# Searching and Sorting

- Searching
  - Linear search
  - Binary search
- Sorting
  - Insertion sort
  - Merge sort

Philip Bille

# Searching and Sorting

- Searching
  - Linear search
  - Binary search
- Sorting
  - Insertion sort
  - Merge sort

# Searching

- Searching. Given a sorted array A and number x, determine if x appears in the array.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 |

# Linear Search

- Linear search. Check if each entry matches x.
- Time?
- Challenge. Can we take advantage of the sorted order of the array?

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 |

# Binary Search

- Binary search. Compare x to middle entry m in A.
  - if A[m] = x return true and stop.
  - if A[m] < x continue recursively on the right half.
  - if A[m] > x continue recursively on the left half.
- If array size ≤ 0 return false and stop.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 |

# Binary Search

```
BINARYSEARCH(A,i,j,x)
  if j < i return false
  m = ⌊(i+j)/2⌋
   if A[m] = x return true
   elseif A[m] < x return BINARYSEARCH(A,m+1,j,x)
   else return BINARYSEARCH(A,i,m-1,x)     // A[m] > x
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 |

- Time?
- Analysis 1. Analogue of recursive peak algorithm.
  - A recursive call takes constant time.
  - Each recursive call halves the size of the array. We stop when the size is $\leq 0$.
  - $\Rightarrow$ Running time is $O(\log n)$

# Binary Search

- Analysis 2. Let T(n) be the running time for binary search.
  - Solve the recurrence relation for T(n).

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

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c \\ &= T\left(\frac{n}{4}\right) + c + c \\ &= T\left(\frac{n}{8}\right) + c + c + c \\ &\vdots \\ &= T\left(\frac{n}{2^k}\right) + ck \\ &\vdots \\ &= T\left(\frac{n}{2^{\log_2 n}}\right) + c\log_2 n \\ &= T(1) + c\log_2 n \\ &= d + c\log_2 n \\ &= O(\log n) \end{aligned}$$

# Searching

- We can search in
  - O(n) time with linear search.
  - O(log n) time with binary search.

# Searching and Sorting

- Searching
  - Linear search
  - Binary search
- Sorting
  - Insertion sort
  - Merge sort

# Sorting

- Sorting. Given array A[0..n-1] return array B[0..n-1] with same values as A but in sorted order.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 33 | 4 | 25 | 28 | 45 | 18 | 7 | 12 | 36 | 1 | 47 | 42 | 50 | 16 | 31 |

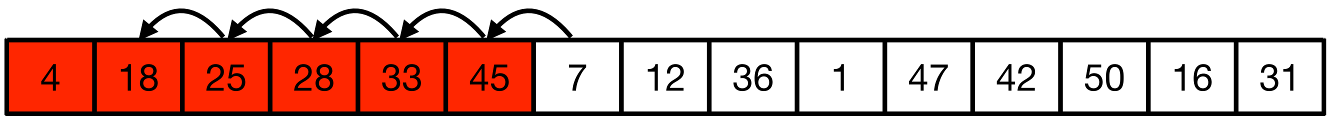
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 |

# Applications

- Obvious.
  - Sort list of names, show Google PageRank results, show social media feed in chronological order.
- Non obvious.
  - Data compression, computer graphics, bioinformatics, recommendations systems.
- Easy problem for sorted data.
  - Search, find median, find duplicates, find closest pair, find outliers.

# Insertion Sort

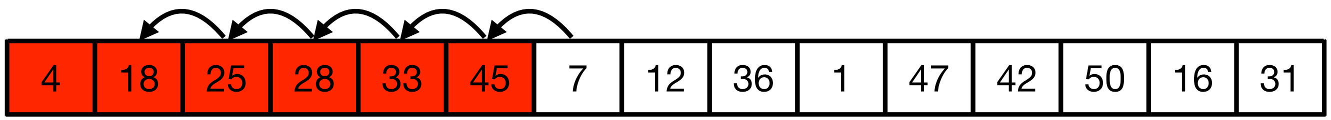
- Insertion sort. Start with unsorted array A.
- Proceed left-to-right in n rounds.
- Round i:
  - Subarray A[0..i-1] is sorted.
  - Insert A[i] into A[0..i-1] to make A[0..i] sorted.



| | | | | | |
|---|---|---|---|---|---|
| 33 | 4 | 25 | 5 | 20 | 45 |

# Insertion Sort

