CS 161: Design and Analysis of Algorithms

Midterm

- Wednesday, July 25th in class 2:15 – 3:30
- Covers material through today
- No bluebooks needed
- SCPD students:
  - Can take exam on campus, let us know by Monday
  - Otherwise, must take at scheduled time with exam proctor

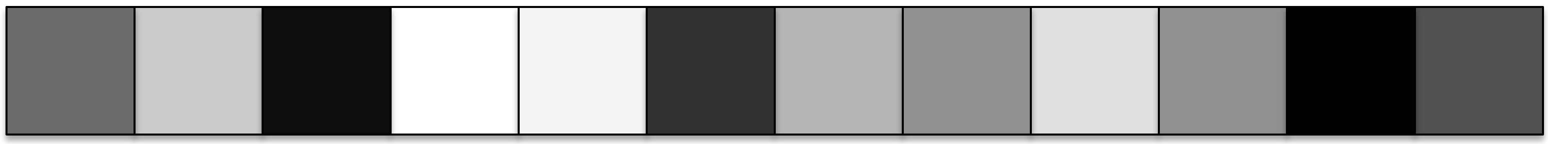
Divide & Conquer II: Sorting/Median Finding

- Merge Sort
- Quick Sort
- Sorting Lower Bound
- Median Finding

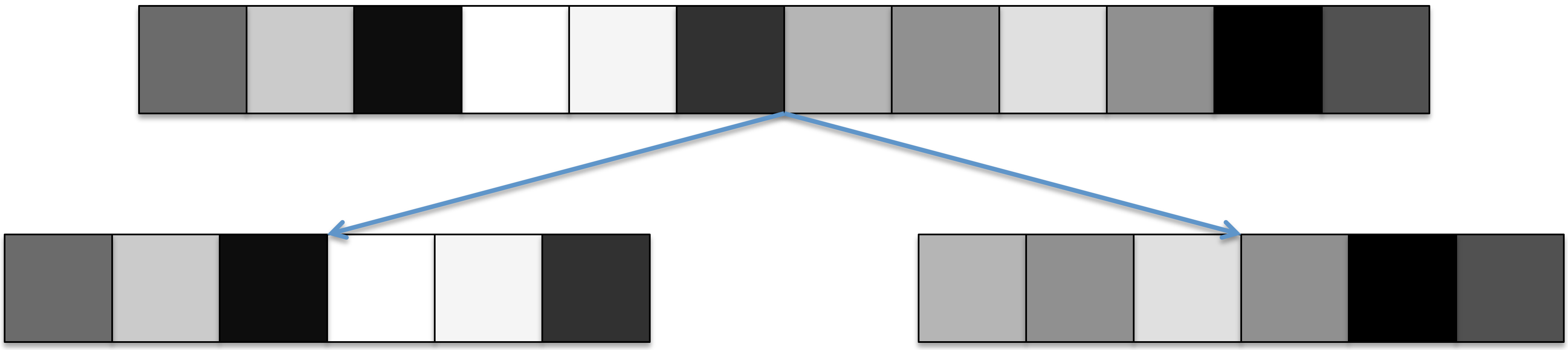
Merge Sort

- Want to sort a list of n elements
- Divide and conquer approach:
  - Split list into two sublists of size n/2
  - Recursively sort each sublist
  - Construct sorted list by merging sorted sublists

