separating set can be found in time \( O(n) \).

corresponding recurrence involves a \( \max \) operation, e.g.,

\[
T(n) \leq \max\{n + T(n_1) + T(n_2) : n_1 + n_2 \leq n; \ n_1, n_2 \leq 2n/3\}
\]

theorem holds for these recurrences too!

**Example 8.** \( T(n) = \max\{n + T(n_1) + T(n_2) : n_1 + n_2 \leq n; \ n_1, n_2 \leq 9n/10\} \quad T(n) = \underline{\ldots} \)

**F Master Theorem for Unequal Subproblems.**

Consider any recurrence

\[
T(n) = \sum_{i=1}^{k} T(a_in) + D(n)
\]

where \( 0 < a_i < 1, \ i = 1, \ldots, k \) and \( D(n) = n^d f(n) \) for a nondecreasing function \( f(n) \). (change the arguments \( a_i n \) to \( a_i n + A_i \) if you wish).

Set \( s = \sum_{i=1}^{k} a_i^d \).

\[
T(n) = \begin{cases} 
\Theta(D(n)) & s < 1 \\
O(D(n) \log n) & s = 1 \\
\Theta(n^h) & s > 1 
\end{cases}
\]

where \( h \) satisfies \( \sum_{i=1}^{k} a_i^h = 1 \).

The middle case is tight, \( T(n) = \Theta(D(n) \log n) \), for \( s = 1 \) and \( f \) satisfying (F).

**Example 9.** \( T(n) = n^3 + 3T(2n/3) + 3T(n/3) \quad T(n) = \underline{\ldots} \)

**Example 10.** \( T(n) = n^4 + 3T(2n/3) + 3T(n/3) \quad T(n) = \underline{\ldots} \)

**Example 11.** \( T(n) = n^2 + 3T(2n/3) + 3T(n/3) \quad T(n) = \underline{\ldots} \)

**Remarks**

1. for equal size problems, this is precisely the original F Master Theorem

   since \( h = \log_b a \) satisfies \( a(1/b)^h = 1 \)

2. since \( h \) is usually hard to compute, we phrase the first 2 cases using only \( d \)

   but they correspond to the cases \( d > h \) and \( d = h \) of the F Master Theorem