Getting Started (cont.)

Content

- Sorting problem
- Sorting algorithms
  - Insertion sort - $\Theta(n^2)$.
  - Merge sort - $\Theta(n \lg n)$.

Merge

- What is merge sort?
  - A sorting algorithm using merge.

- What is merge?
  - Given two sorted lists of keys, generate a sorted list of the keys in the given sorted lists.
  - <1, 5, 6, 8> < 2, 4, 7, 9> → < 1, 2, 4, 5, 6, 7, 8, 9>

Merge

- Merging example
  - <1, 5, 6, 8> <2, 4, 7, 9> → < 1 >
  - < 5, 6, 8> <2, 4, 7, 9> → < 1, 2 >
  - < 5, 6, 8> < 4, 7, 9> → < 1, 2, 4 >
  - < 5, 6, 8> < 7, 9> → < 1, 2, 4, 5 >
  - < 6, 8> < 7, 9> → < 1, 2, 4, 5, 6 >
  - < 8> < 7, 9> → < 1, 2, 4, 5, 6, 7 >
  - < 8> < 9> → < 1, 2, 4, 5, 6, 7, 8 >
  - < > < 9> → < 1, 2, 4, 5, 6, 7, 8, 9>

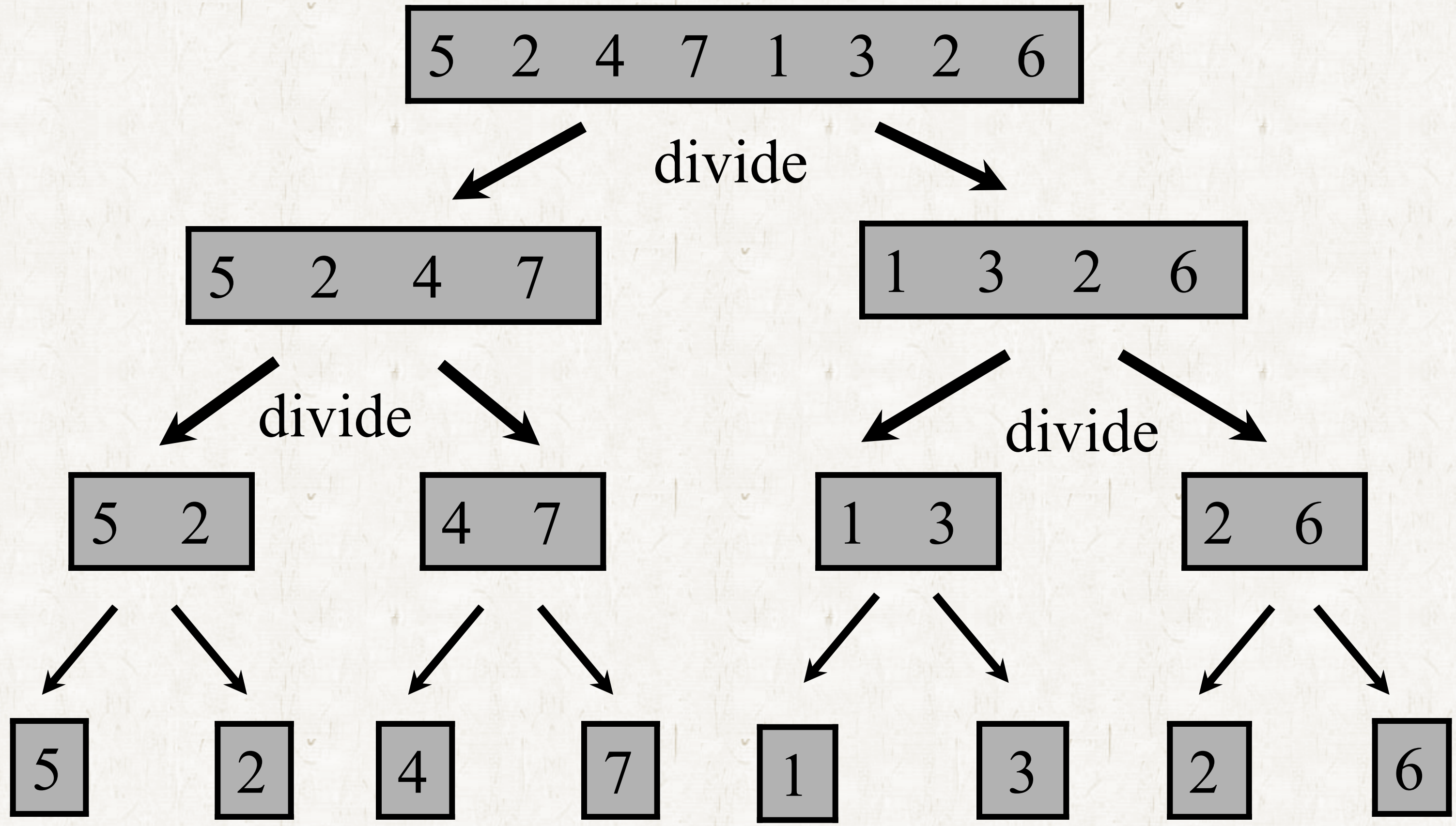
Merge

- Running time of merge
  - Let n_1 and n_2 denote the lengths of two sorted lists.
  - $\Theta(n_1 + n_2)$ time.
    - Main operations: compare and move
    - #comparison $\leq$ #movement
    - Obviously, #movement $= n_1 + n_2$
    - So, #comparison $\leq n_1 + n_2$
    - Hence, #comparison + #movement $\leq 2(n_1 + n_2)$
    - which means $\Theta(n_1 + n_2)$.