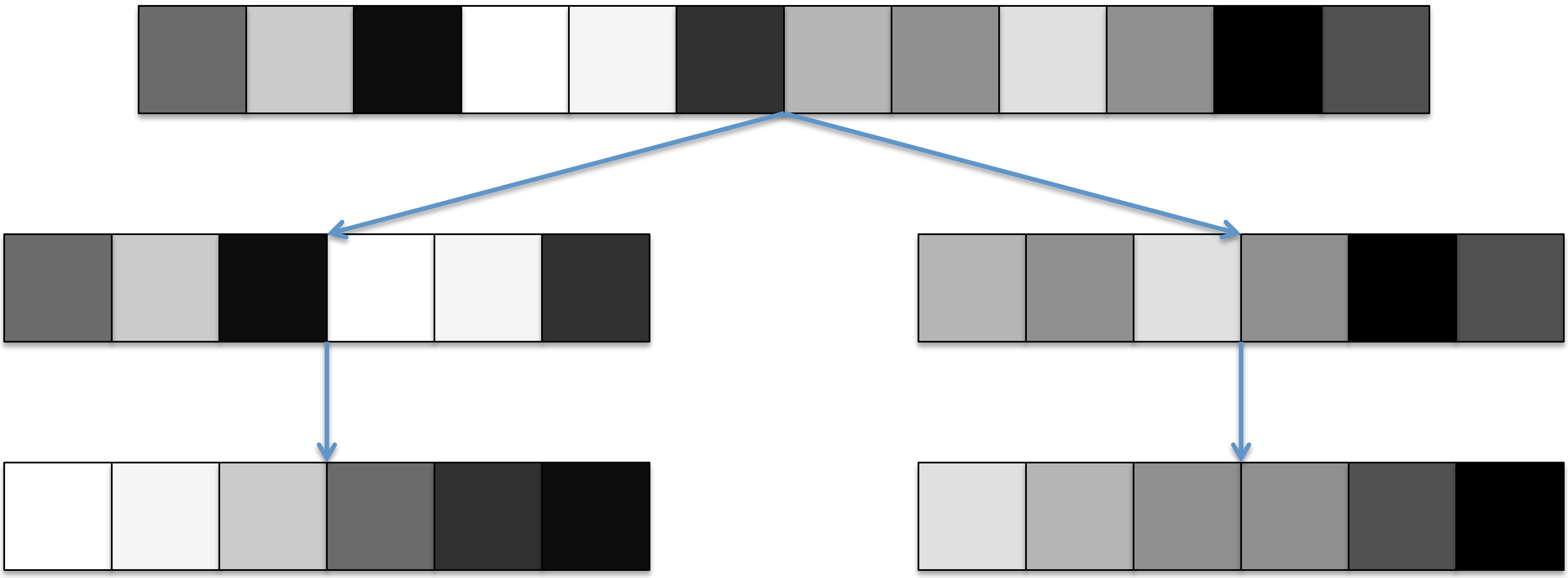
Merge Sort



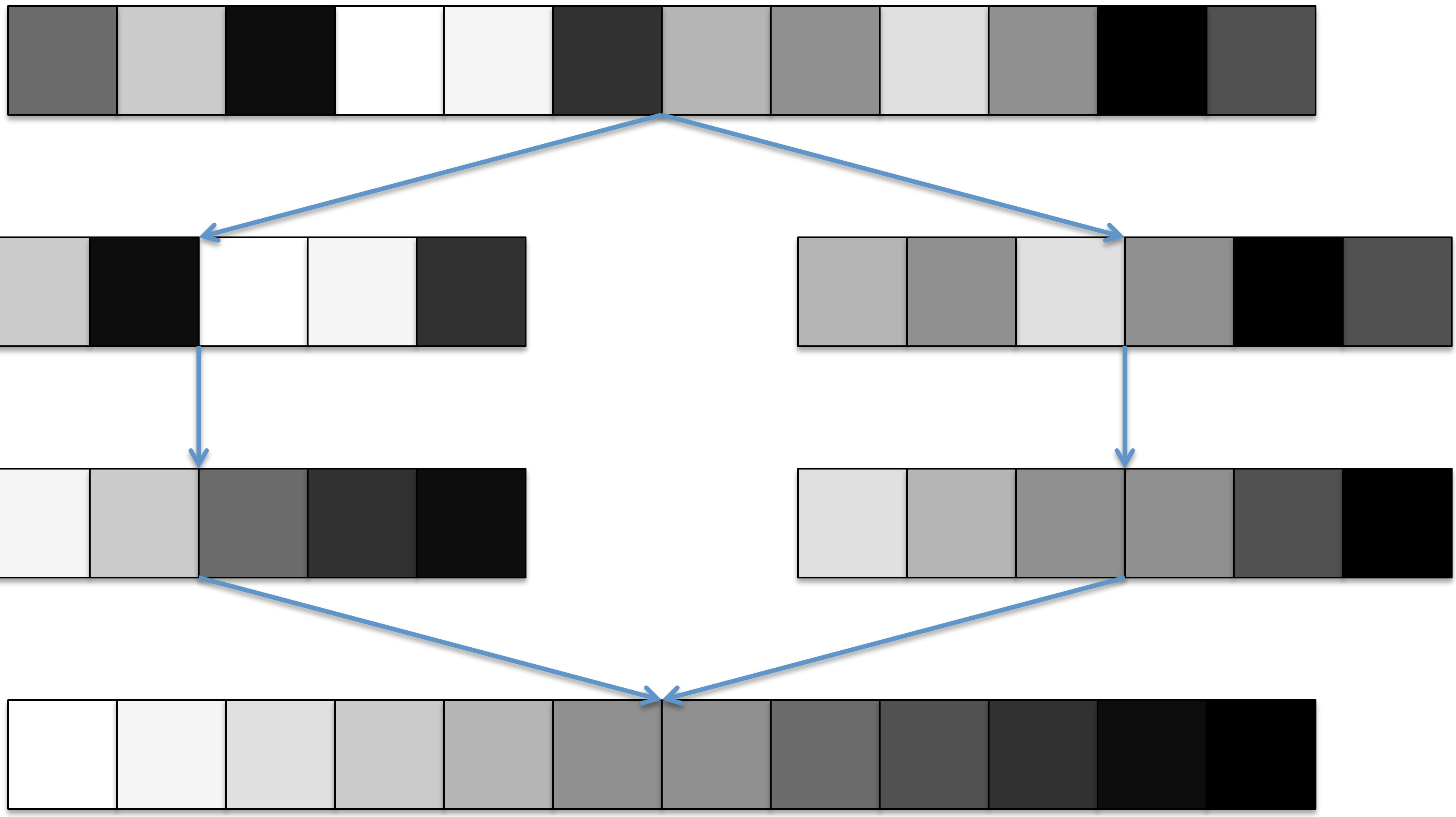
Merge Sort



Merge Sort



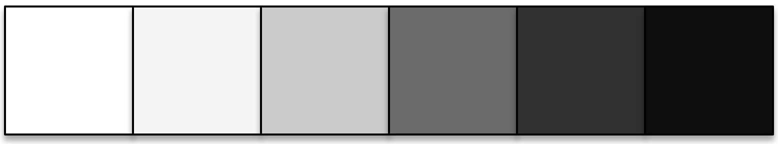
Merge Sort

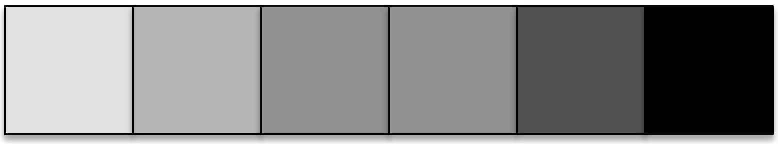


Merge Sort

- Splitting the list: Easy! O(n)
- Two recursive calls: Easy!
- Merging two sorted lists?
  - Lowest element in merged list is the lowest element of one of the lists
  - Pick smaller of the first elements of the two lists, remove it, and add it to the final list.

