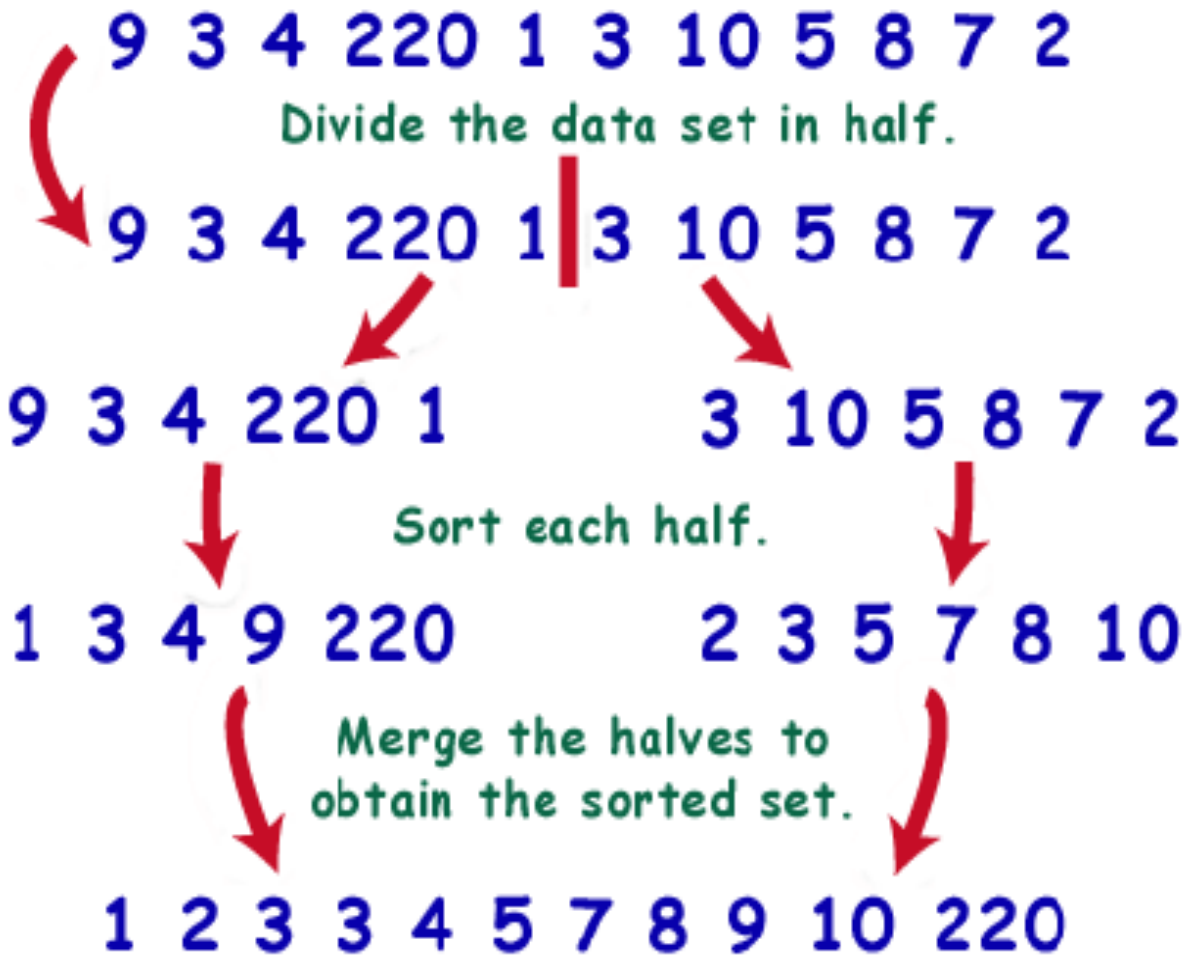