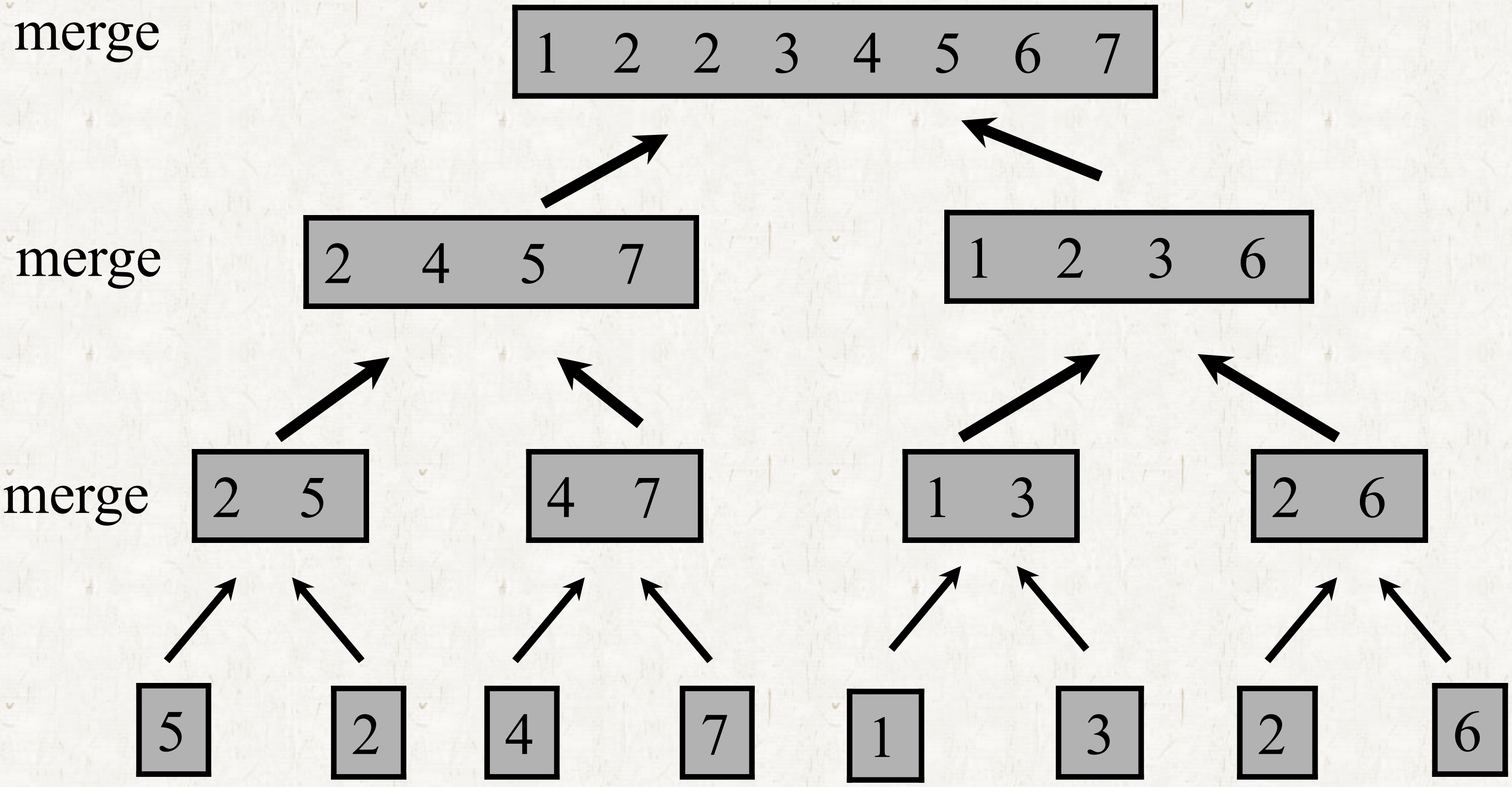
Merge sort

- A divide-and-conquer approach
  - **Divide:** Divide the n keys into two lists of $n/2$ keys.
  - **Conquer:** Sort the two lists recursively using merge sort.
  - **Combine:** Merge the two sorted lists.

Merge sort


5 2 4 7 1 3 2 6
divide
5 2 4 7
1 3 2 6
divide
divide
5 2
4 7
1 3
2 6
5
2
4
7
1
3
2
6

Merge sort



Pseudo code

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Merge(A, p, q, r)

for $i = 1$ **to** n_1
    $L[i] = A[p + i - 1]$
for $j = 1$ to n_2
    $R[j] = A[q + j]$
$L[n_1+1] = \infty$
$R[n_2+1] = \infty$
$i = 1$
$j = 1$
for $k = p$ **to** r
    if $L[i] \le R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else $A[k] = R[j]$
        $j = j + 1$

What if there are two keys with the same values?

Insertion sort? Merge sort?

Running time

- **Divide:** $\Theta(1)$
  - The divide step just computes the middle of the subarray, which takes constant time.
- **Conquer:** $2T(n/2)$
  - We recursively solve two subproblems, each of size $n/2$.
- **Combine:** $\Theta(n)$
  - We already showed that merging two sorted lists of size $n/2$ takes $\Theta(n)$ time.

Running time

- $T(n)$ can be represented as a recurrence.