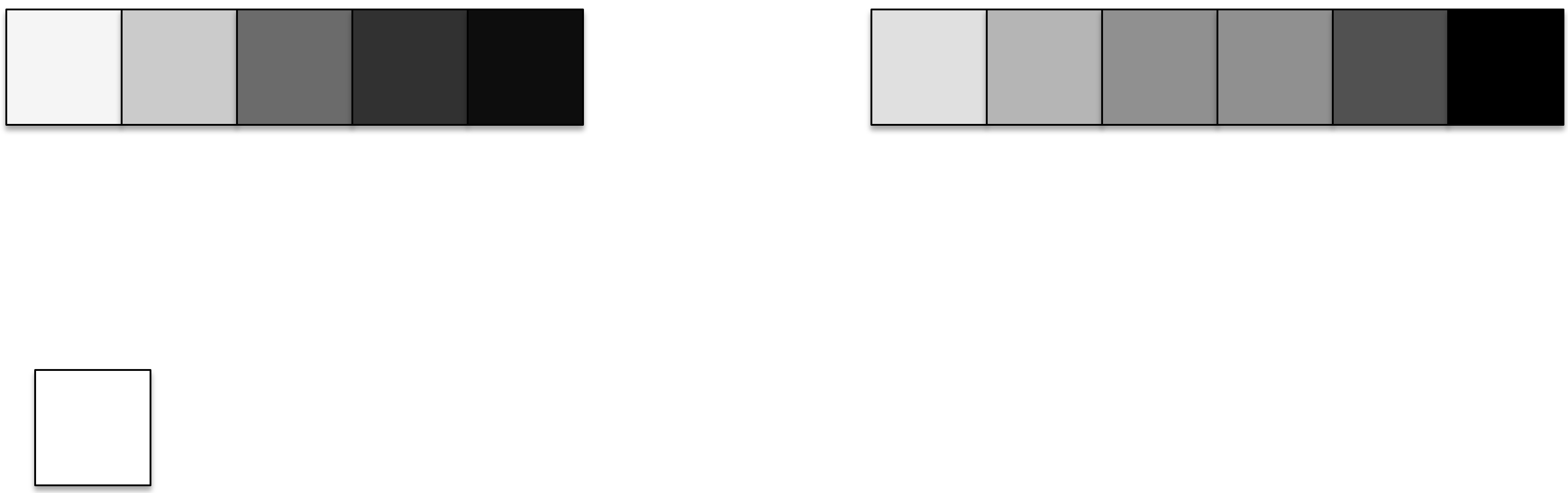
Merge Sort



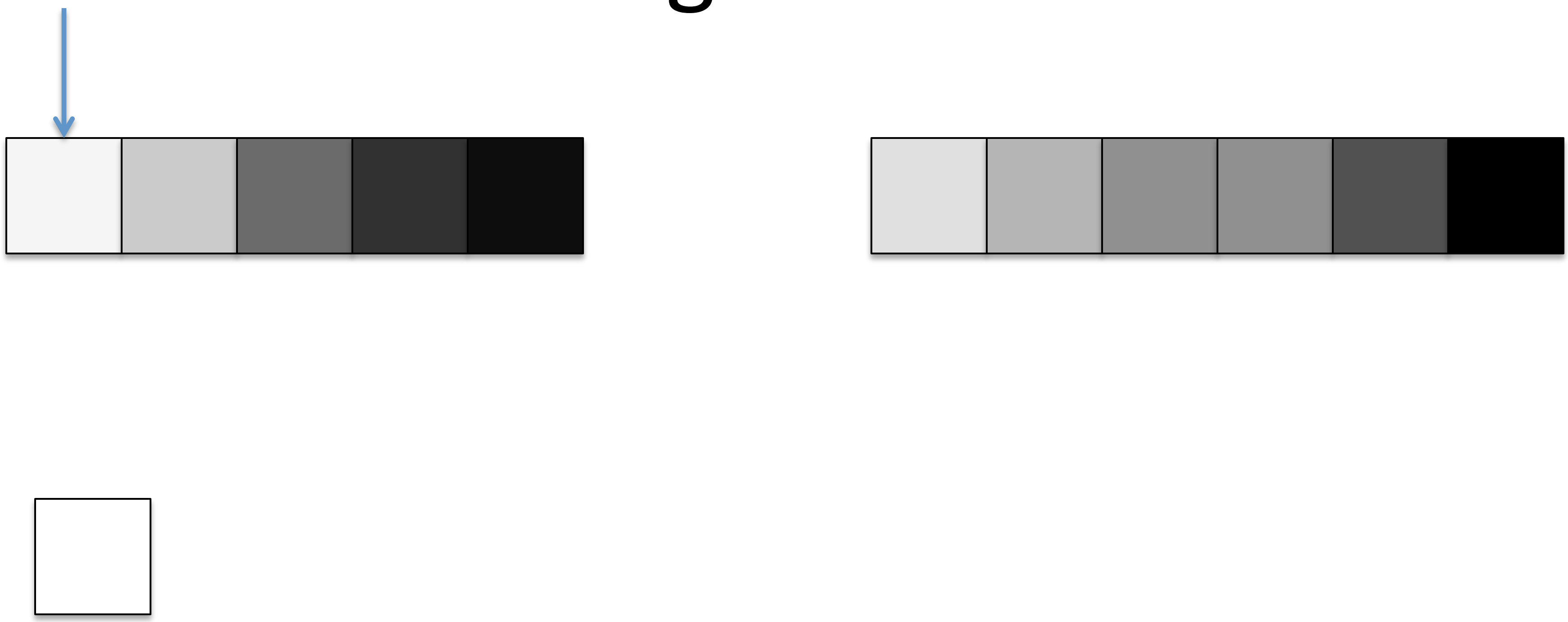


Merge Sort

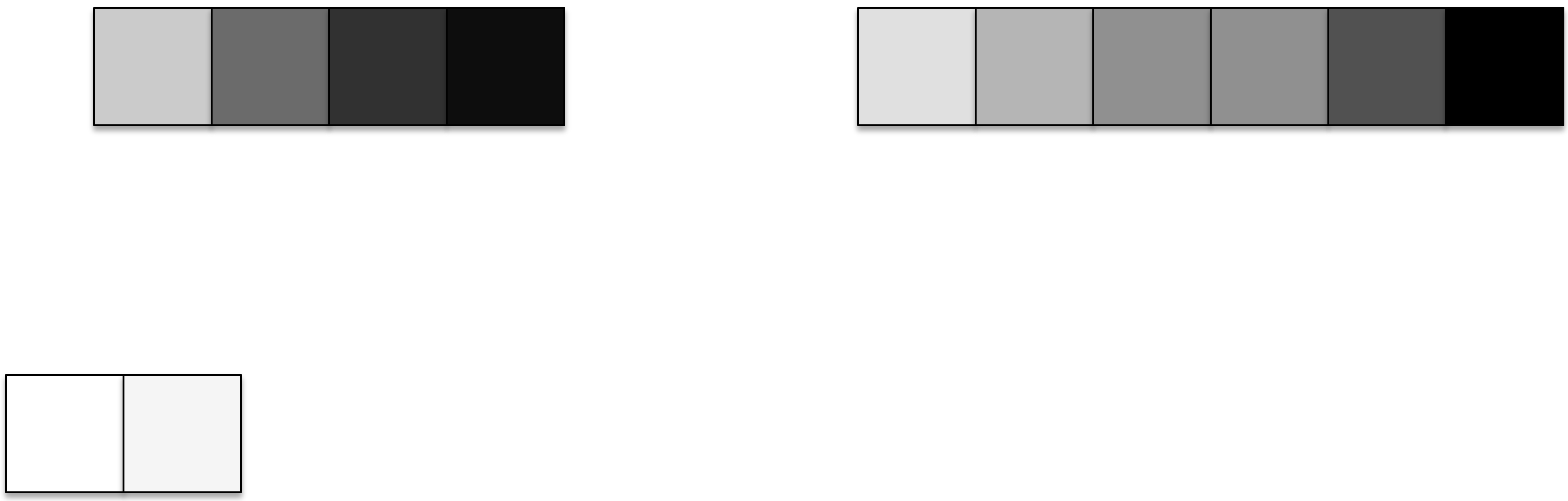
