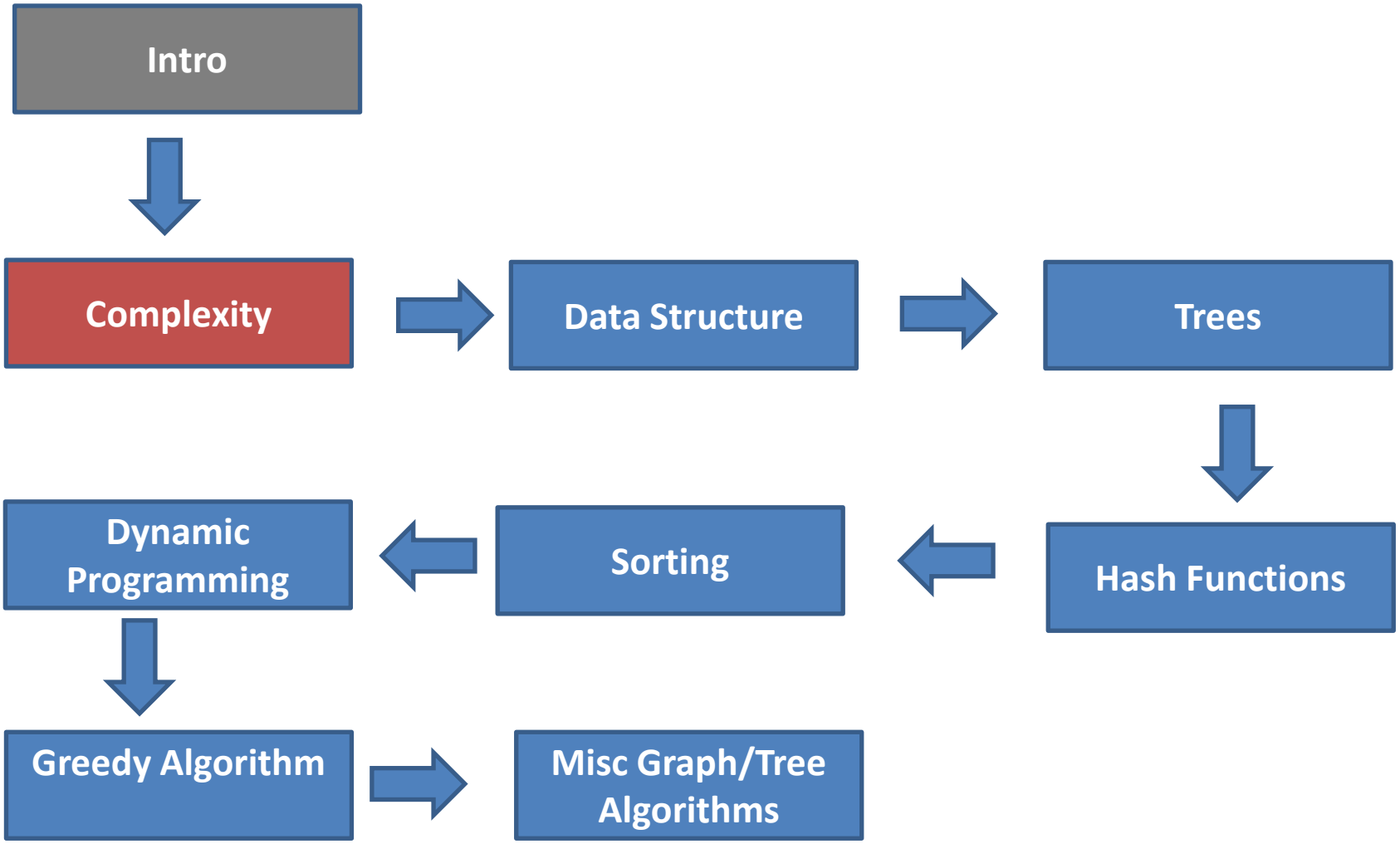
An Introduction to Algorithms

By

Hossein Rahmani

h_rahmani@iust.ac.ir

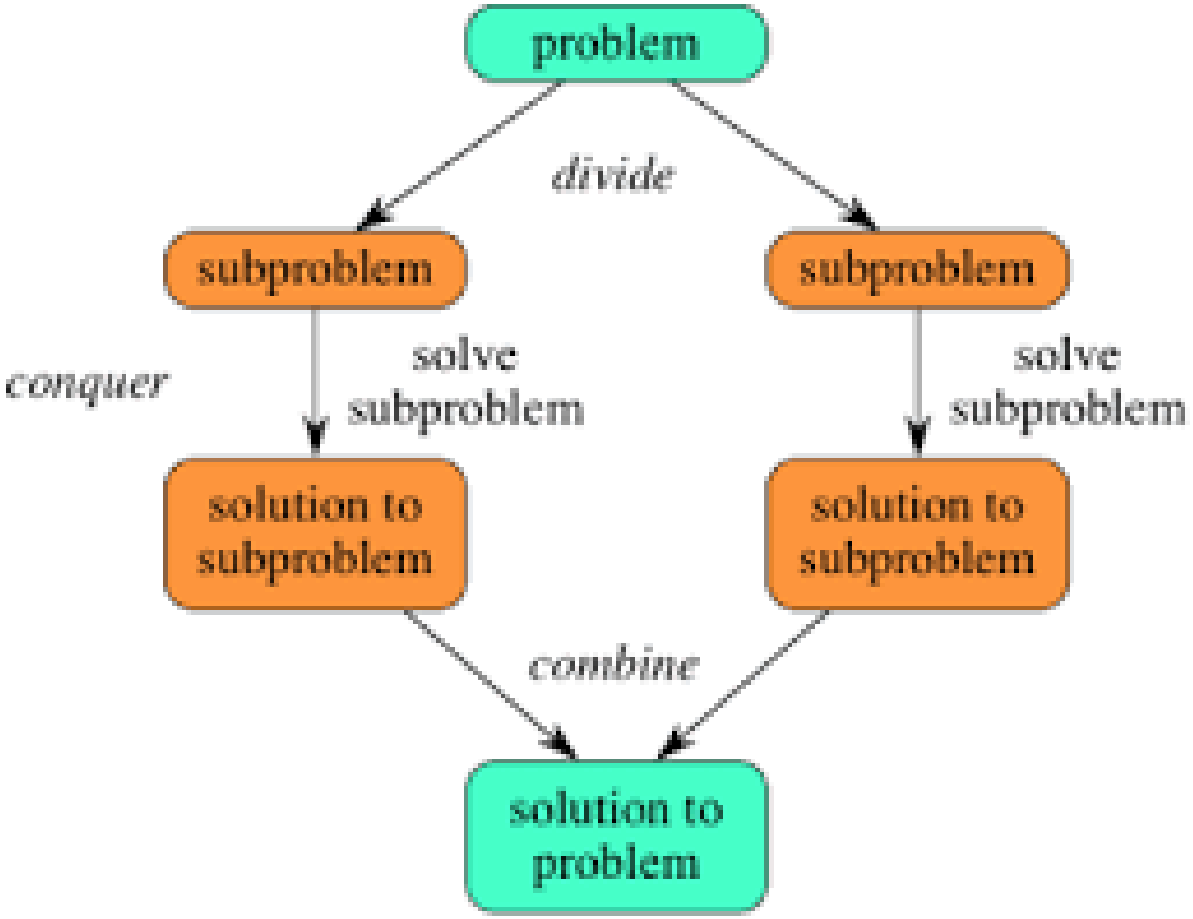
http://webpages.iust.ac.ir/h_rahmani/


Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms

Motivation

- For insertion sort (and other problems) as *n* doubles in size, the quadratic quadruples!
- Can we decrease *n*?
- What if we ***Divide*** the sort into smaller pieces?
- We can then solve those (***Conquer*** them).
- We need to be able to combine the pieces in a manner simpler than quadratic.

Divide-and-Conquer



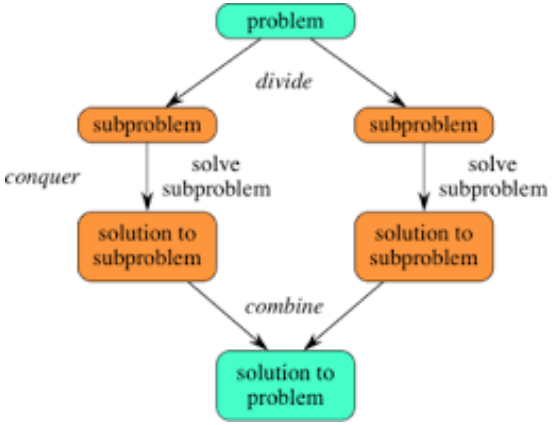

problem
divide
subproblem
subproblem
divide
divide
subproblem
subproblem
subproblem
subproblem
divide
divide
divide
divide
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
conquer
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
combine
combine
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
solution to subproblem
solution to subproblem
combine
solution to problem

Divide-and-Conquer

- **Divide**: If the input size is too large to deal with in a straightforward manner
  - Divide the problem into two or more disjoint subproblems
- **Conquer**: conquer recursively to solve the subproblems
- **Combine**: Take the solutions to the subproblems and "merge" these solutions into a solution for the original problem

Divide-and-Conquer Algorithms

- Binary search
- Quick sort
- Tower of Hanoi
- Fast multiplication
- Merge sort
  - Divide into two equal parts
  - Sort each part using merge-sort (recursion!!!)
  - Merge two sorted subsequences