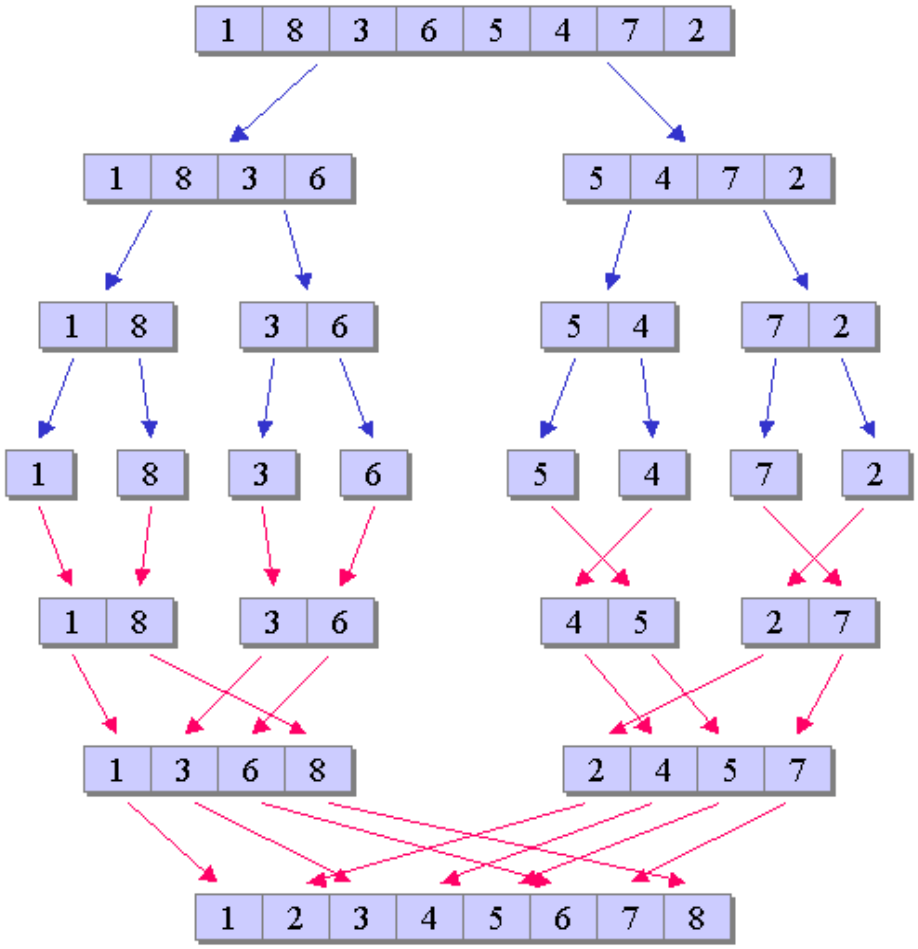
COMP20003 Workshop Week 8