Merge Sort



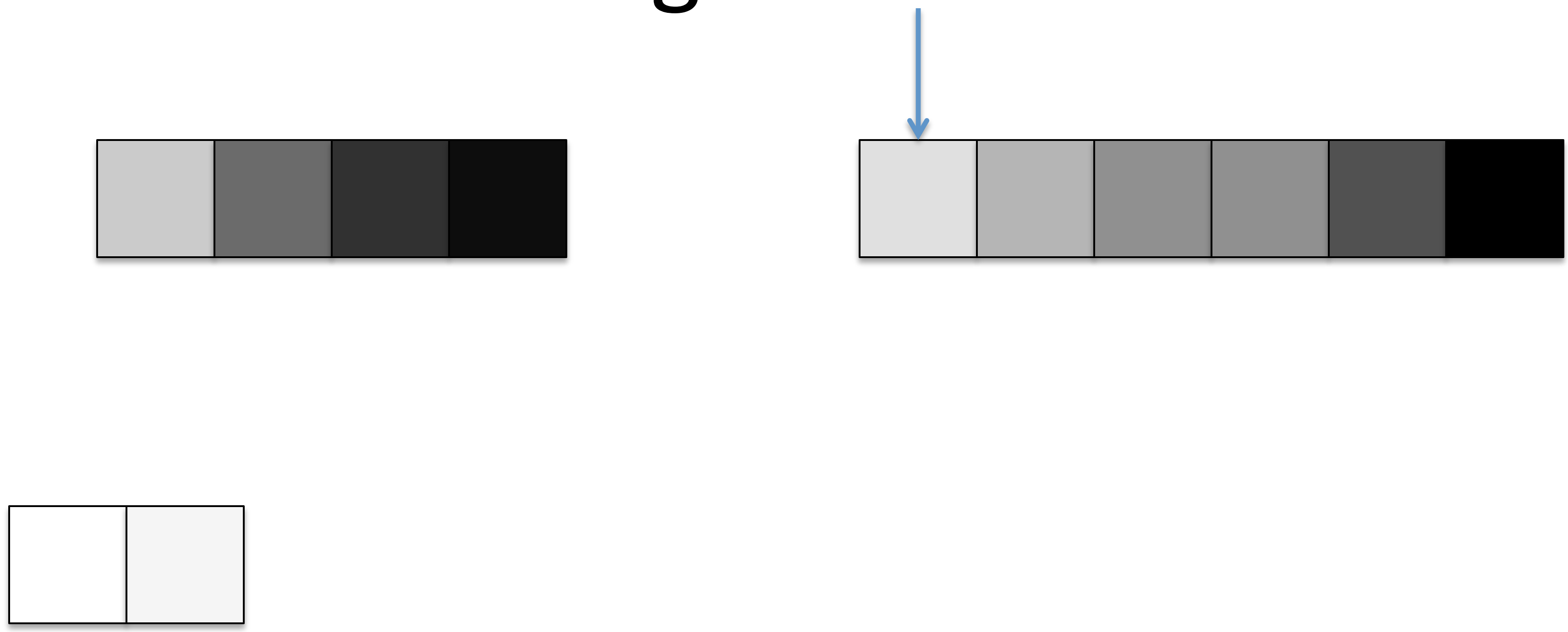
Merge Sort



Merge Sort

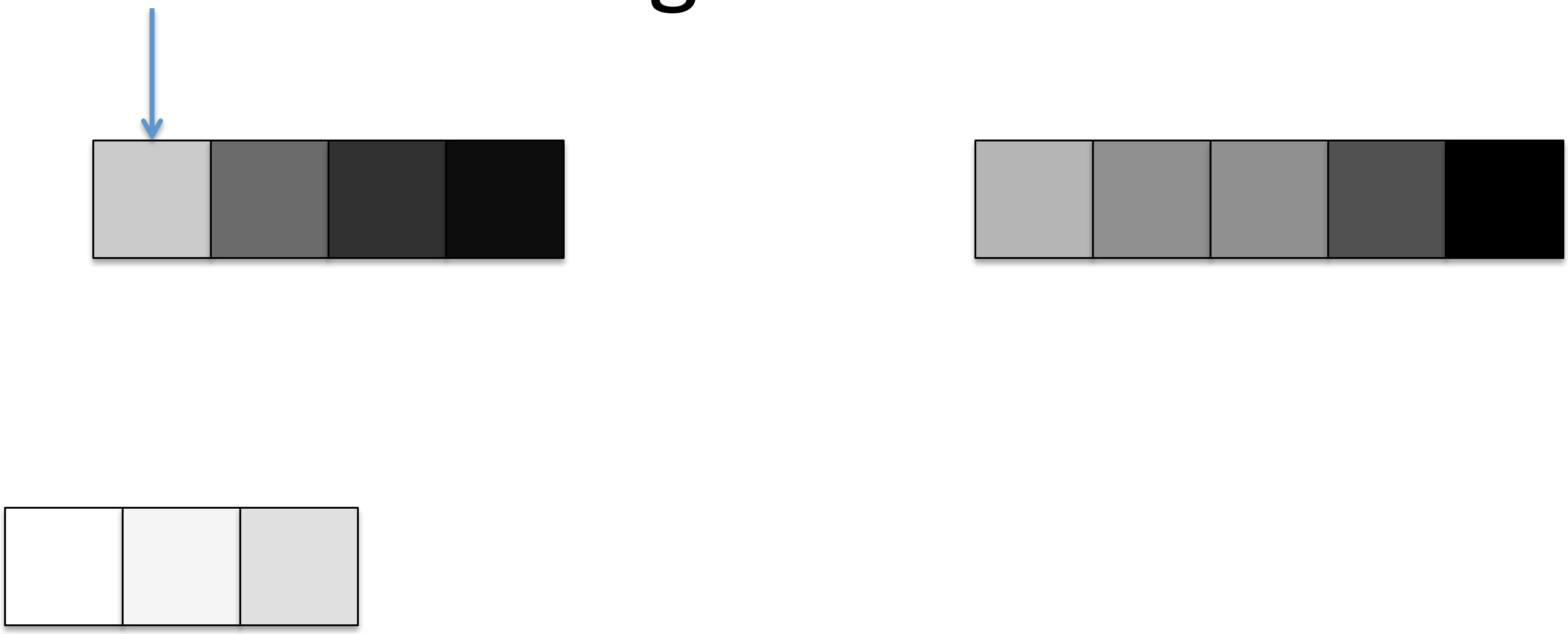


Merge Sort