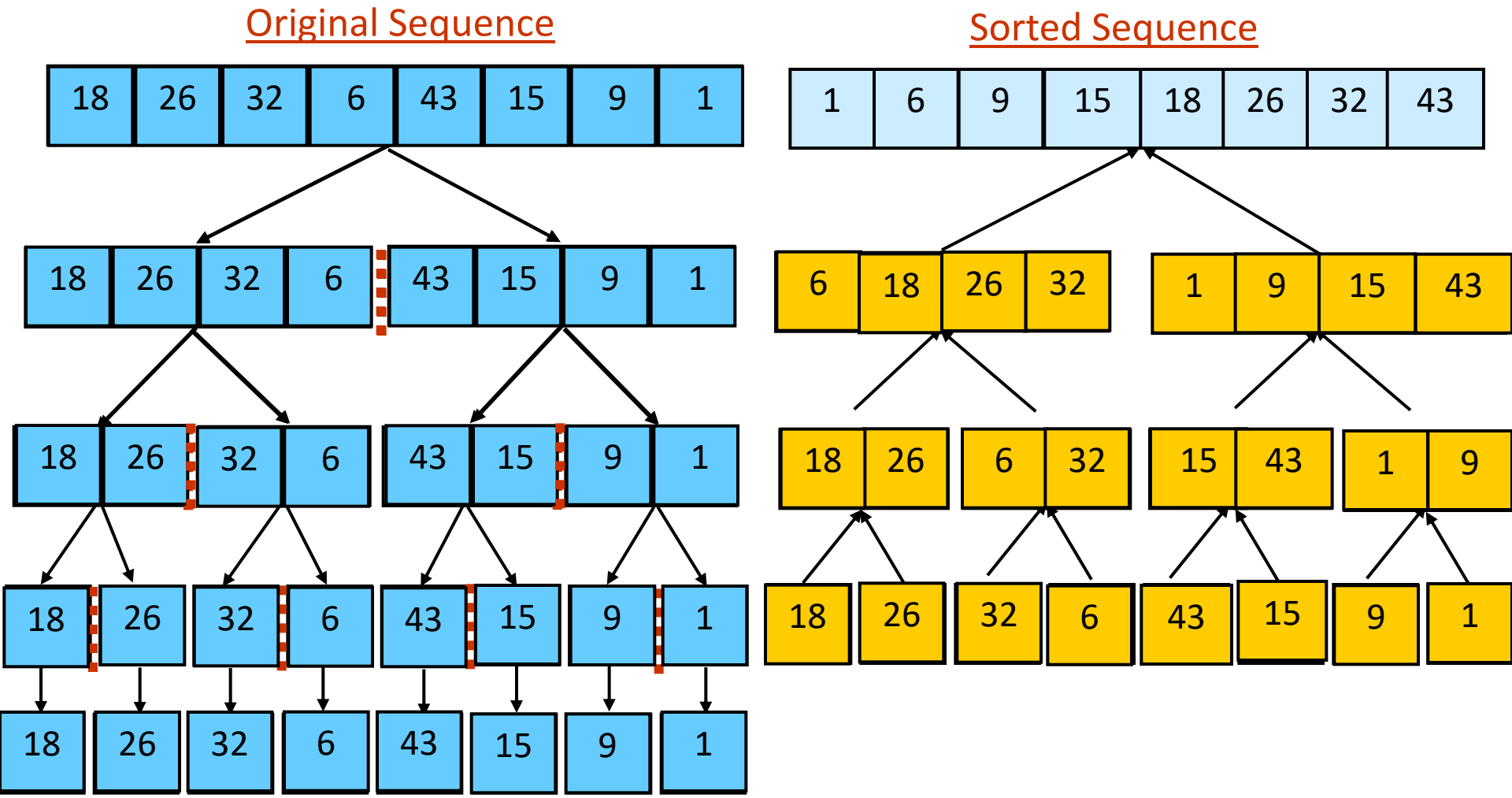


Merge Sort

```
MergeSort(A, left, right) {
   if (left < right) {
       mid = floor((left + right) / 2);
       MergeSort(A, left, mid);
       MergeSort(A, mid+1, right);
       Merge(A, left, mid, right);
   }
}

// Merge() takes two sorted subarrays of A and
// merges them into a single sorted subarray of A
//    (how long should this take?)
```