1	Merge Sort: top-down algorithm
2	The Master Theorem
3	Merge Sort: bottom-up algorithms
4	Group exercises
5	Lab: implementing bottom-up mergesort (P 8.1, 8.2)

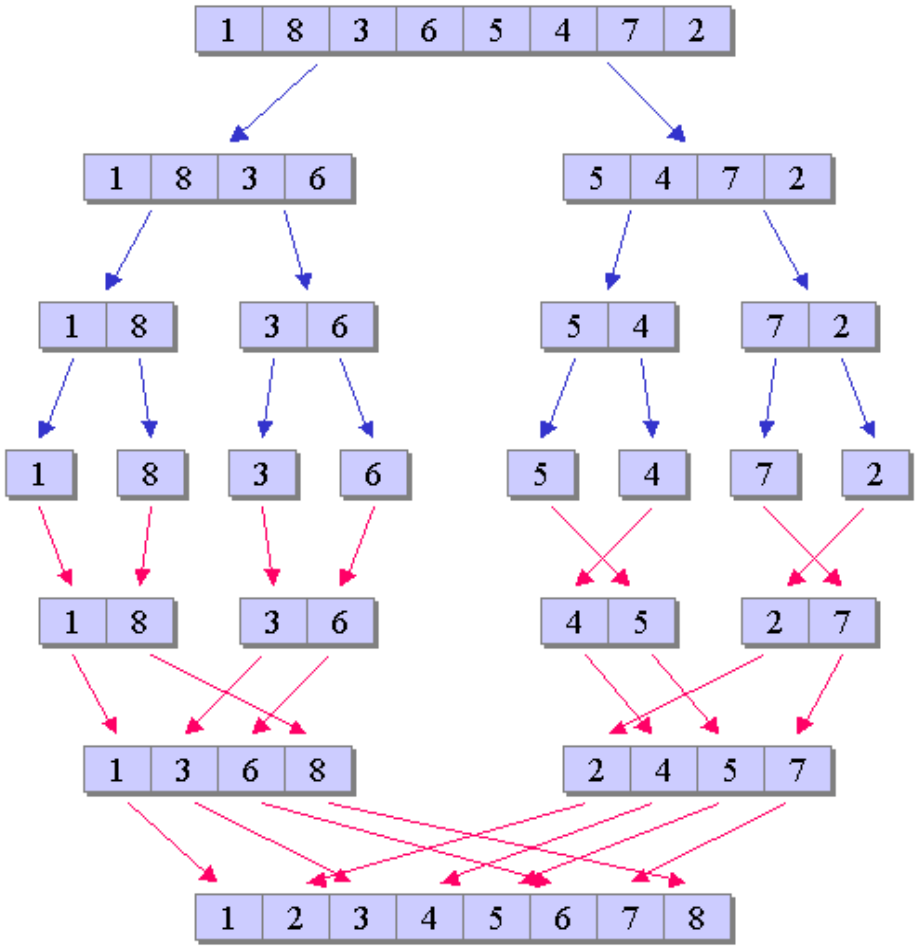
Merge Sort: Main Idea



Top-Down MergeSort: Divide-And-Conquer!



Top-Down MergeSort: Implementation Notes

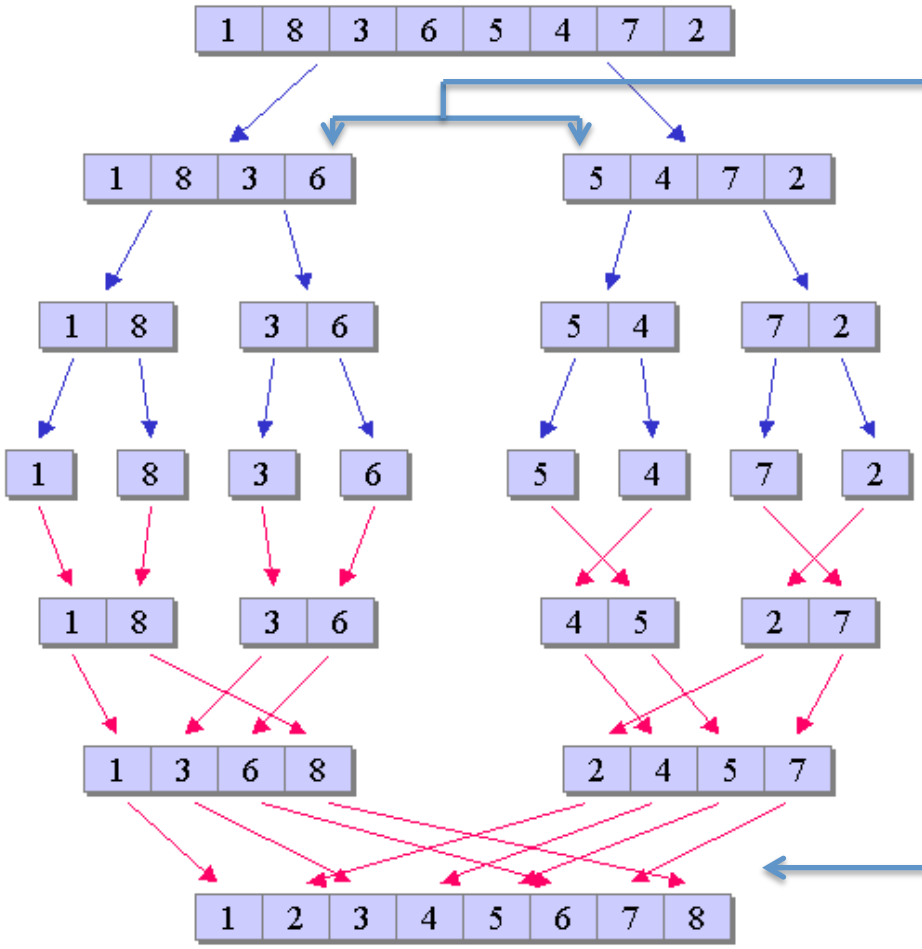


the sorting algorithm is simple!

"divide"
is
simple!