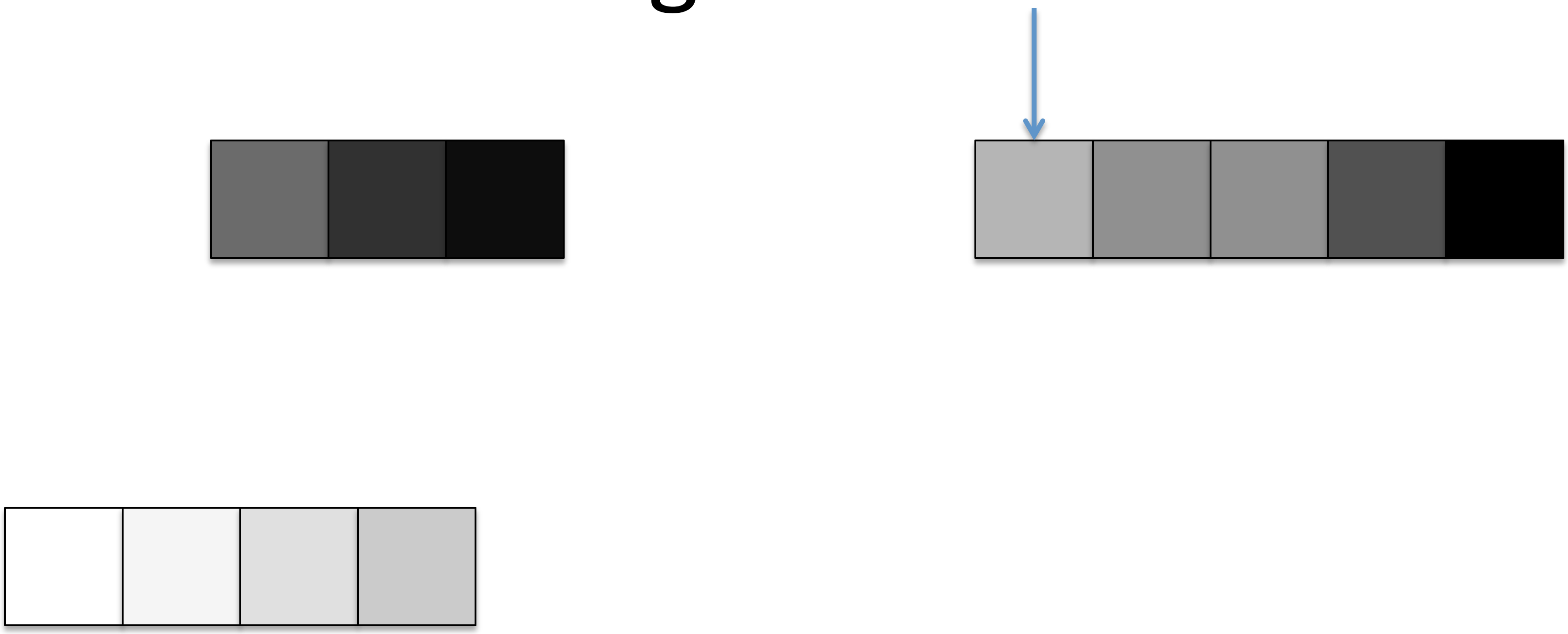
Merge Sort

Merge Sort



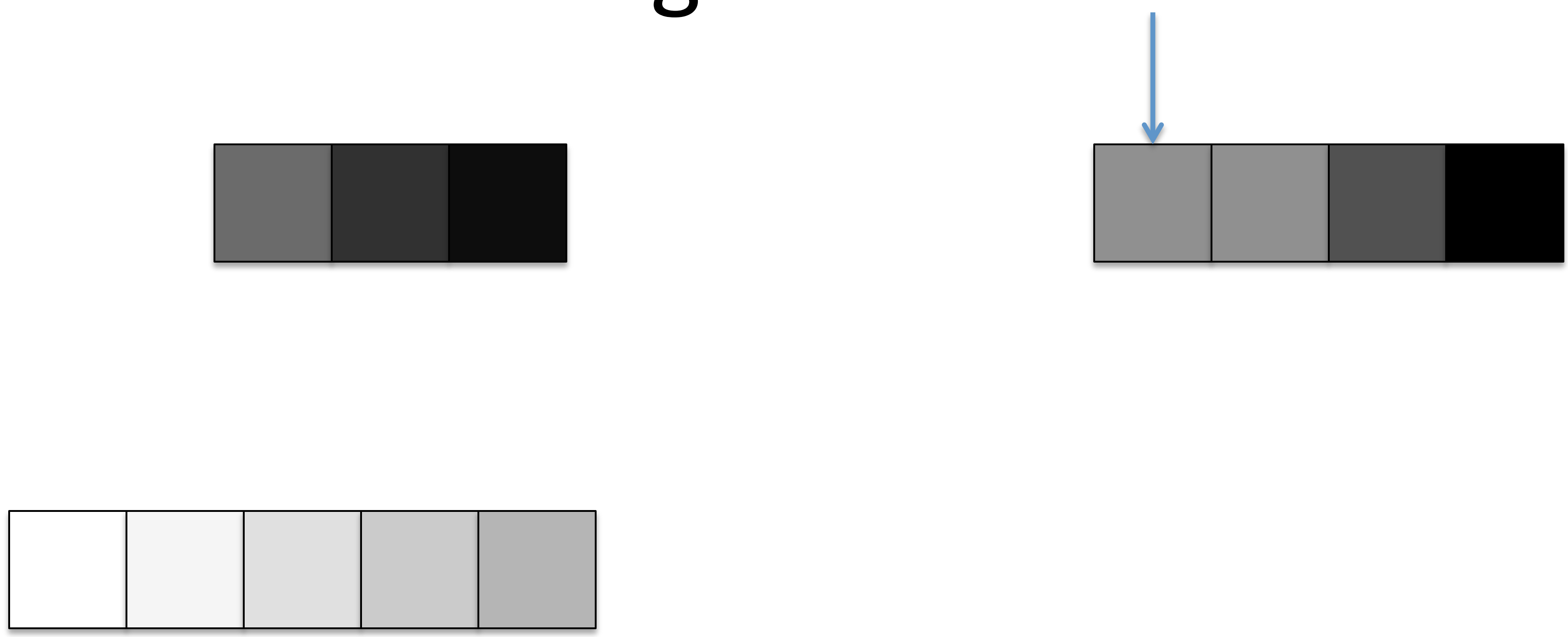
Merge Sort

Merge Sort