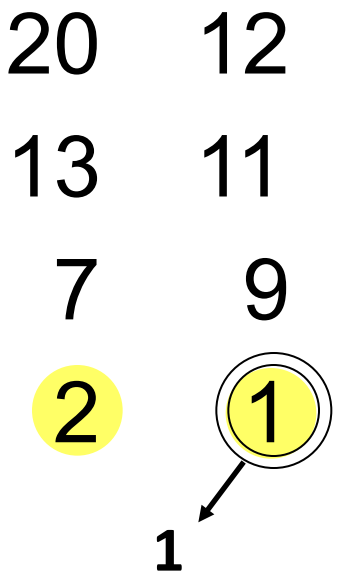
Merge Sort – Example



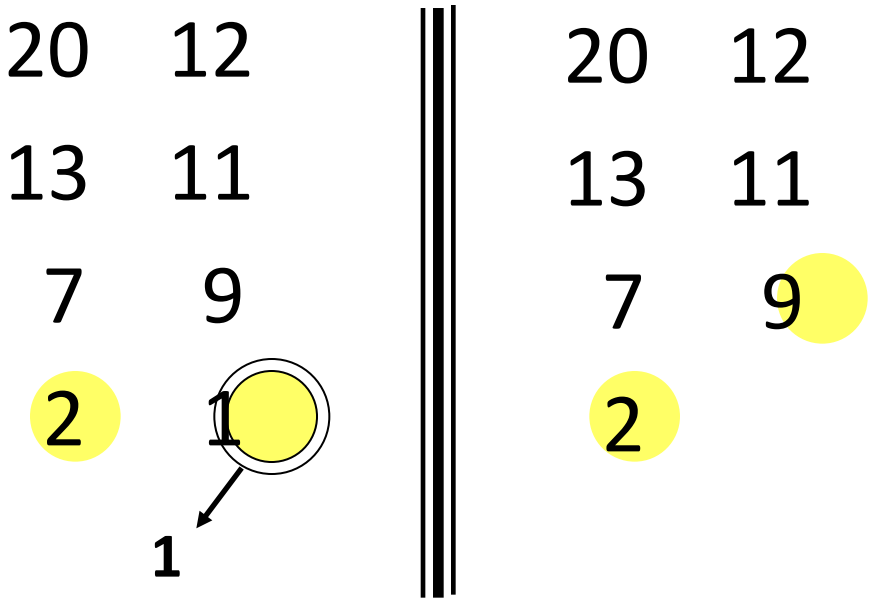
Merging two sorted arrays

20	12
13	11
7	9
2	1

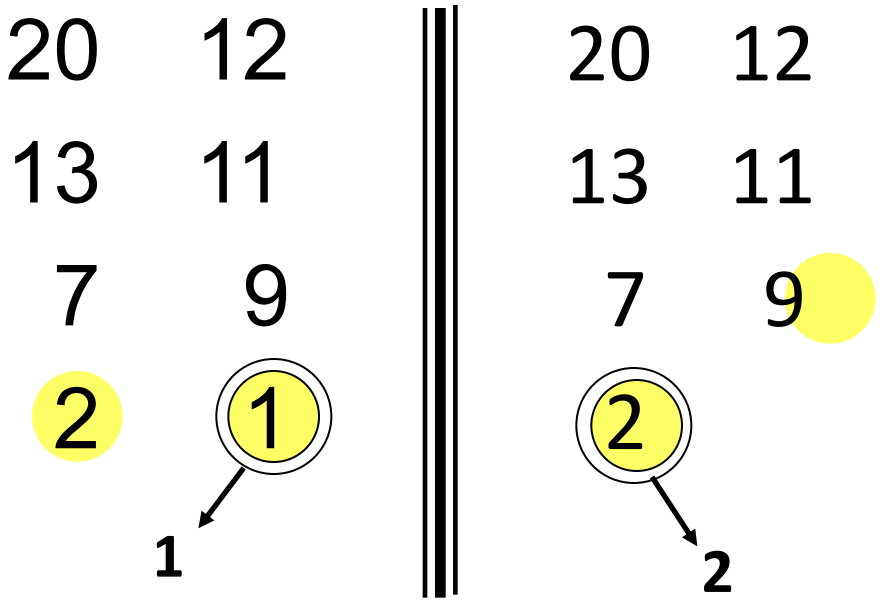
Merging two sorted arrays



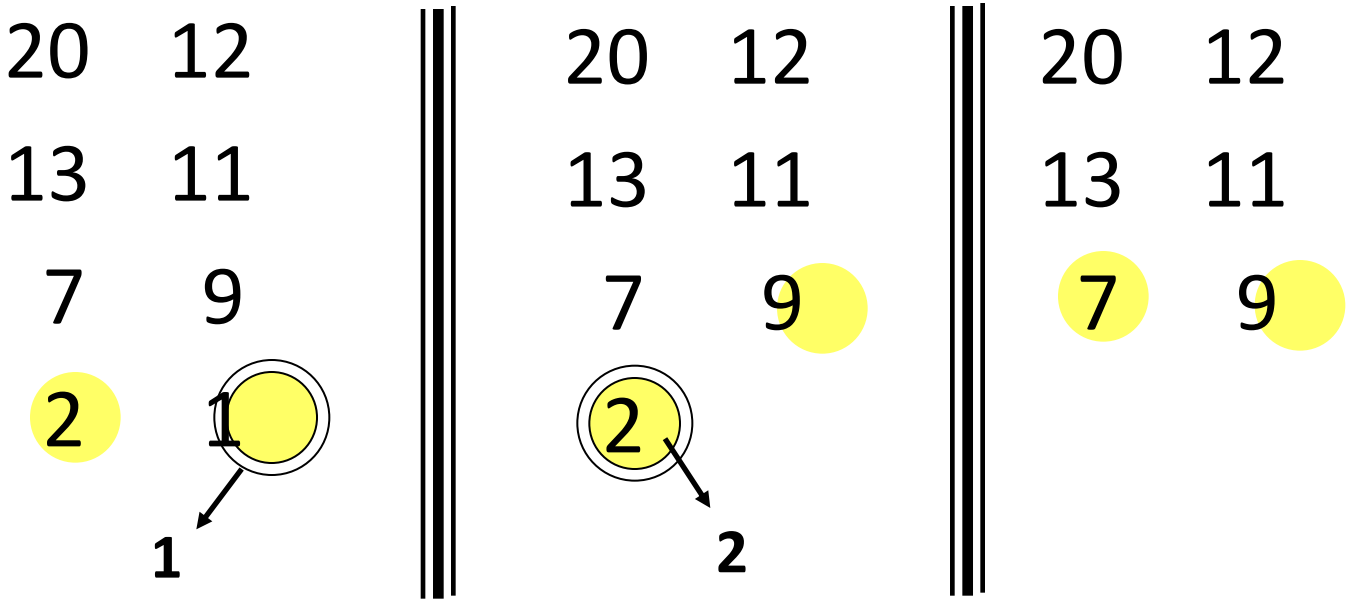
Merging two sorted arrays



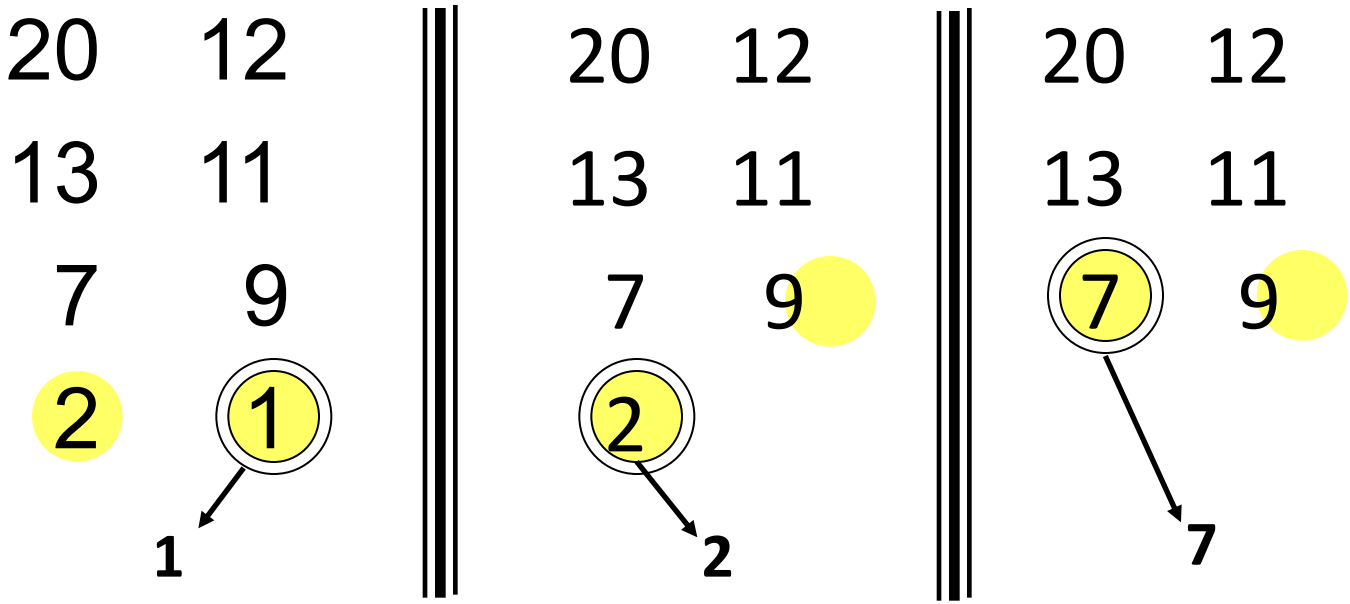
Merging two sorted arrays



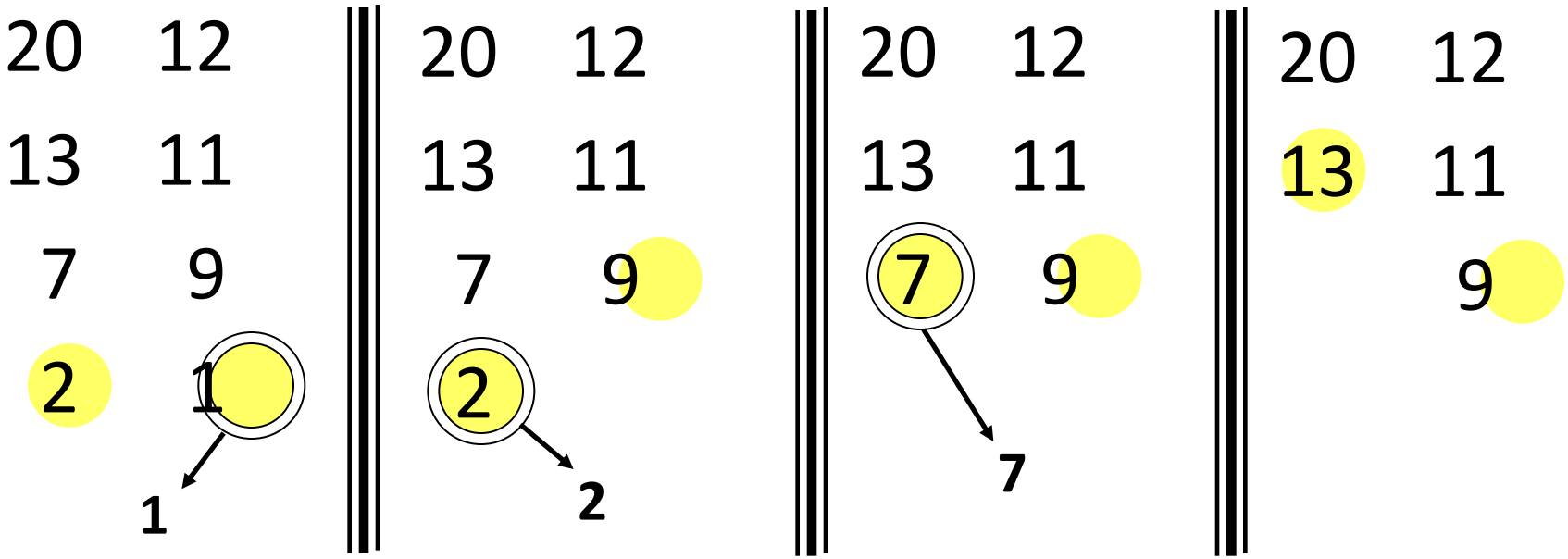
Merging two sorted arrays



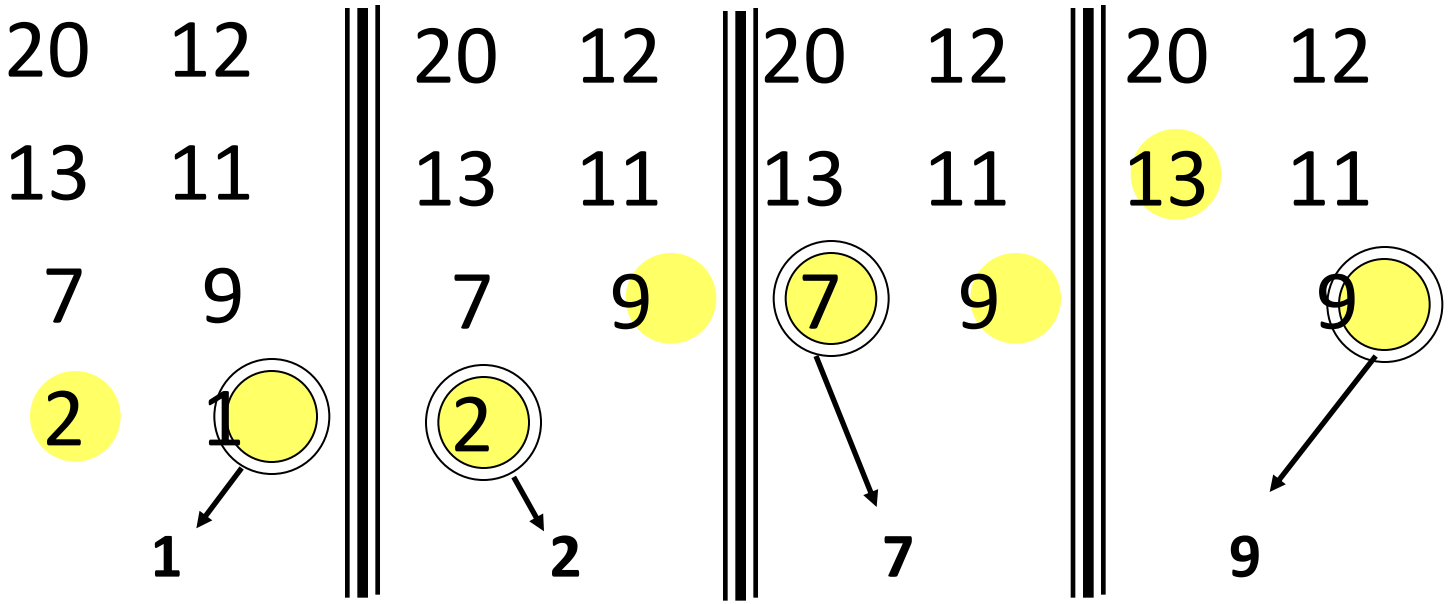
Merging two sorted arrays