"conquer"= merge
and is more complicated
Need to be careful on using a temporal array for the merging result

Complexity of mergesort: Recurrences



Recurrence:

$$T(n) = 2T(n/2) + \Theta(n)$$

$$T(1) = 1$$

Recurrences: examples

Recurrences for the algorithms:

- computing n!
- binary search

The Master Theorem

If

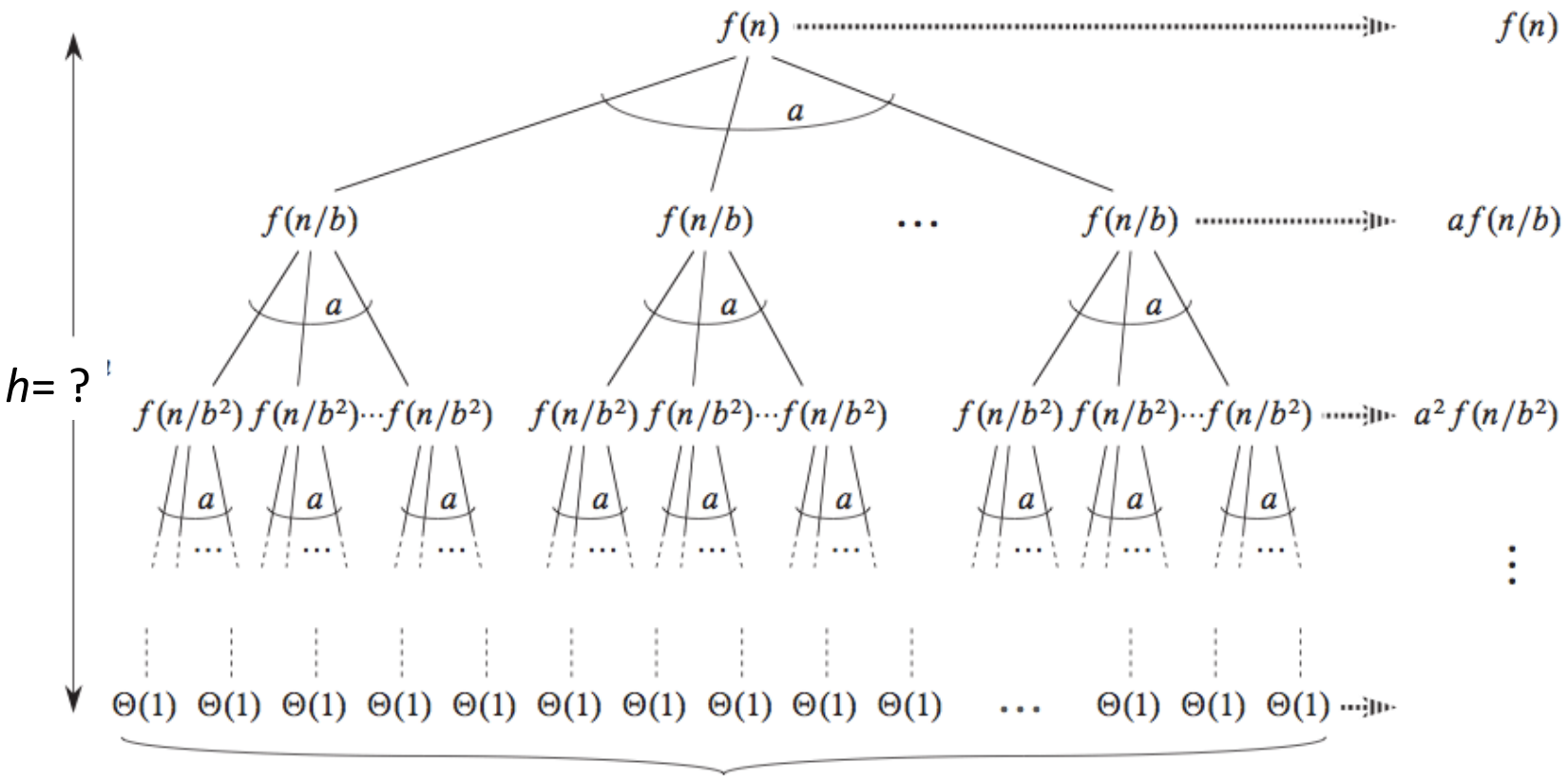
$$T(n) = aT(n/b) + \Theta(n^d)$$

$$T(1) = \Theta(1)$$

where $a \geq 1$, $b > 1$, and $d \geq 0$, then

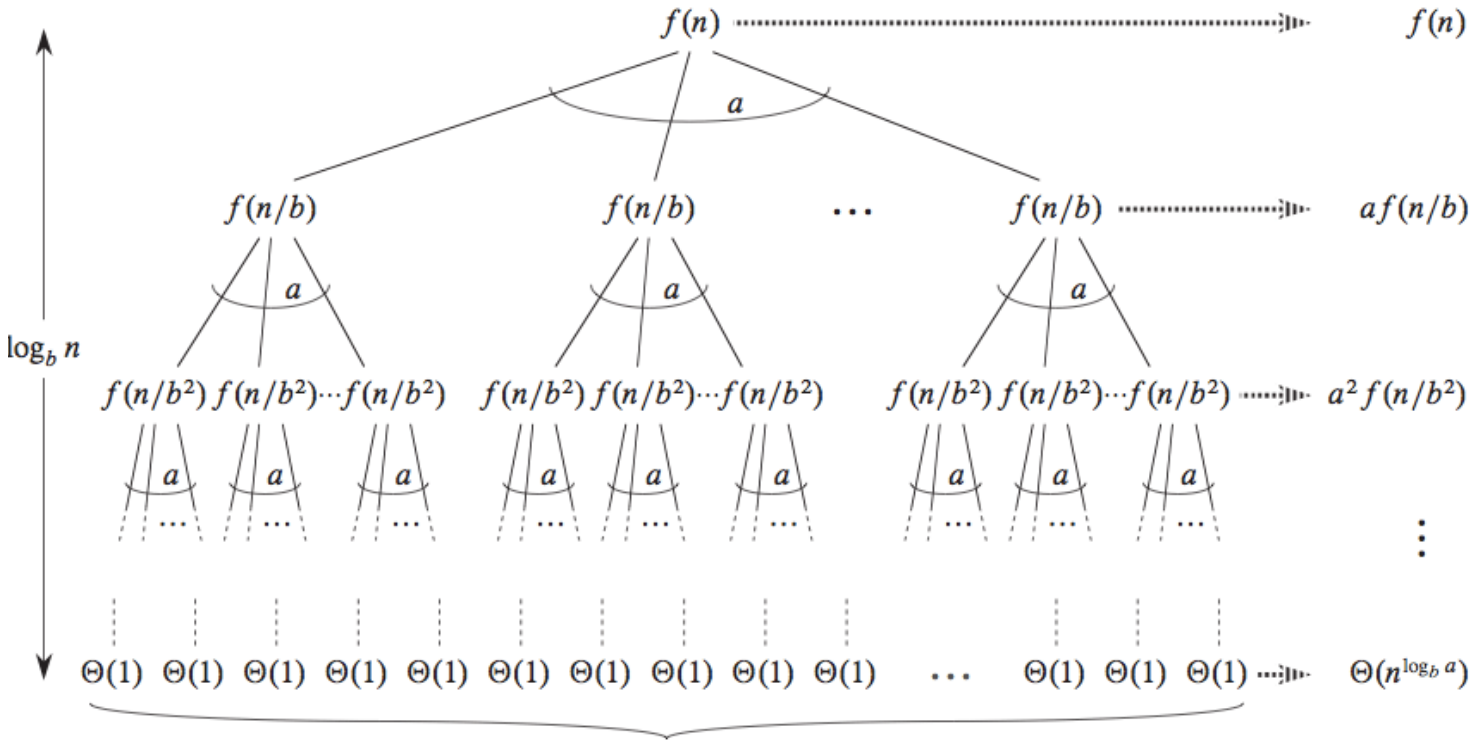
$$T(n) = \begin{cases} \Theta(n^d) & \text{if } d > \log_b a \\ \Theta(n^d \log n) & \text{if } d = \log_b a \\ \Theta(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Master Theorem is for Divider & Conquer with $f(n)=\theta(n)$



leaves= ?

Master Theorem: Complexity Computation



Note: $leaves = a^h = a^{\log_b n} = n^{\log_b a}$

Total time: $n^d + a\,(n/b)^d + a^2\,(n/b^2)^d + \ldots + a^h\,(n/b^h)^d$

$= n^d + n^d\,(a/b^d) + n^d\,(a/b^d)^2 + \ldots + n^d\,(a/b^d)^{\log_b n}$

Master Theorem: Complexity Computation

The running time:

$= \quad n^d + \quad n^d\,(a/b^d) \; + \quad n^d\,(a/b^d)^2 \; + \ldots \; + n^d\,(a/b^d)^{\log_b n}$

$= \quad n^d + \qquad\qquad \ldots \qquad\qquad + \; a^{\log_b n}\,(n \,/\, b^{\log_b n})^{\,d}$