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

Running time

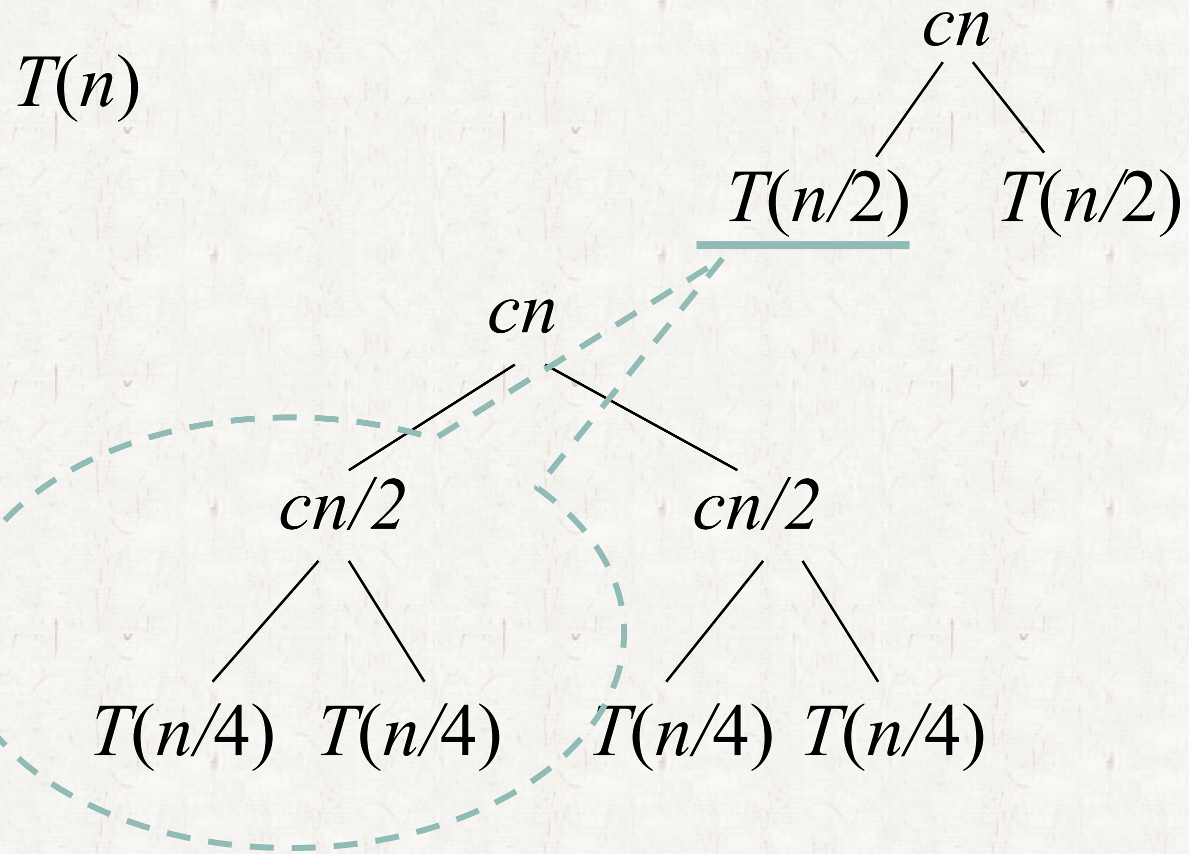
- where the constant c represents the time required to solve problems of size 1 as well as the time per array element of the divide and combine steps.