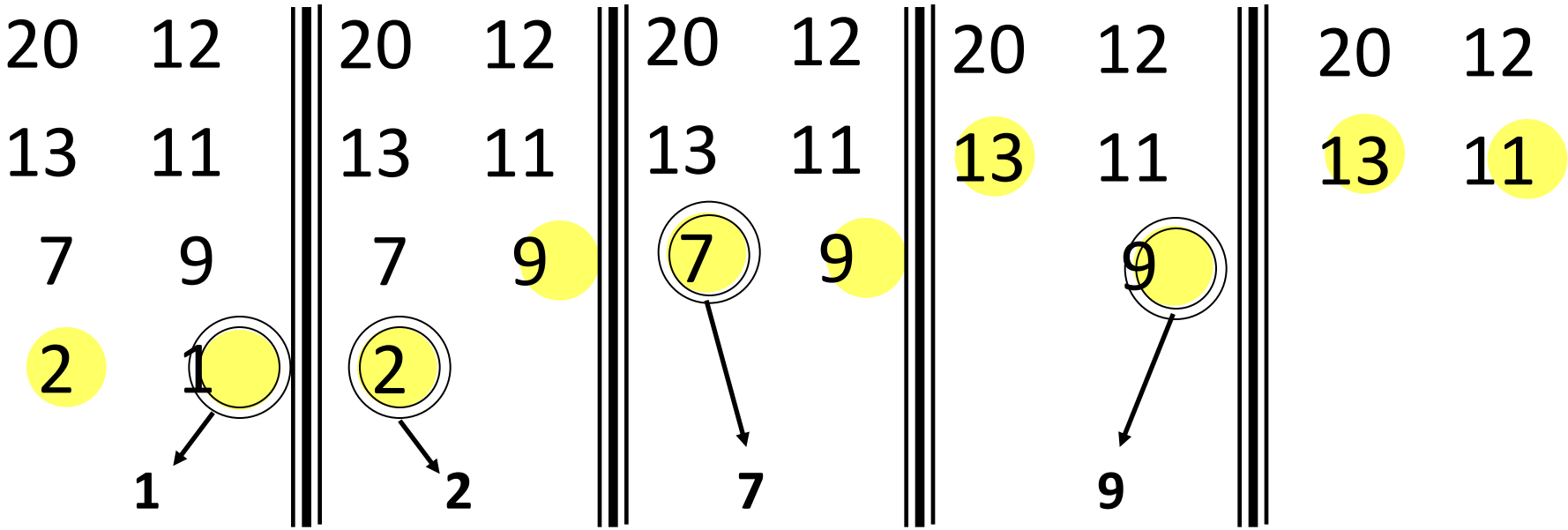
Merging two sorted arrays



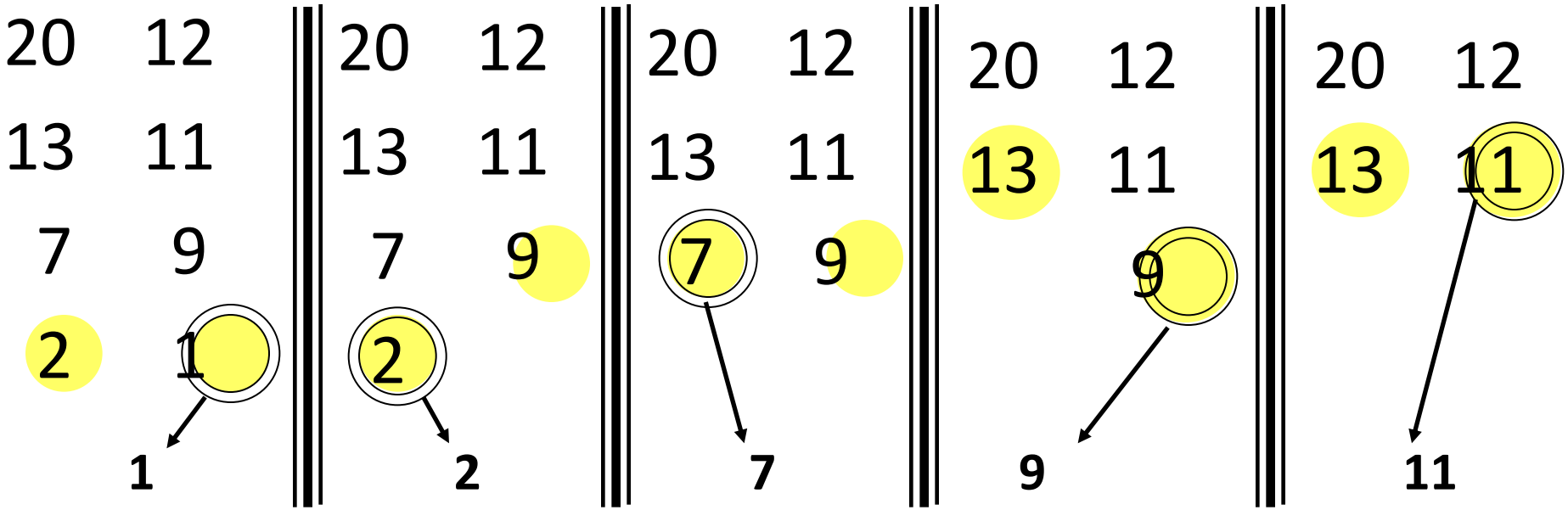
Merging two sorted arrays



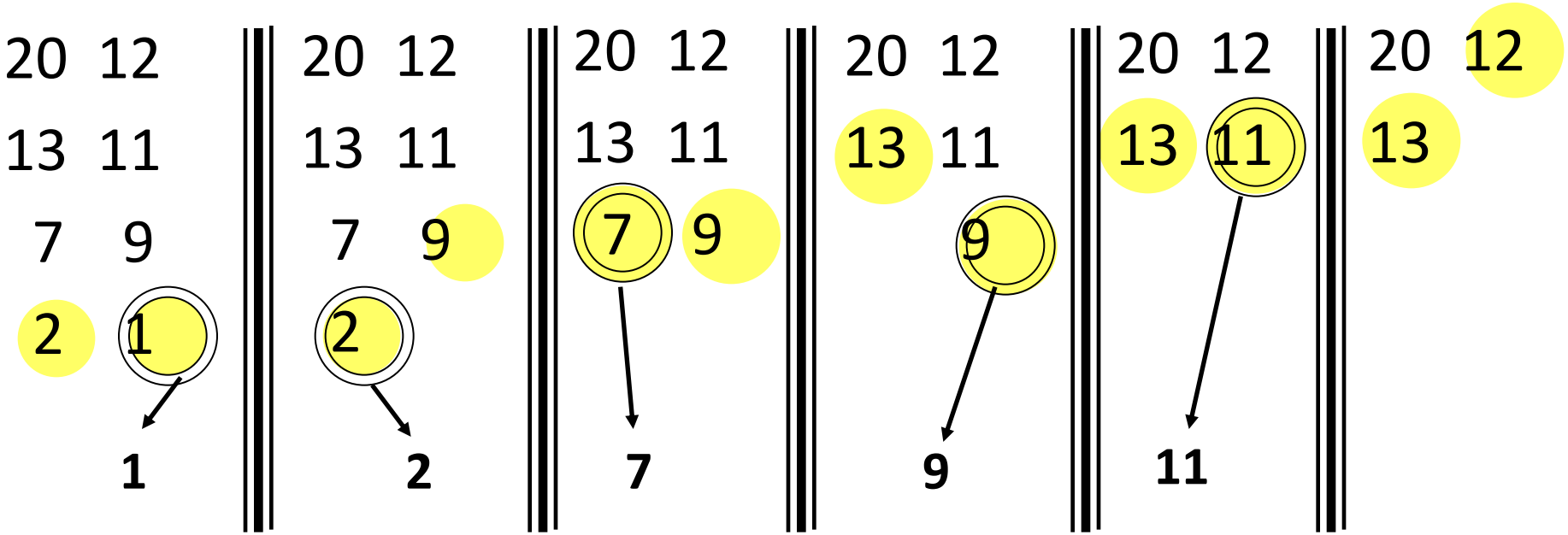
Merging two sorted arrays



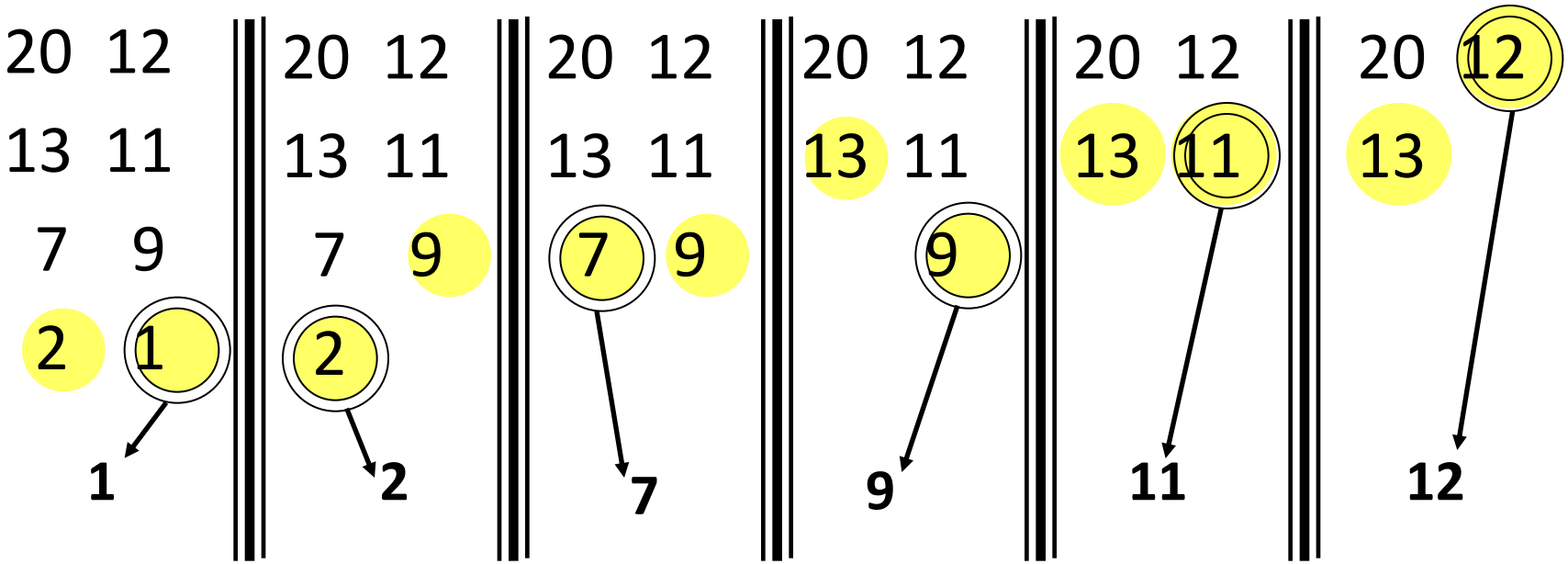
Merging two sorted arrays



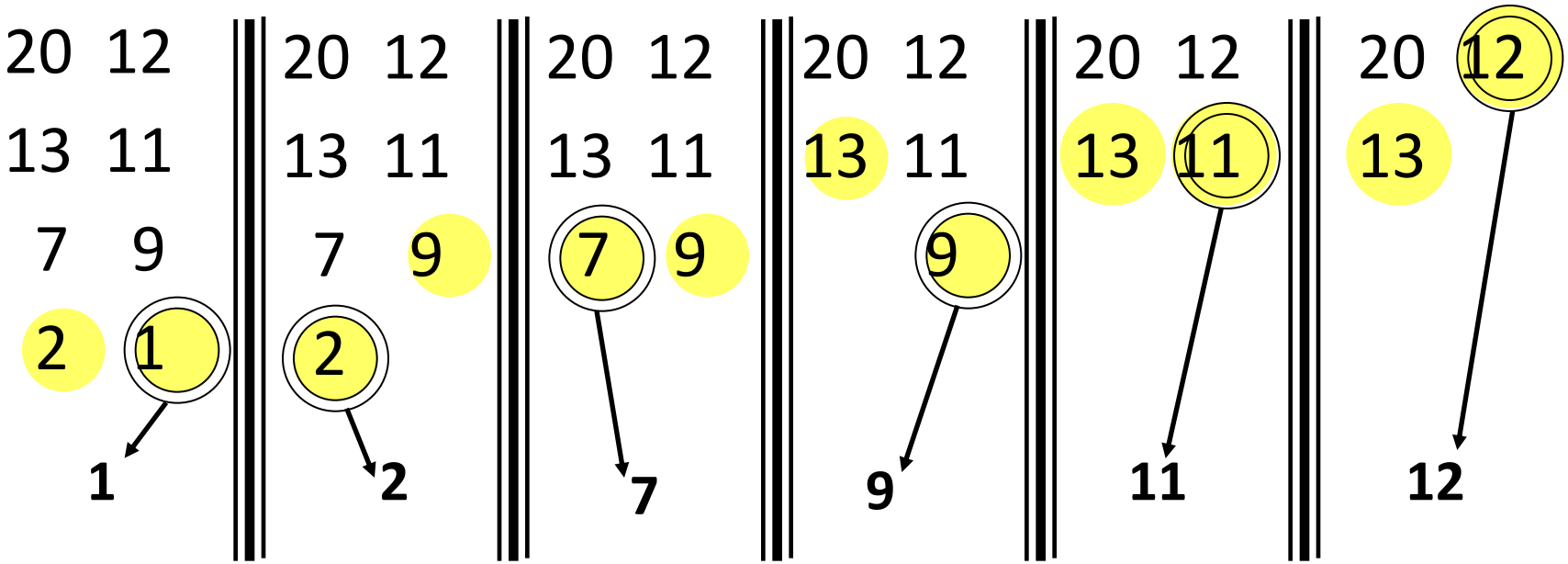
Merging two sorted arrays



Merging two sorted arrays

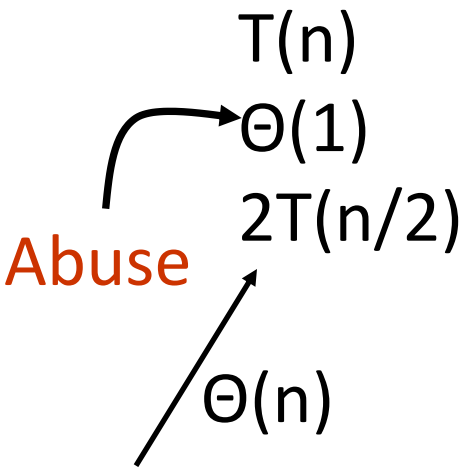


Merging two sorted arrays



Time = $\Theta(n)$ to merge a total of n elements (linear time).