Remember sum of geometric sequence:

$1 + c + c^2 + \ldots + c^n \;\; ?$

Winner	Condition	Equivalent condition	Time complexity
Conquer	$a < b^d$	$\log_b a < d$	$\Theta(n^d)$
Divider	$a > b^d$	$\log_b a > d$	$\Theta(n^{\log_b a})$
none	$a = b^d$	$\log_b a = d$	$\Theta(n^d \log n)$

Examples

Merge sort:

$$T(n)= 2T(n/2) + \Theta(n)$$

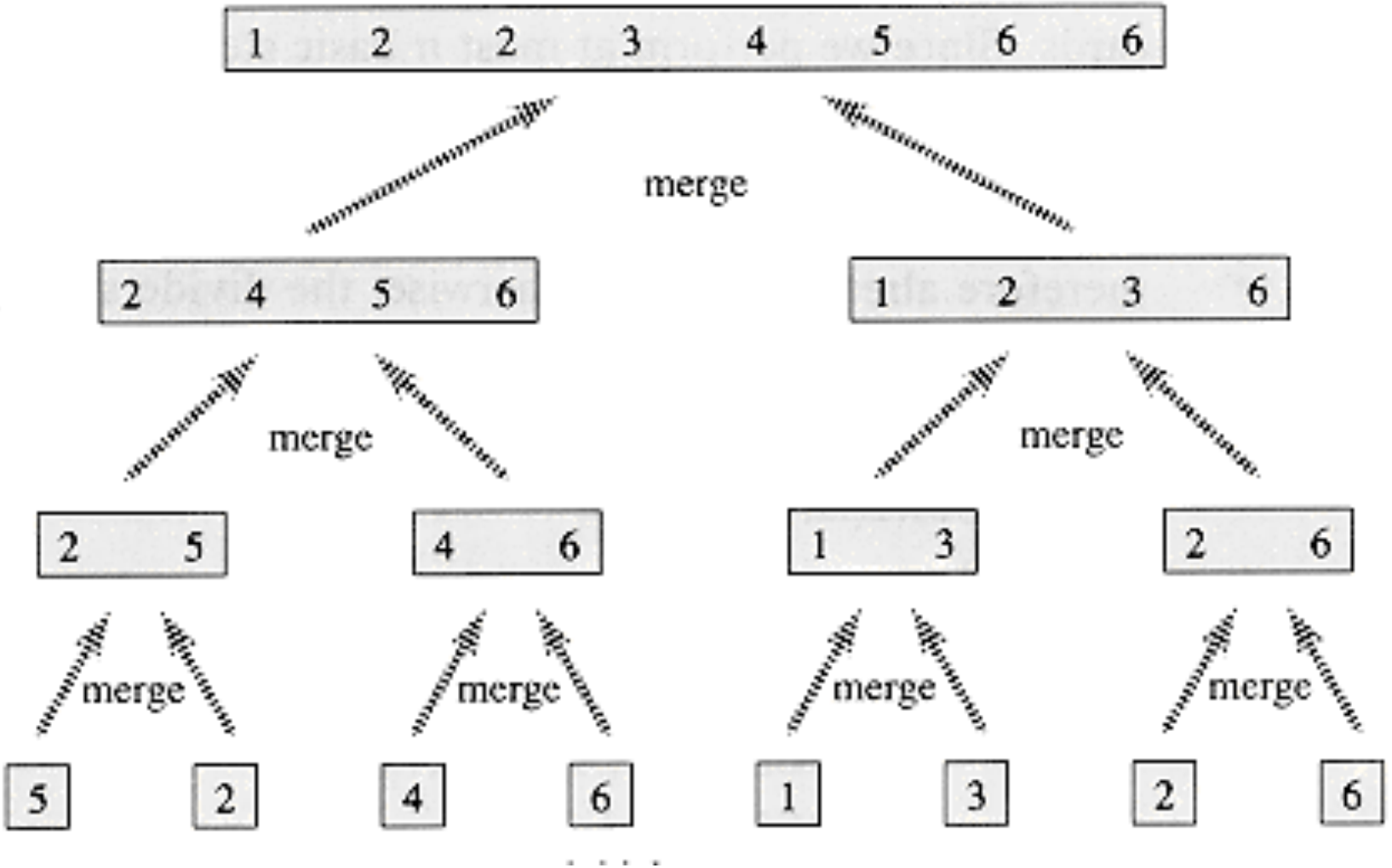
$$a = 2, b=2, d=1 \rightarrow d= \log_b a$$