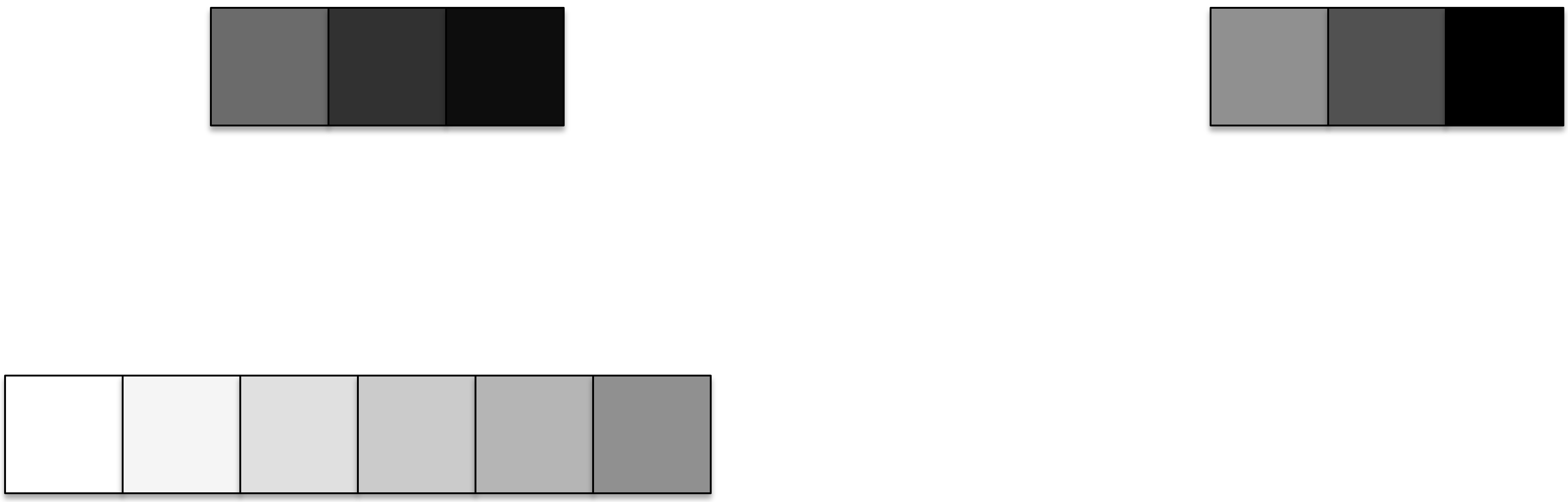


Merge Sort

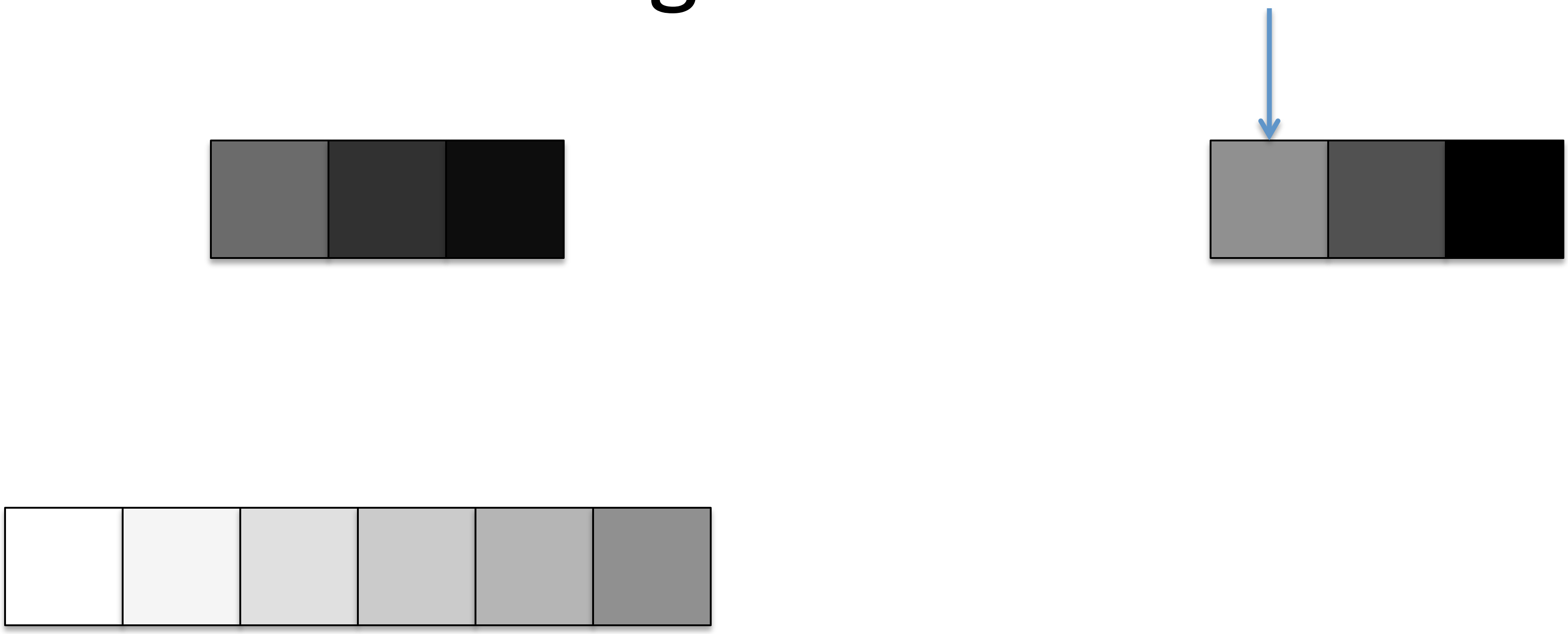
Merge Sort



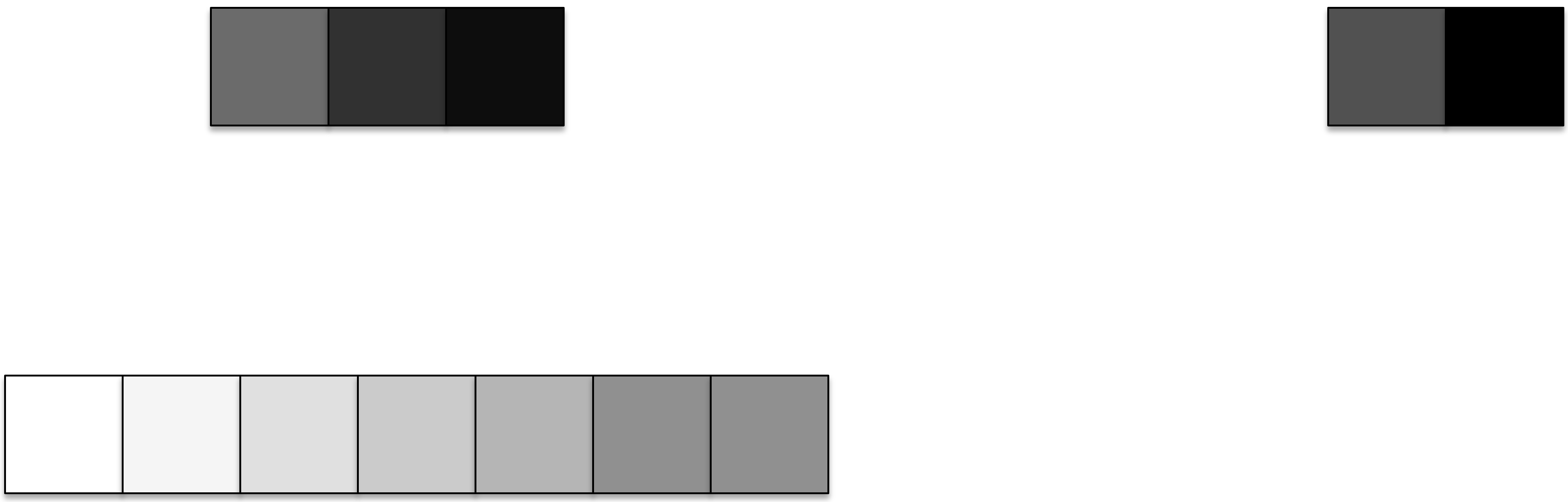
Merge Sort