Analyzing merge sort



MERGE-SORTA[1 . . n]

1.If n= 1, done.

2.Recursively sort A[1 . . ⌜ n/2 ⌟]
and A[⌜n/2⌟+1 . . n] .

3.“Merge”the 2sorted lists

Sloppiness: Should be T(⌜ n/2 ⌟) + T(⌜n/2⌟) ,
but it turns out not to matter asymptotically.

Recurrence for merge sort

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

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

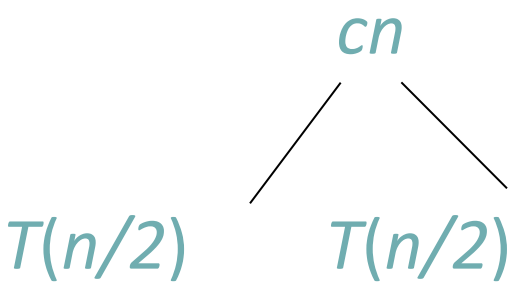
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$T(n)$

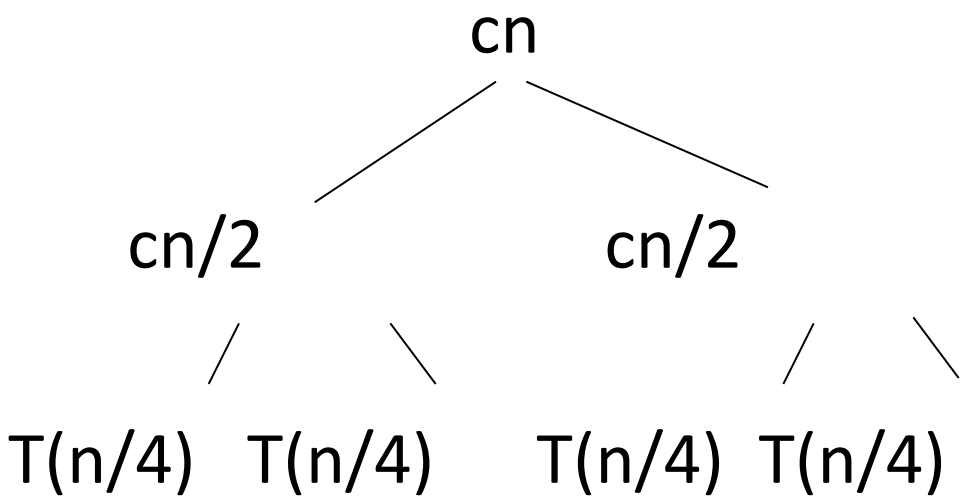
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



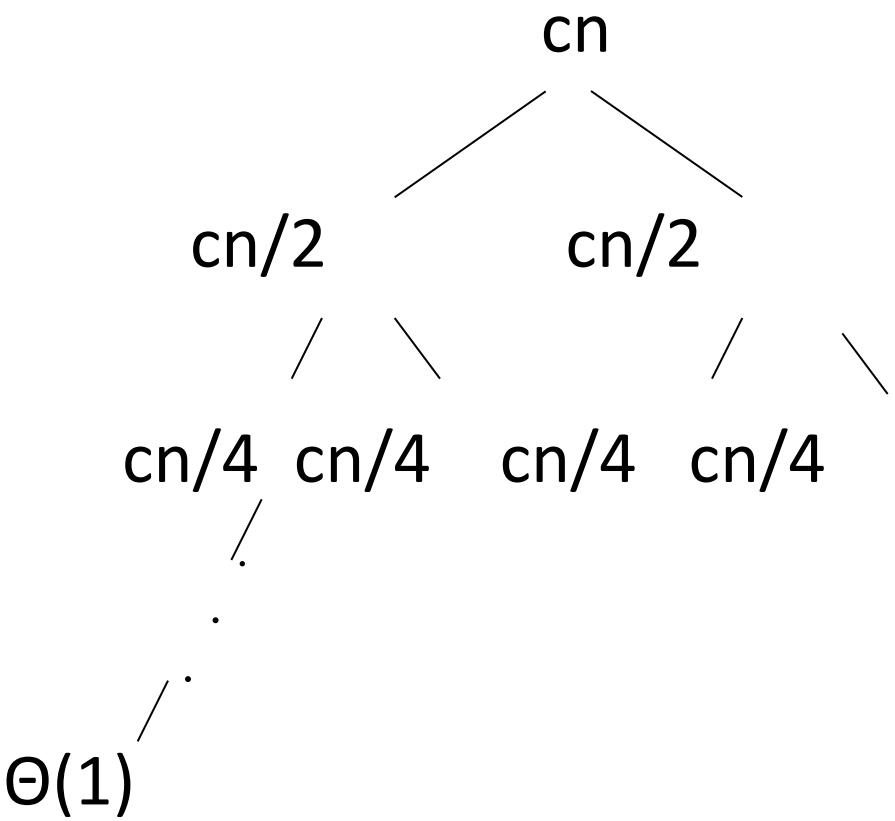
Recursion tree

Solve T(n) = 2T(n/2) + cn, where c > 0 is constant.



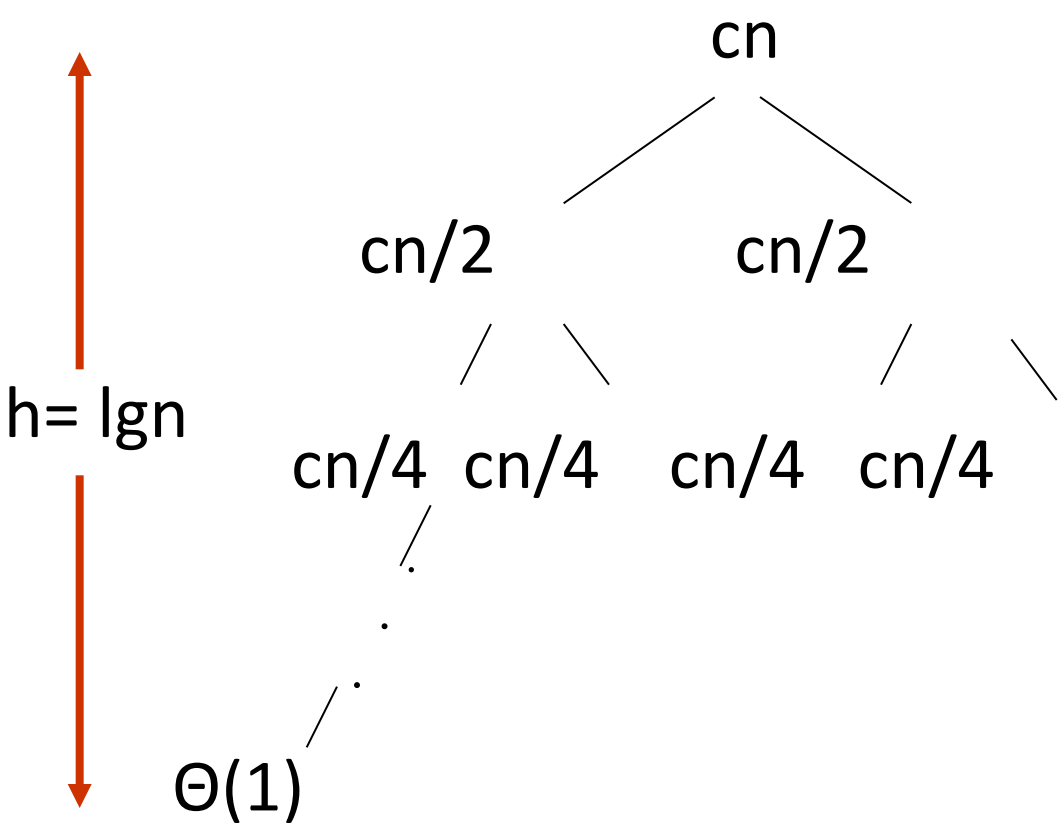
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