$$\rightarrow T(n) = n \log n$$

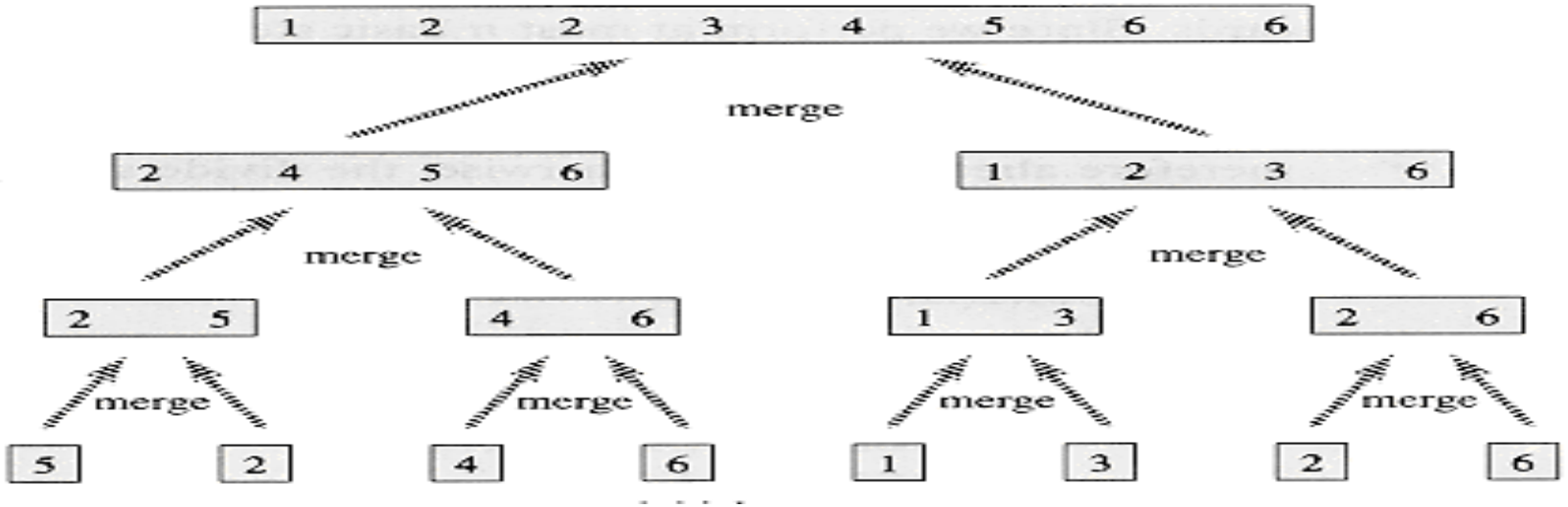
Computing n! ?

Binary Search ?

Merge Sort: Bottom-Up



Merge Sort: Bottom-Up



How to implement using a queue?

Group Exercises

1. Trace the action of bottom-up mergesort on the array:

```
2 3 97 23 15 21 4 23 29 37 5 23
```

2. Solve recurrences (assuming T(1)= 1):

a) T(n)= T(n-1) + n

b) T(n)= 2T(n/3) + n

2. Write a function:

```
void pair_merge(int A[], int B[], int C[], int m, int n)
```

that merges `A` (a sorted array of `m` elements) with `B` (a sorted array of `n` elements) into `c` (a sorted array of `m+n` elements).

3. Suppose that `A[]` is a sequence of sorted chunks of size `k`, write a code fragment that employs `pair_merge()` to turn `A[]` into a sequence of sorted chunks of size `2k`. Beware:

- the number of chunks might be not even
- the last chunk might have less than k elements

Lab: P8.1 and P8.2

Programming 7.1 Write code for bottom-up mergesort where the data are contained in an initially unsorted linked list. You will have to construct an artificial linked list to test your code. You can populate your linked list with random numbers before sorting.

Notebook users: still benefit from looking at `github`

Others: download from `github.com/anhvir/c203`

Programming 7.2 Write code for bottom-up mergesort where the data are contained in an initially unsorted array. You will have to construct an artificial array to test your code. You can populate your array with random numbers before sorting.

Notebook users:

Others: `p.7.2.c` is the same as the notebook version.