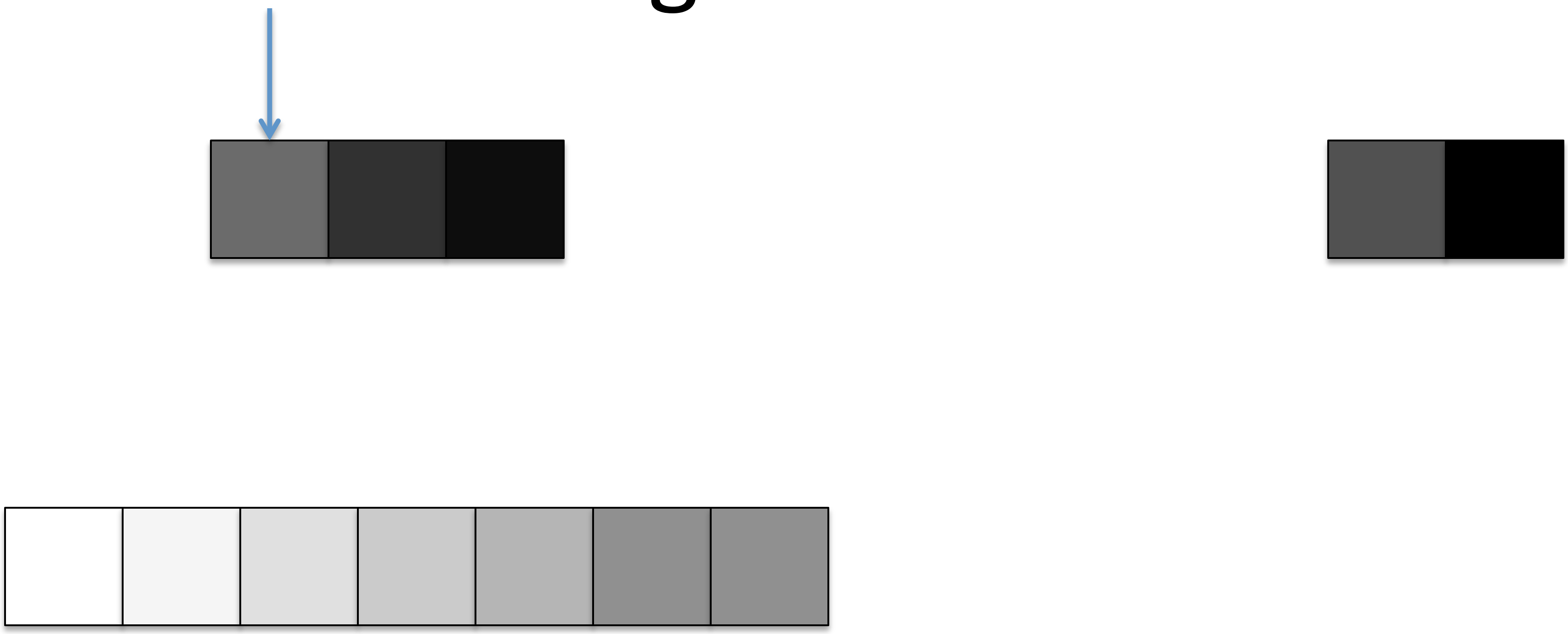
Merge Sort



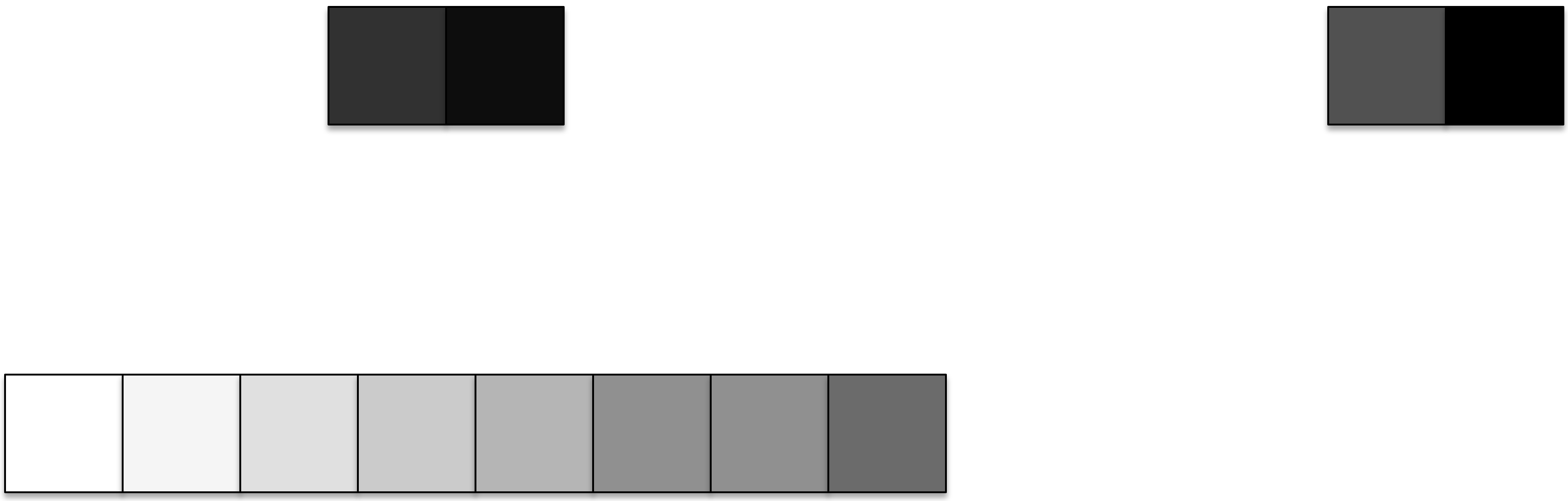
Merge Sort



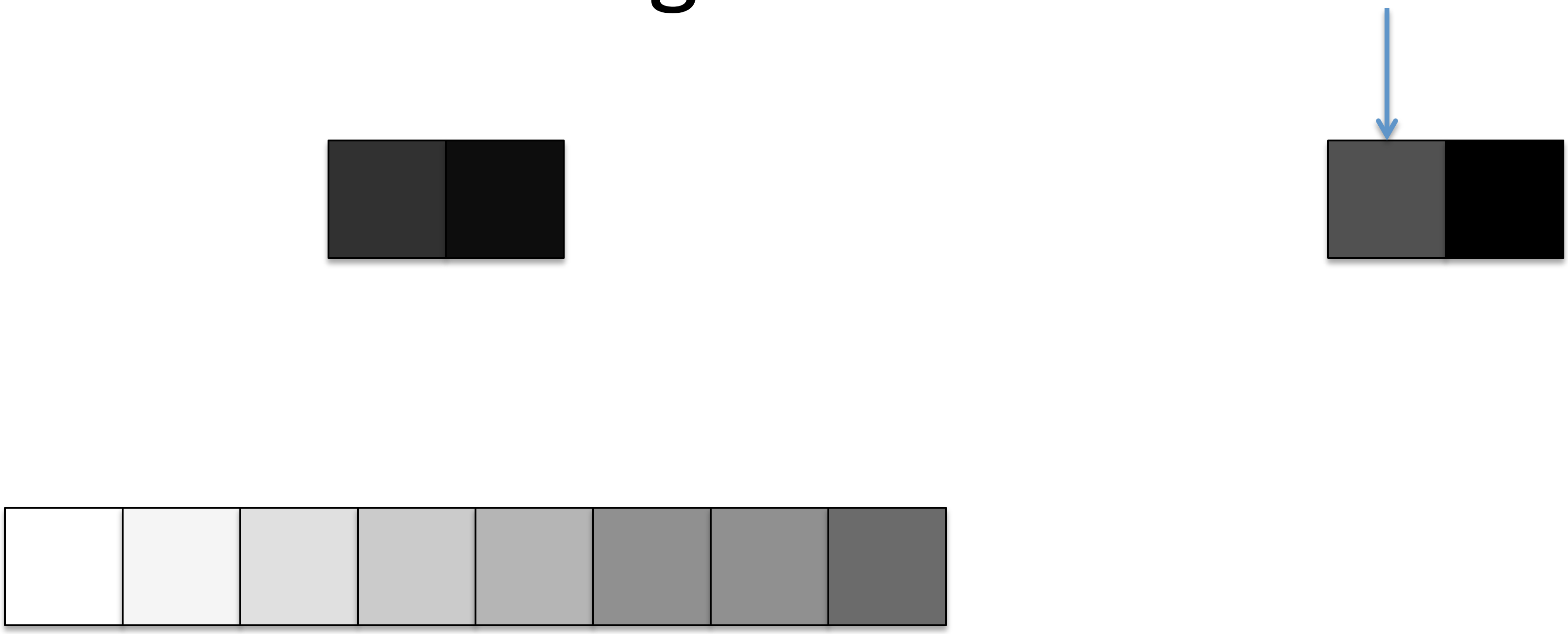