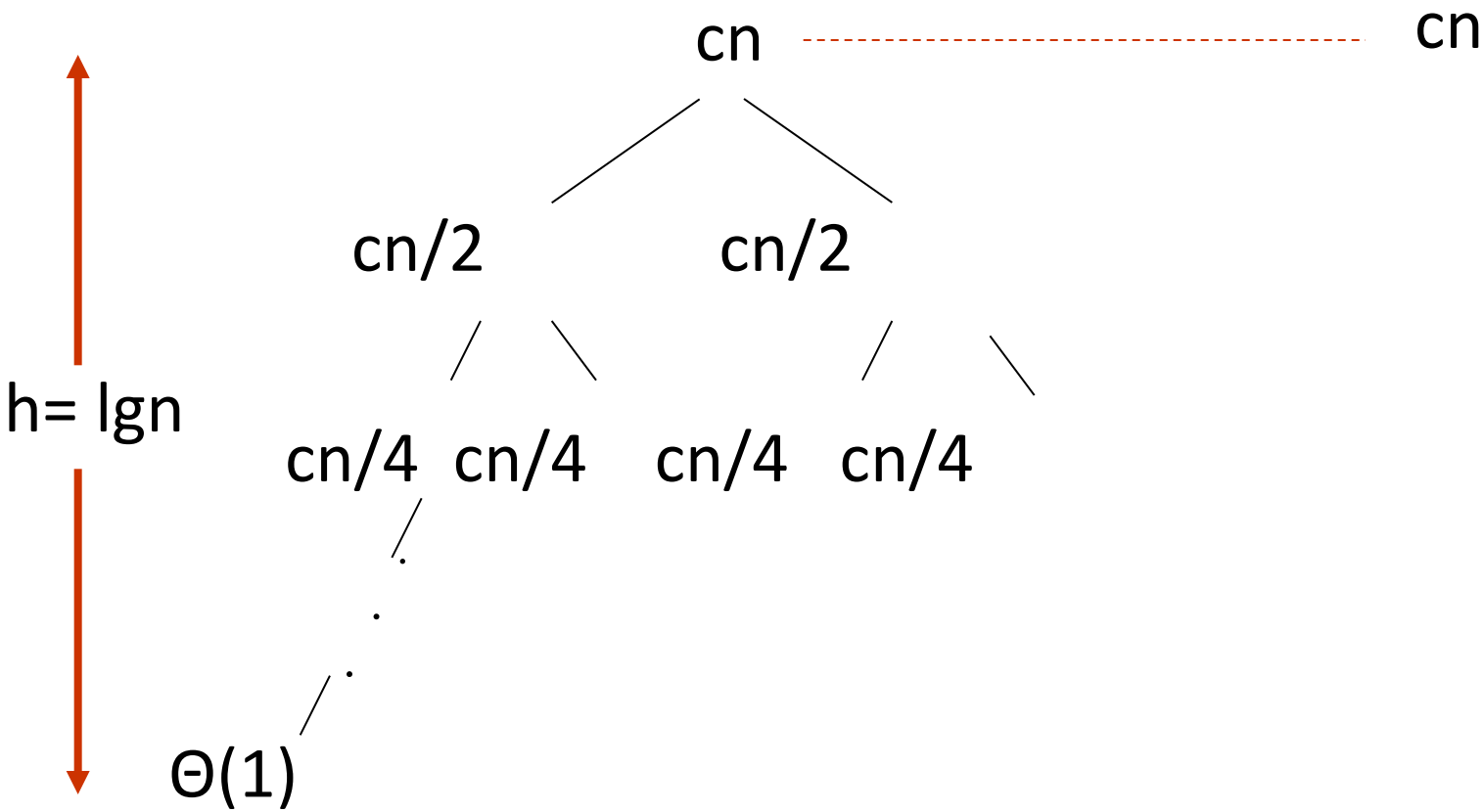
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



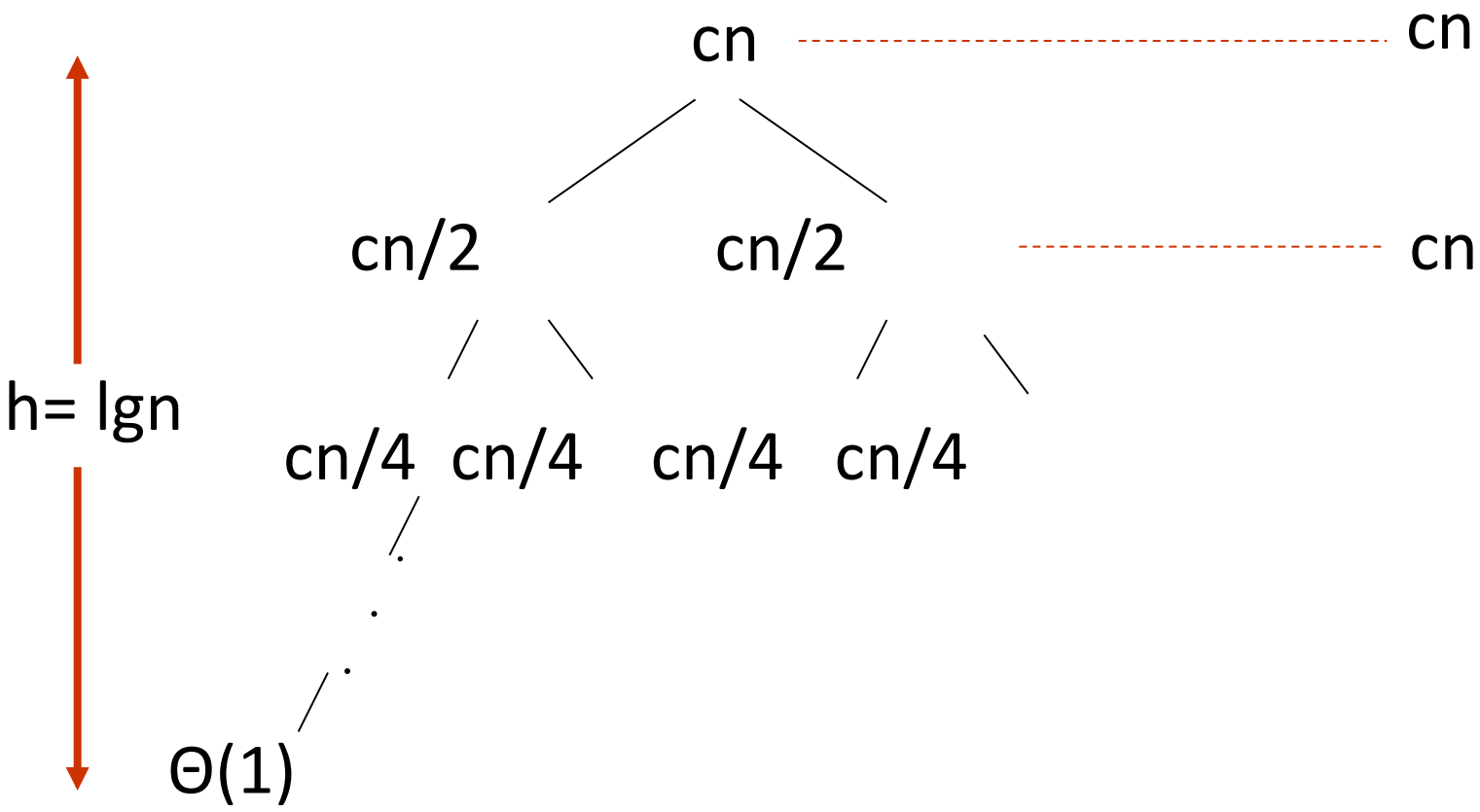
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



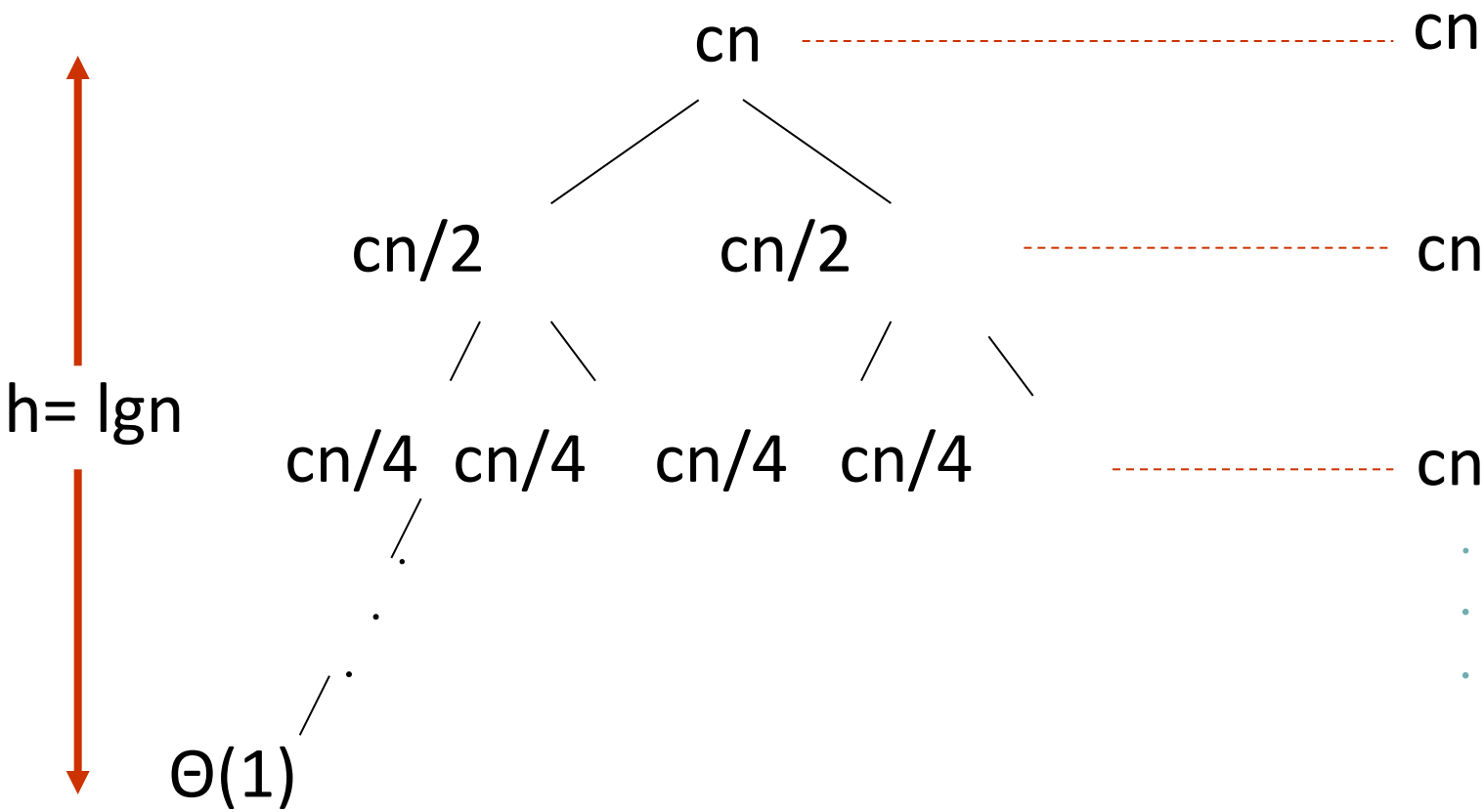
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