```
INSERTIONSORT(A, n)
   for i = 1 to n-1
      j = i
      while j > 0 and A[j-1] > A[j]
         swap A[j] og A[j-1]
         j = j - 1
```




- Time?
  - To insert A[i] we use c·i time for constant c.
  - $\Longrightarrow$ total time T(n):

$$T(n) = \sum_{i=1}^{n-1} ci = c\sum_{i=1}^{n-1} i = \frac{cn(n-1)}{2} = O(n^2)$$

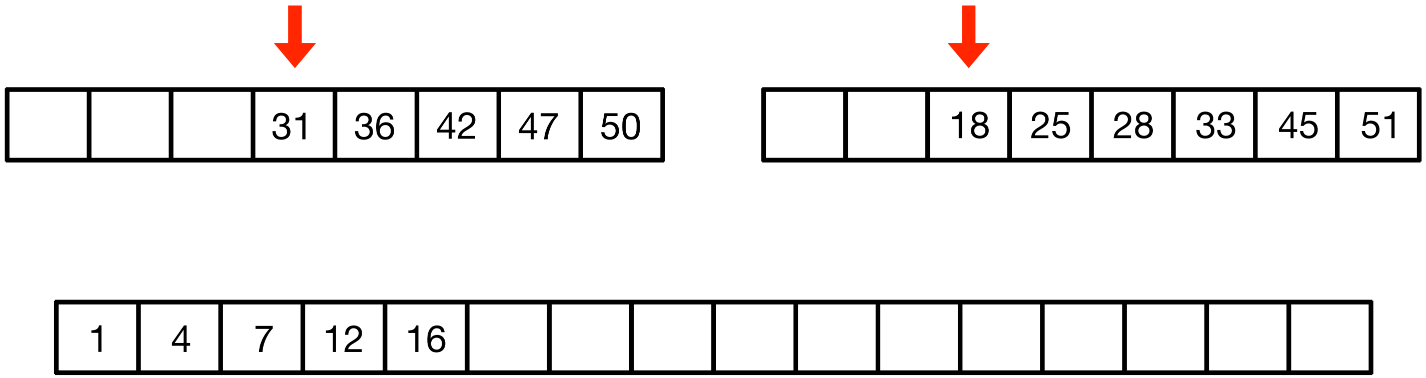
- Challenge. Can we sort faster?

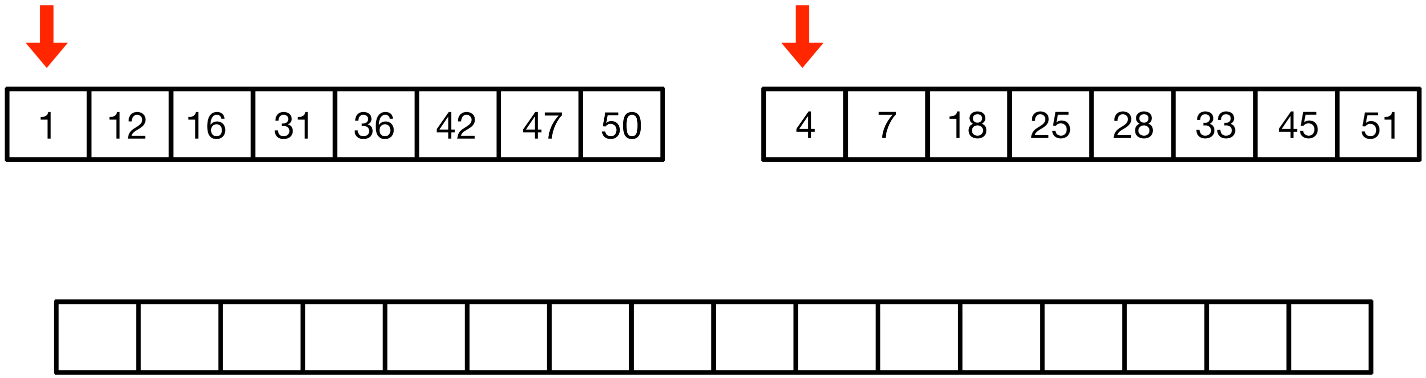
# Merge sort

- Merge sort.
  - Idea. Recursive sorting via merging sorted subarrays.

# Merge

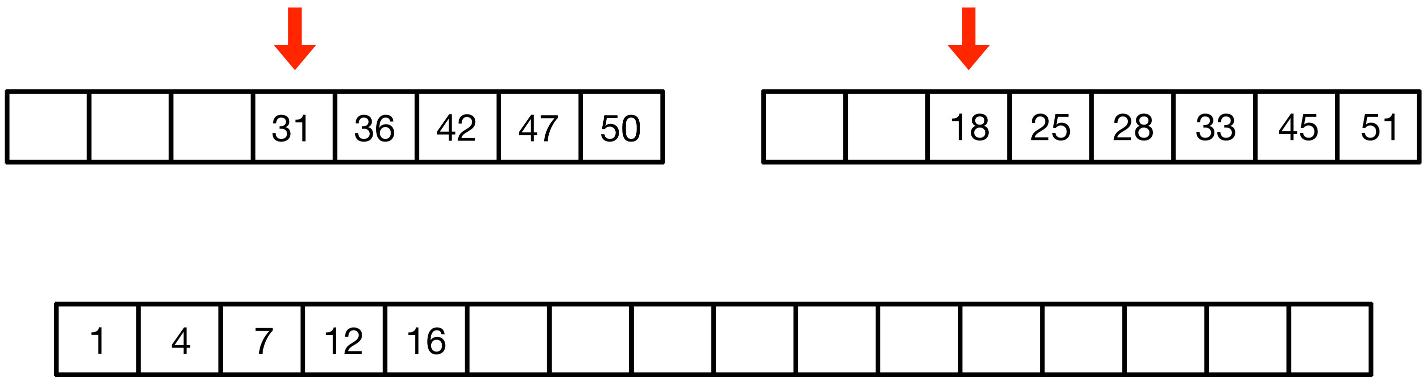
- Goal. Combine two sorted arrays into a single sorted array.
- Idea.
  - Scan both arrays left-to-right. In each step:
    - Insert smallest of the two entries in new array.