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

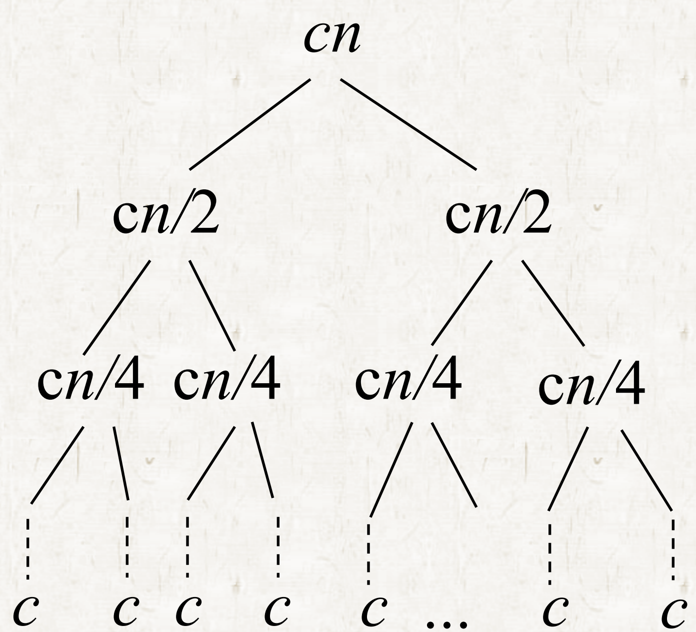
$$\Downarrow$$

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

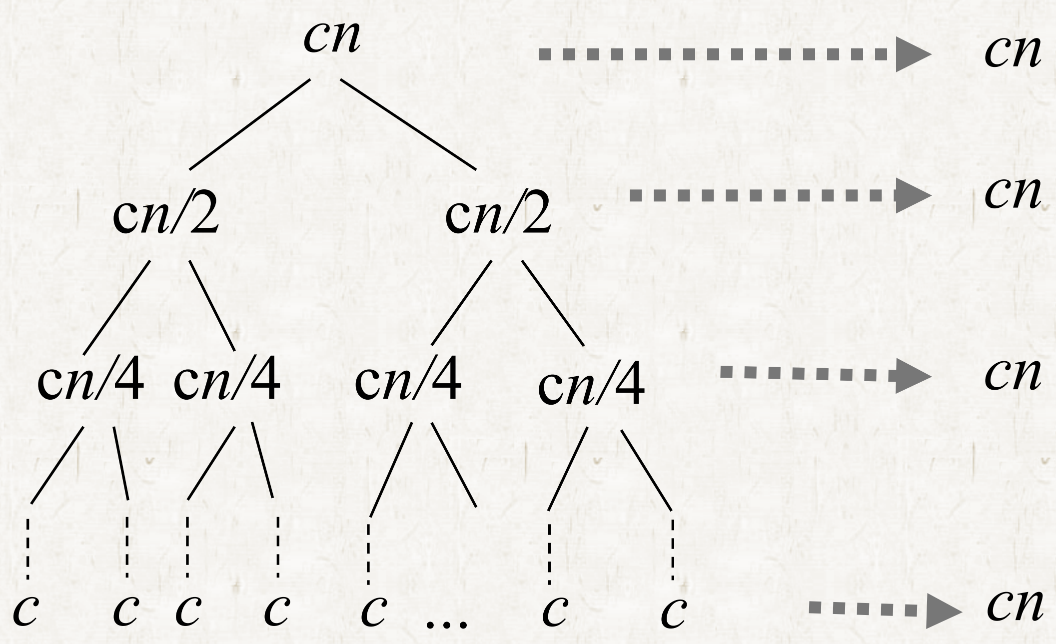
Recursion tree



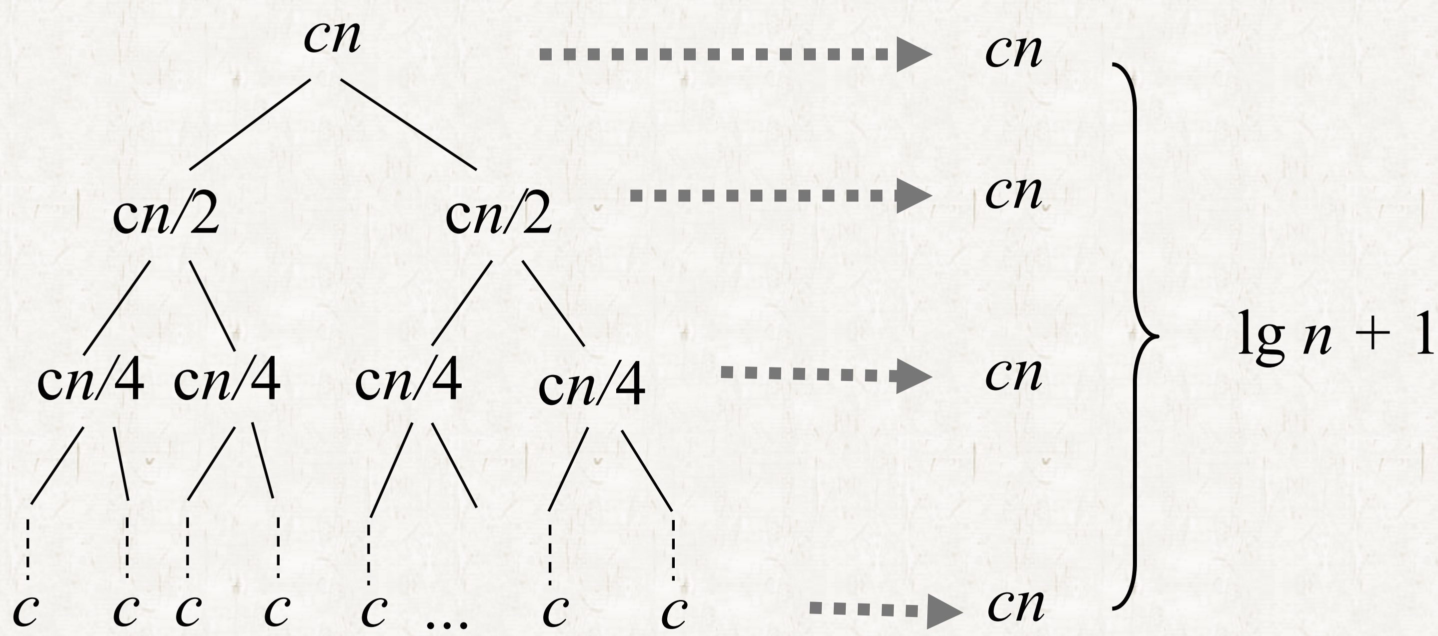
Recursion tree



Recursion tree



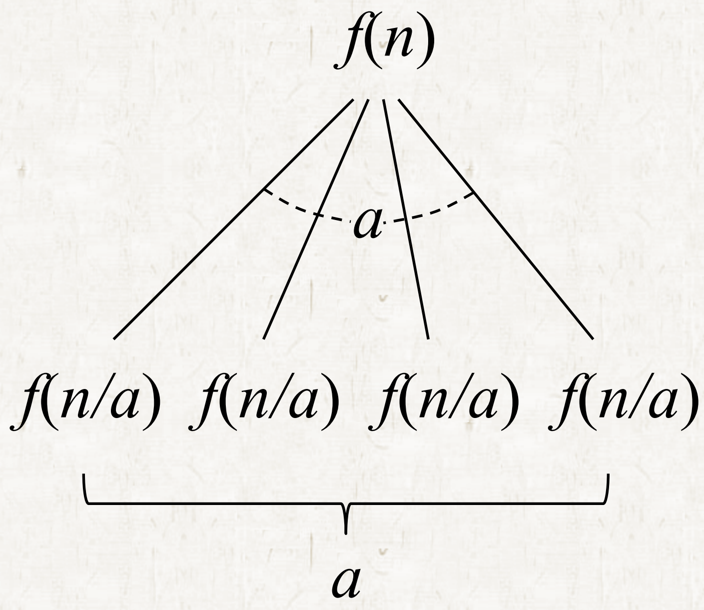
Recursion tree



Total : $cn\lg n + cn = \Theta(n\lg n)$

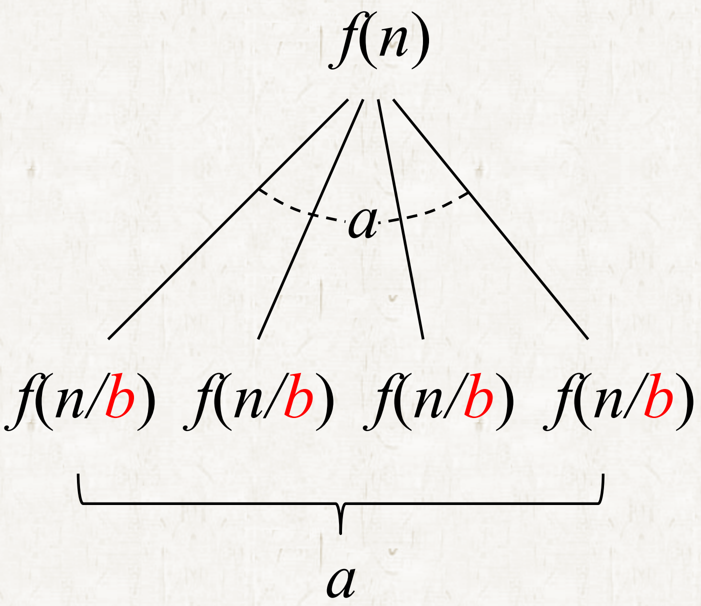
Divide and conquer

- Divide and conquer with a subproblems and each of which is $1/a$ the size of the original.



Divide and conquer

- Divide and conquer with a subproblems and each of which is $1/b$ the size of the original.



Divide and conquer

- Suppose that our division of the problem yields a subproblems, each of which is $1/b$ the size of the original.
- Let $D(n)$ denote time to divide the problem into subproblems.
- Let $C(n)$ denote time to combine the solutions to the subproblems into the solution to the original problem.
- We get the recurrence

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

Divide and conquer

- For merge sort,
  - $a = b = 2$.
  - $D(n) = \Theta(1)$.
  - $C(n) = \Theta(n)$.
- The worst-case running time $T(n)$ of merge sort:

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

Self-study

- **Merge sort**
  - Exercise 2.3-1
  - Exercise 2.3-2

- **Horner’s rule**
  - Problem 2-3 (a) (b)
  - Loop invariant is difficult.

More (sorting) algorithms

- **Binary Search**
  - Exercise 2.3-5
- **Selection sort**
  - Exercise 2.2-2
- **Bubble sort**
  - Problem 2-2
- http://www.sorting-algorithms.com/