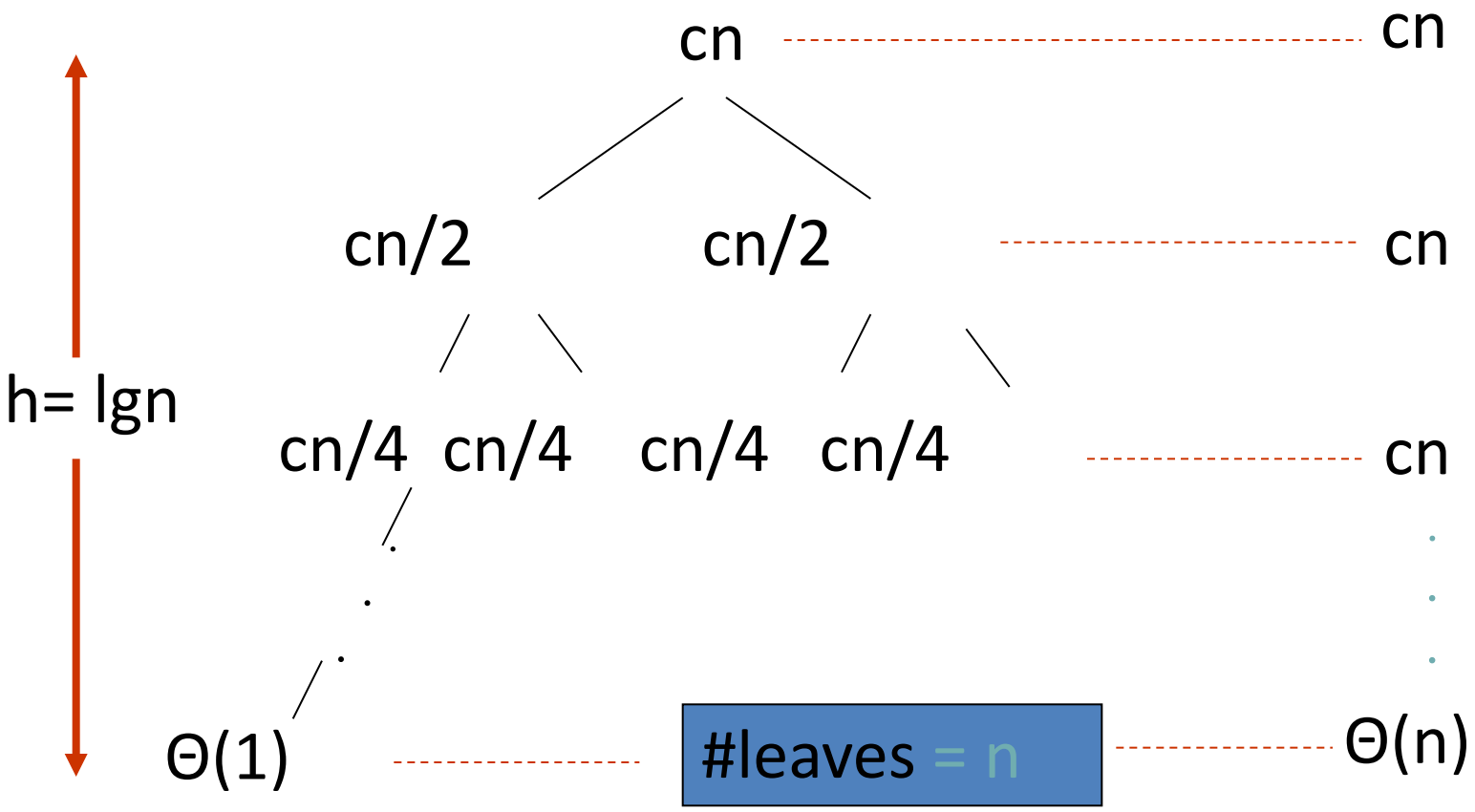
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



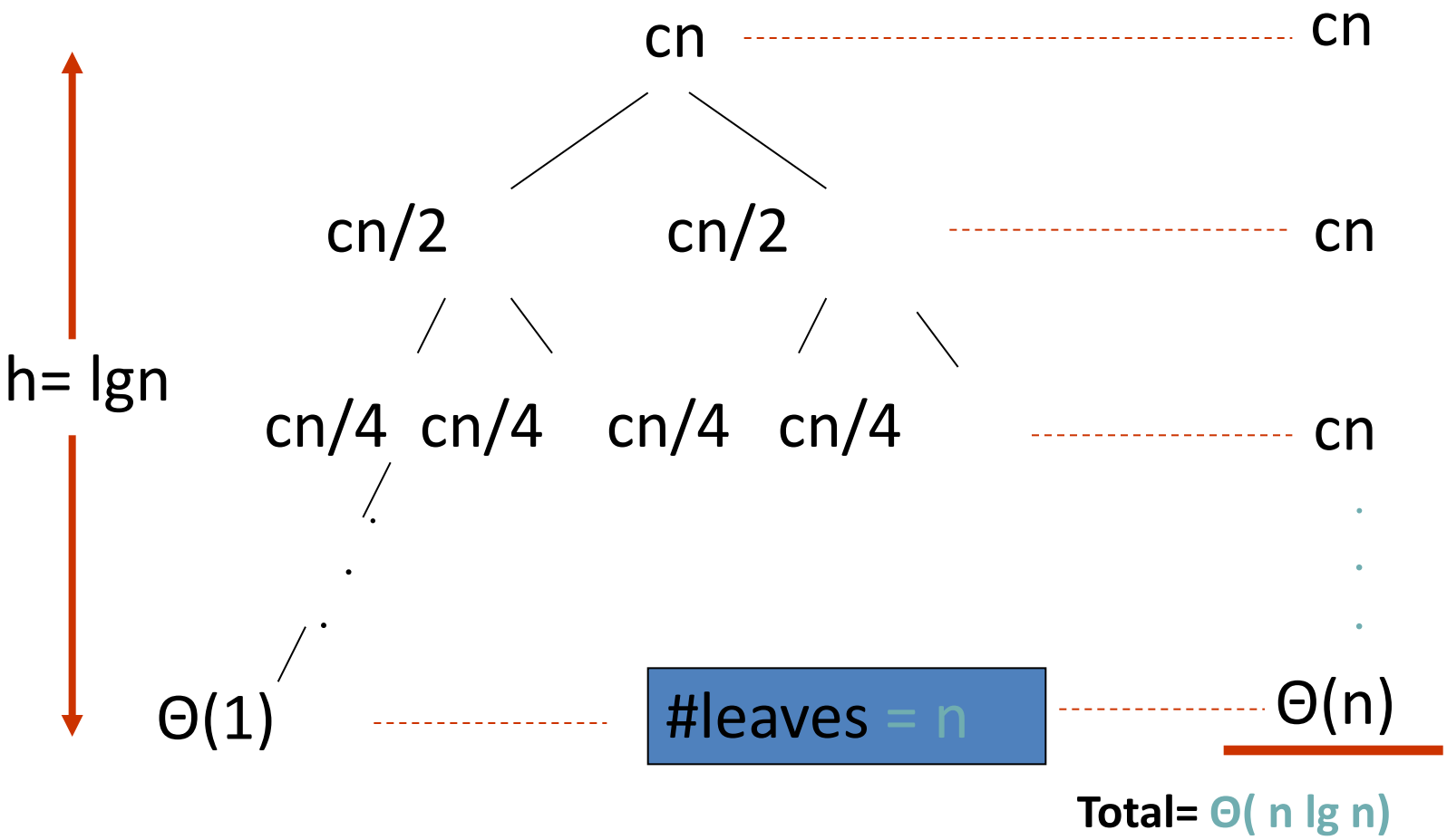
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



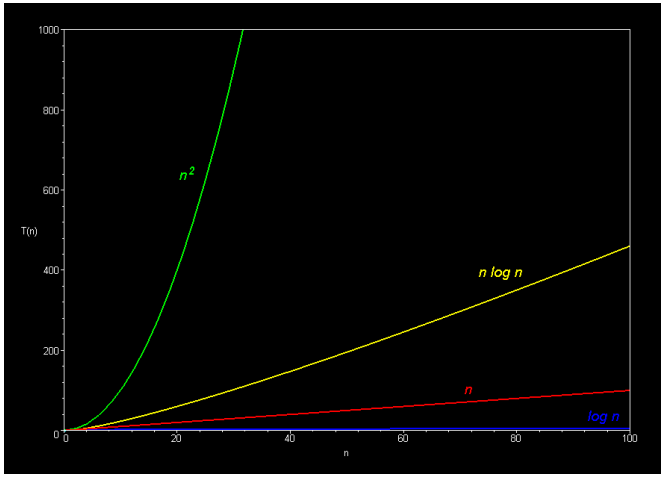
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Conclusions

- $\Theta(n \lg n)$ grows more slowly than $\Theta(n^2)$.
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so.
- Go test it out for yourself!



Quiz

The Tower Of Hanoi

Edouard Lucas - 1883

Once upon a time!

- The Tower of Hanoi (sometimes referred to as the Tower of Brahma or the End of the World Puzzle) was invented by the French mathematician, Edouard Lucas, in 1883. He was inspired by a legend that tells of a Hindu temple where the pyramid puzzle might have been used for the mental discipline of young priests.

What does legend say??

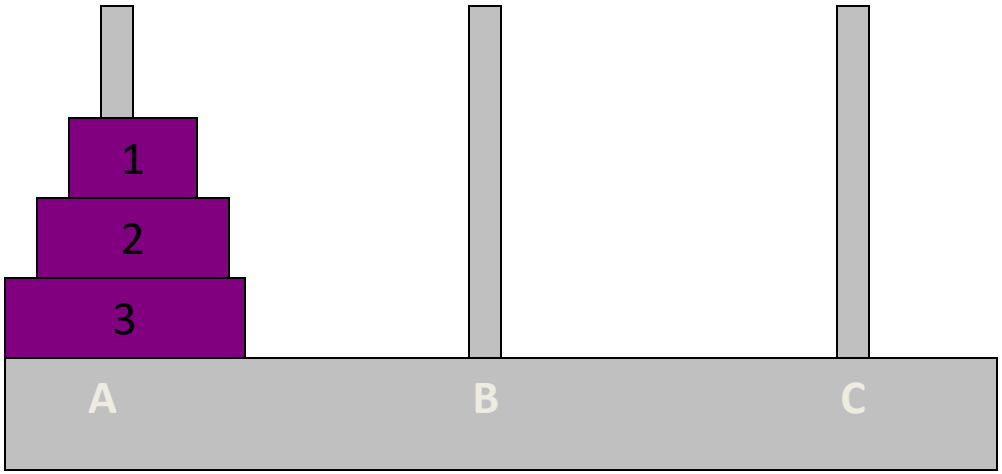
Legend says that at the beginning of time the priests in the temple were given a stack of 64 gold disks, each one a little smaller than the one beneath it. Their assignment was to transfer the 64 disks from one of three poles to another, with one important proviso a large disk could never be placed on top of a smaller one. The priests worked very efficiently, day and night. When they finished their work, the myth said, the temple would crumble into dust and the world would vanish. How many moves (& how long) *would* the priests need to take to complete the task??