    - Move forward in array with smallest entry.
    - Repeat until input arrays are exhausted.




1 12 16 31 36 42 47 50
4 7 18 25 28 33 45 51

# Merge

- Time. Merging two arrays $A_1$ og $A_2$?
  - Each step take O(1) time.
  - Each step we move forward in one array.
  - $\Rightarrow O(|A_1| + |A_2|)$ time.



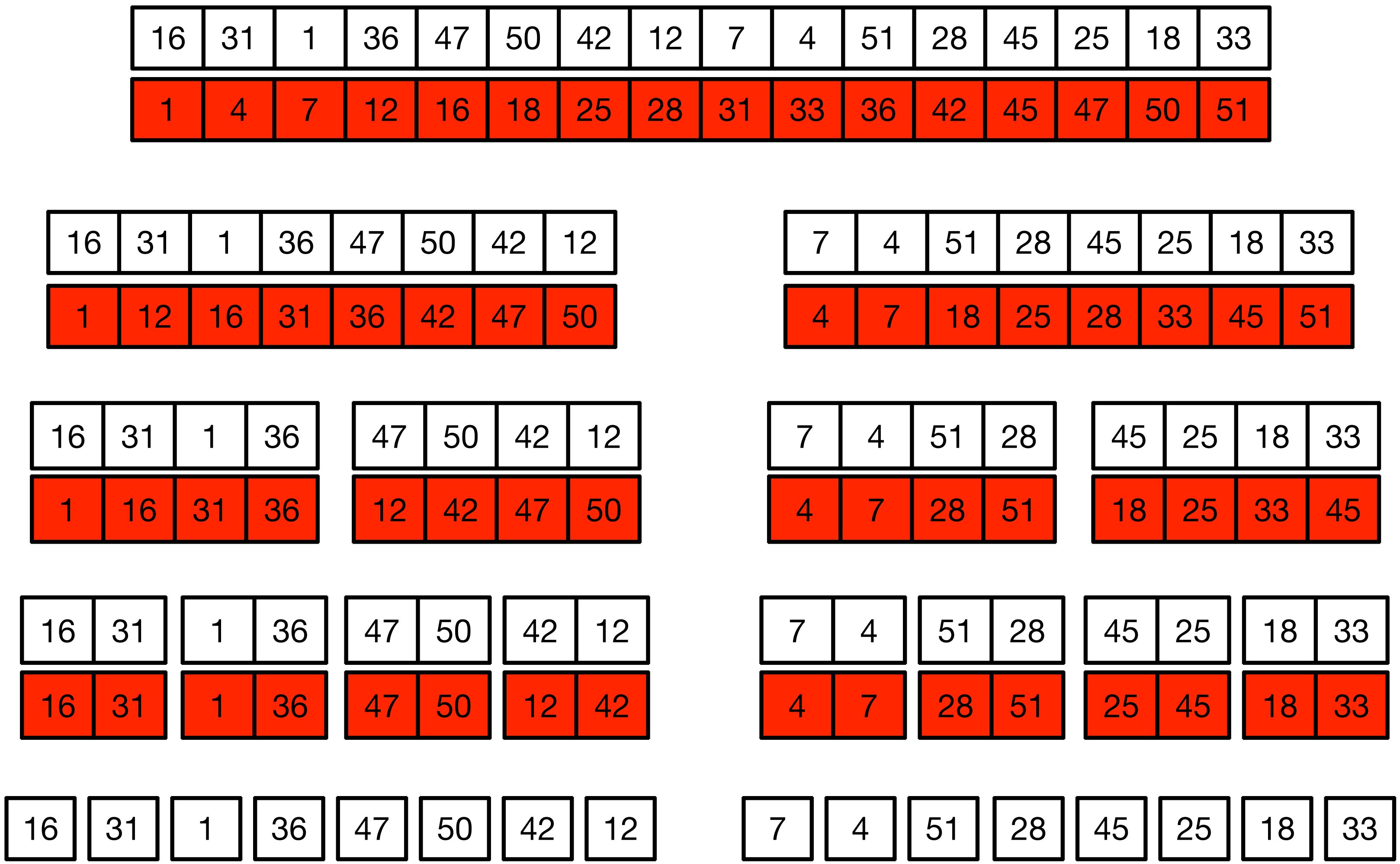
# Merge Sort

- Merge sort.
- If |A| ≤ 1, return A.
- Otherwise:
  - Split A into halves.
  - Sort each half recursively.
  - Merge the two halves.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 16 | 31 | 1 | 36 | 47 | 50 | 42 | 12 | 7 | 4 | 51 | 28 | 45 | 25 | 18 | 33 |
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 | 51 |

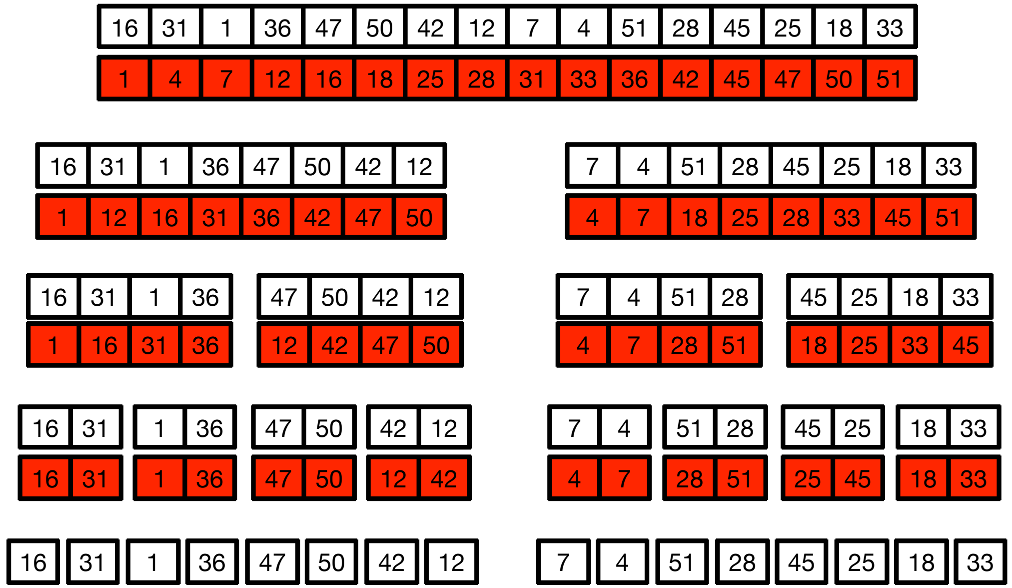
