Analysis of Divide-and-Conquer Algorithms

The divide-and-conquer paradigm (Ch.2)

- divide **the problem into a number of subproblems**
- conquer **the subproblems (solve them)**
- combine **the subproblem solutions to get the solution to the original problem**

Example: Merge Sort

- **divide the n-element sequence to be sorted into two n/2-element sequences.**
- **conquer the subproblems recursively using merge sort.**
- **combine the resulting two sorted n/2-element sequences by merging.**

More Math Review

- Floor: $\lfloor x \rfloor$ = the largest integer $\leq x$
- Ceiling: $\lceil x \rceil$ = the smallest integer $\geq x$
- Summations: (see Appendix A, p.1058)
- Geometric, Telescoping & Harmonic series: (see Appendix A, p.1060-1061)

```
Merge-Sort(A,p,r)
if p < r then
   q ← ⌊(p+r)/2⌋
   Merge-Sort (A,p,q)
   Merge-Sort (A,q+1,r)
   Merge (A,p,q,r)
```

Initial call:
`Merge-sort(A,1,length(A))`

The Merge subroutine takes $\theta(n)$ time to merge n elements that are divided into two *sorted* arrays of $n/2$ elements each.

```
Merge(A,p,q,r)
n1← q-p+1; n2← r-q;
Create arrays
   L[1...n1+1] and
   R[1...n2+1]
for i← 1 to n1
    L[i]← A[p+i-1]
for i← 1 to n2
    R[i]← A[q+i]
L[n1+1] = R[n2+1] = ∞
i ← j ← 1
for k ← p to r
  if L[i] ≤ R[j]
      A[k]← L[i]
      i ← i+1
  else A[k]← R[j]
      j ← j+1
```

Analyzing Divide-and-Conquer Algorithms

A recursive algorithm can often be described by a *recurrence equation* that describes the overall runtime on a problem of size *n* in terms of the runtime on smaller inputs.

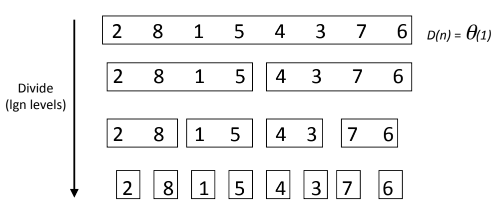
For divide-and-conquer algorithms, we get recurrences like:

$$T(n) = \begin{cases} \theta(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

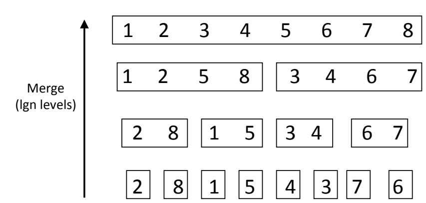
where

- a = number of subproblems we divide the problem into
- n/b = size of the subproblems (in terms of n)
- $D(n)$ = time to divide the size n problem into subproblems
- $C(n)$ = time to combine the subproblem solutions to get the answer for the problem of size n

Analyzing Merge-Sort



Analyzing Merge-Sort



$C(n) = \theta(n)$

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

Recurrence for worst-case running time for Merge-Sort

Analyzing Merge-Sort

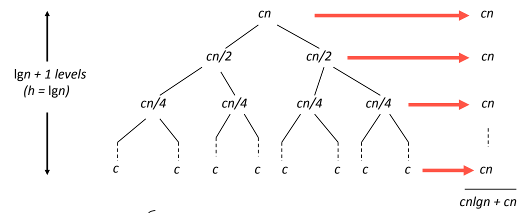
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{otherwise} \end{cases}$$

Recurrence for worst-case running time for Merge-Sort

$$aT(n/b) + D(n) + C(n)$$

- a = 2 (two subproblems)
- n/b = n/2 (each subproblem has size approx. $n/2$)
- $D(n)$ = $\Theta(1)$ (just compute midpoint of array)
- $C(n)$ = $\Theta(n)$ (merging can be done by scanning sorted subarrays)

Recursion Tree for Merge-Sort



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

Recurrence for worst-case running time of Merge-Sort