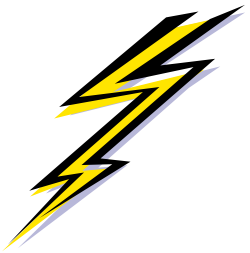
Start here - *Instructions*

1. Transfer all the disks from one pole to another pole.

2. You may move only ONE disk at a time.

3. A large disk may not rest on top of a smaller one at any time.





Fascinating fact

So the formula for finding the number of steps it takes to transfer n disks from post A to post C is:

$$2^n - 1$$

- The number of separate transfers of single disks the priests must make to transfer the tower is 2 to the 64th minus 1, or 18,446,744,073,709,551,615 moves! If the priests worked day and night, making one move every second it would take slightly more than 580 billion years to accomplish the job! - far, far longer than some scientists estimate the solar system will last.

Quiz Time

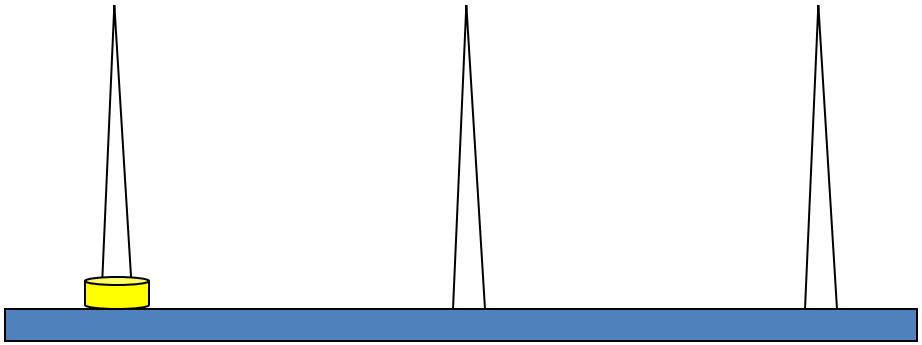
- Make the group 2
- Discuss the solution for the tower of Hanoi

....

Design

Basis: What is an instance of the problem that is trivial?

→ n == 1

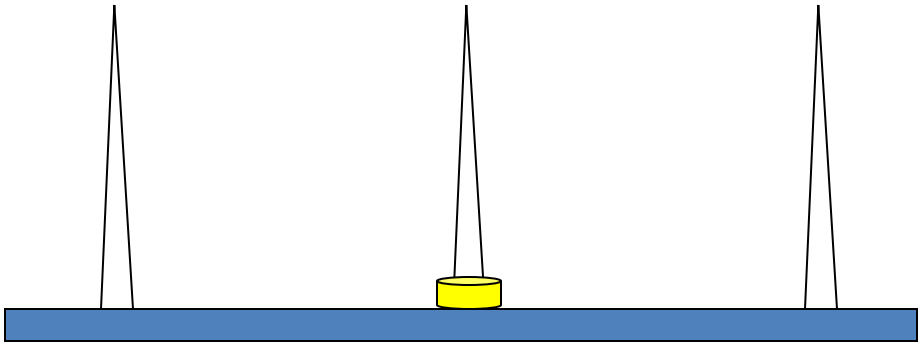


Since this base case could occur when the disk is on any needle, we simply output the instruction to move the top disk from *src* to *dest*.

Design

Basis: What is an instance of the problem that is trivial?

→ n == 1

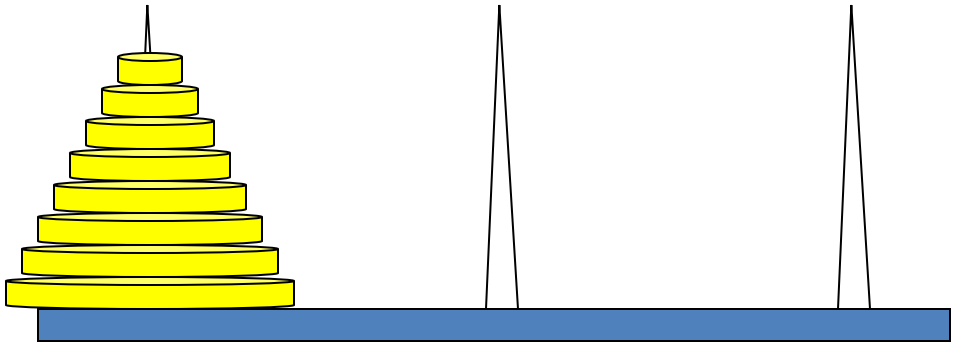


Since this base case could occur when the disk is on any needle, we simply output the instruction to move the top disk from *src* to *dest*.

Design (*Ct'd*)

Induction Step: n > 1

→ How can recursion help us out?

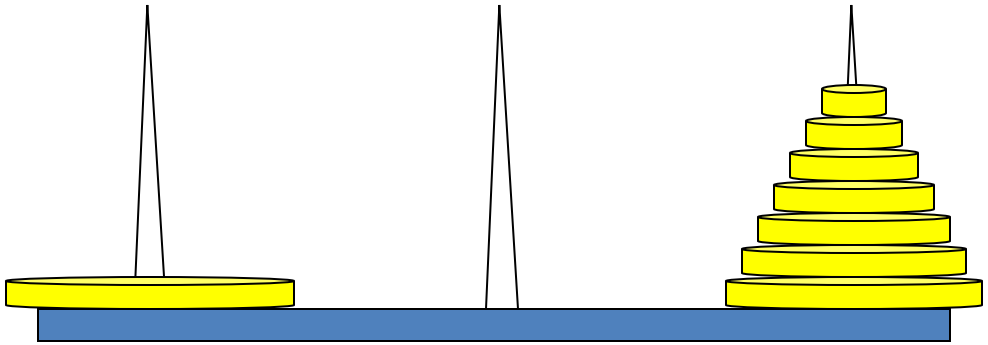


a. <u>Recursively</u> move n-1 disks from *src* to *aux*.

Design (*Ct'd*)

Induction Step: n > 1

→ How can recursion help us out?

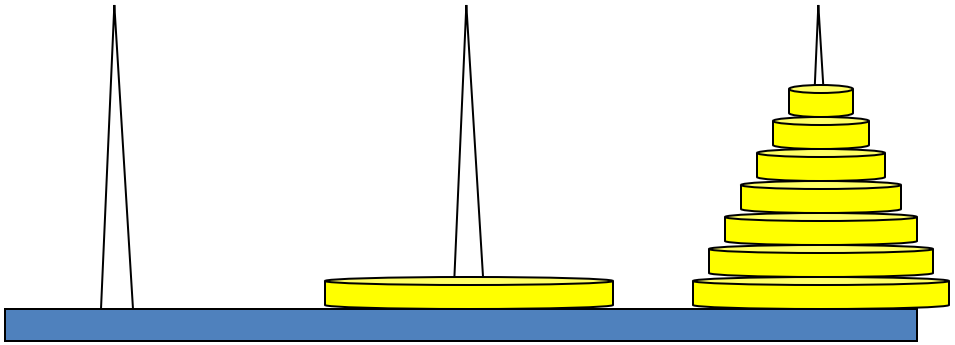


b. Move the one remaining disk from *src* to *dest*.

Design (*Ct'd*)

Induction Step: n > 1

$\rightarrow$ How can recursion help us out?

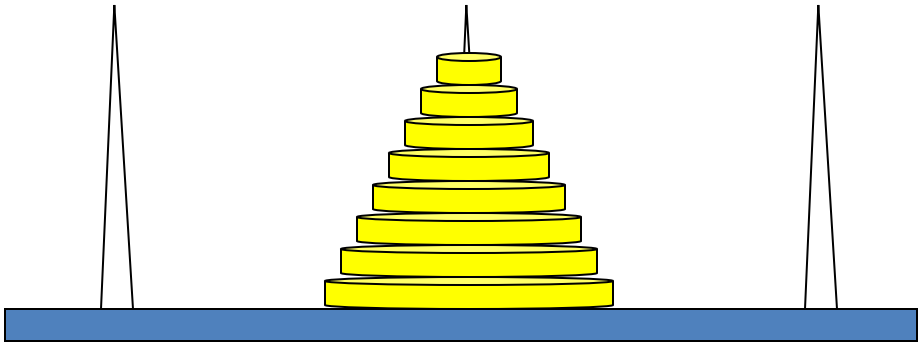


c. *Recursively* move n-1 disks from *aux* to *dest*...

Design (*Ct'd*)

Induction Step: n > 1

$\rightarrow$ How can recursion help us out?



d. We're done!

....

Algorithm

We can combine these steps into the following algorithm:

0. Receive *n, src, dest, aux*.
1. If *n* > 1:
   a. Move(*n-1, src, aux, dest*);
   b. Move(1, *src, dest, aux*);
   c. Move(*n-1, aux, dest, src*);

   Else

   Display “Move the top disk from “, *src*, “ to “, *dest*.

   End if.

Coding

```
// ...

void Move(int n, char src, char dest, char aux)
{
  if (n > 1)
  {
    Move(n-1, src, aux, dest);
    Move(1, src, dest, aux);
    Move(n-1, aux, dest, src);
  }
  else
    cout << "Move the top disk from "
         << src << " to " << dest << endl;
}
```

Testing

```
The Hanoi Towers

Enter how many disks: 1
Move the top disk from A to B
```

Testing (*Ct'd*)

```
The Hanoi Towers

Enter how many disks: 2
Move the top disk from A to C
Move the top disk from A to B
Move the top disk from C to B
```

Testing (*Ct'd*)

```
The Hanoi Towers

Enter how many disks: 3
Move the top disk from A to B
Move the top disk from A to C
Move the top disk from B to C
Move the top disk from A to B
Move the top disk from C to A
Move the top disk from C to B
Move the top disk from A to B
```

Testing (*Ct'd*)

```
The Hanoi Towers

Enter how many disks: 4
move a disk from needle A to needle B
move a disk from needle C to needle B
move a disk from needle A to needle C
move a disk from needle B to needle A
move a disk from needle B to needle C
move a disk from needle A to needle C
move a disk from needle A to needle B
move a disk from needle C to needle B
move a disk from needle C to needle A
move a disk from needle B to needle A
move a disk from needle C to needle B
move a disk from needle A to needle C
move a disk from needle A to needle B
move a disk from needle C to needle B
```

Analysis

Let's see how many moves" it takes to solve this problem, as a function of *n*, the number of disks to be moved.

n	Number of disk-moves required
1	1
2	3
3	7
4	15
5	31
...	
i	2^i-1
64	2^{64}-1 (a big number)