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 31 | 1 | 36 | 47 | 50 | 42 | 12 |
| 1 | 12 | 16 | 31 | 36 | 42 | 47 | 50 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 4 | 51 | 28 | 45 | 25 | 18 | 33 |
| 4 | 7 | 18 | 25 | 28 | 33 | 45 | 51 |



16 31 1 36 47 50 42 12 7 4 51 28 45 25 18 33
1 4 7 12 16 18 25 28 31 33 36 42 45 47 50 51
16 31 1 36 47 50 42 12
1 12 16 31 36 42 47 50
7 4 51 28 45 25 18 33
4 7 18 25 28 33 45 51
16 31 1 36
1 16 31 36
47 50 42 12
12 42 47 50
7 4 51 28
4 7 28 51
45 25 18 33
18 25 33 45
16 31
16 31
1 36
1 36
47 50
47 50
42 12
12 42
7 4
4 7
51 28
28 51
45 25
25 45
18 33
18 33
16 31 1 36 47 50 42 12
7 4 51 28 45 25 18 33

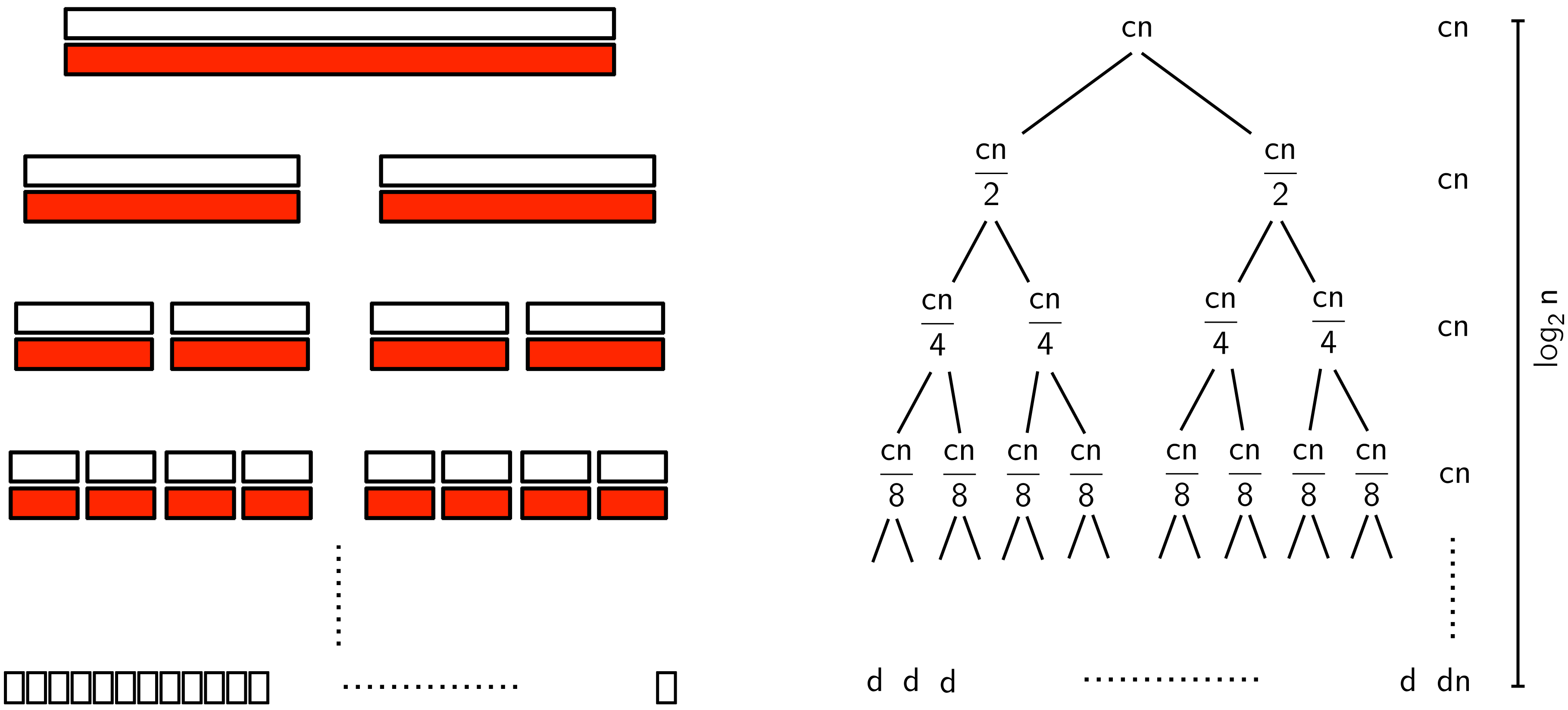
# Merge Sort




```
MERGESORT(A,i,j)
   if i < j
      m =⌊(i+j)/2⌋
      MERGESORT(A,i,m)
      MERGESORT(A,m+1,j)
      MERGE(A, i, m, j)
```

- Time?
- Construct recursion tree.

# Merge Sort




$$T(n) = cn\log_2 n + dn = O(n \log n)$$

# Sorting

- We can sort in
  - $O(n^2)$ time with insertion sort.
  - $O(n\log n)$ time with merge sort.

# Divide and Conquer

- Merge sort is example of a divide and conquer algorithm.
- Algorithmic design paradigm.
  - Divide. Split problem into subproblems.
  - Conquer. Solve subproblems recursively.
  - Combine. Combine solution for subproblem to a solution for problem.
- Merge sort.
  - Divide. Split array into halves.
  - Conquer. Sort each half.
  - Combine. Merge halves.

# Searching and Sorting

- Searching
  - Linear search
  - Binary search
- Sorting
  - Insertion sort
  - Merge sort