Merge Sort



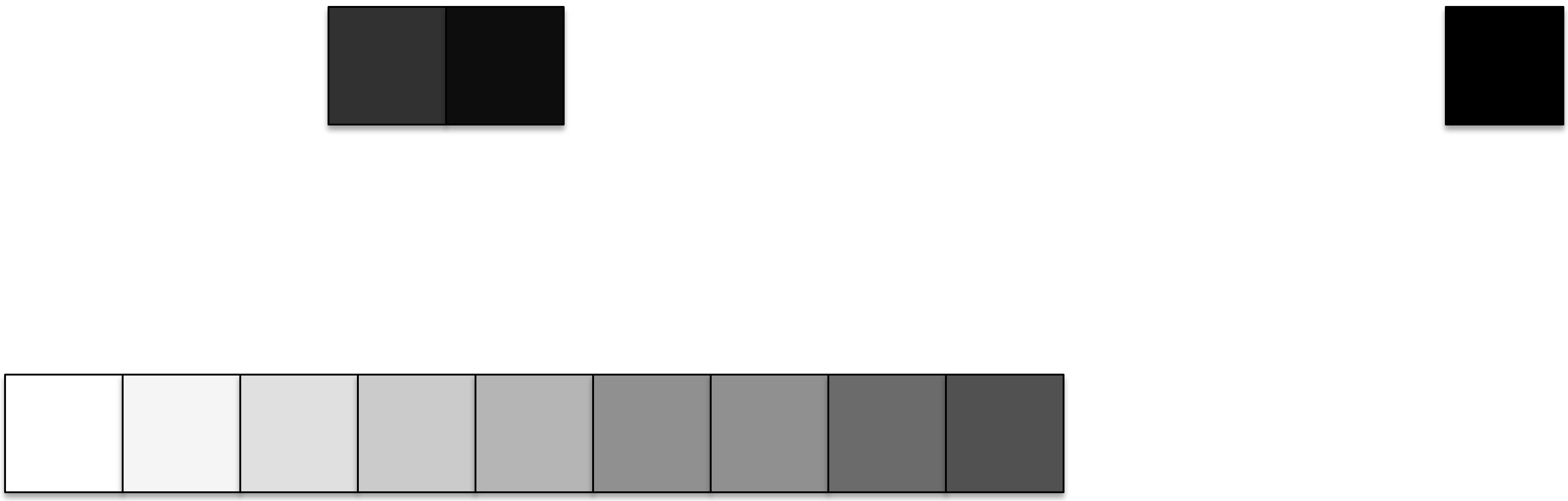
Merge Sort



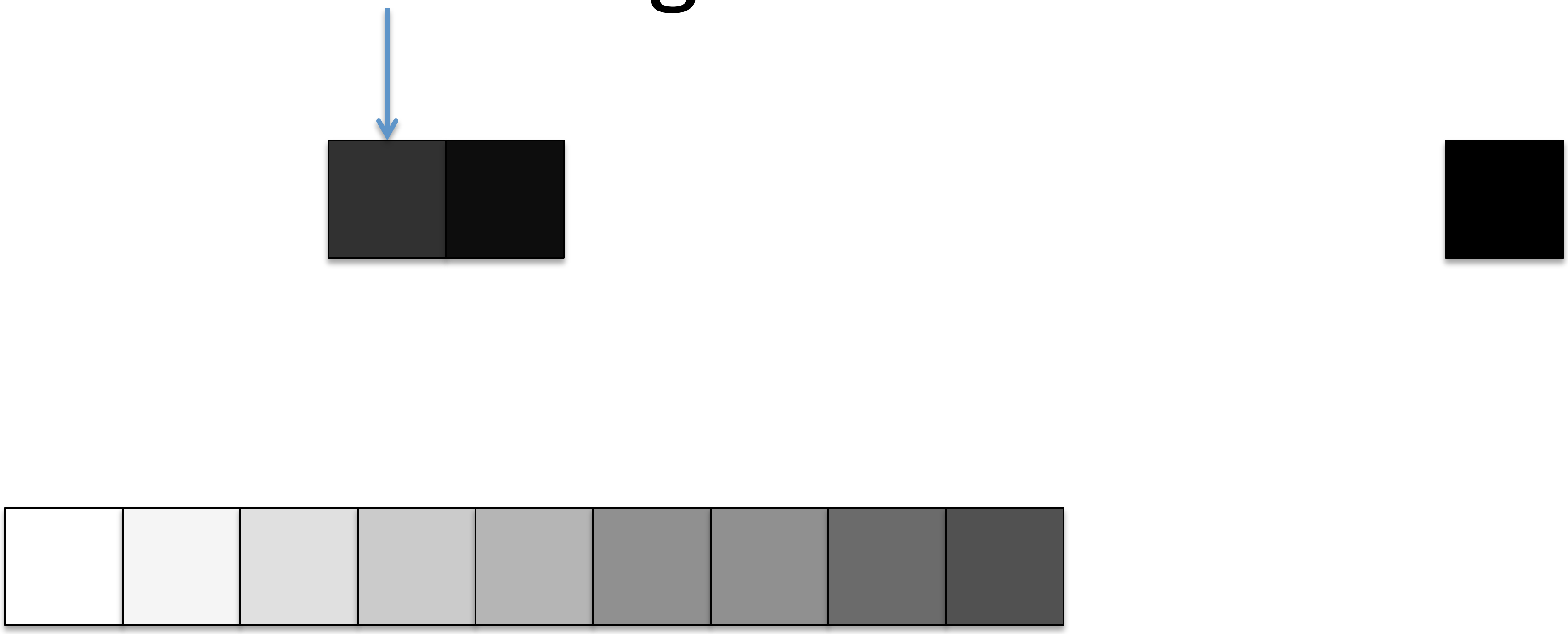
Merge Sort



Merge Sort

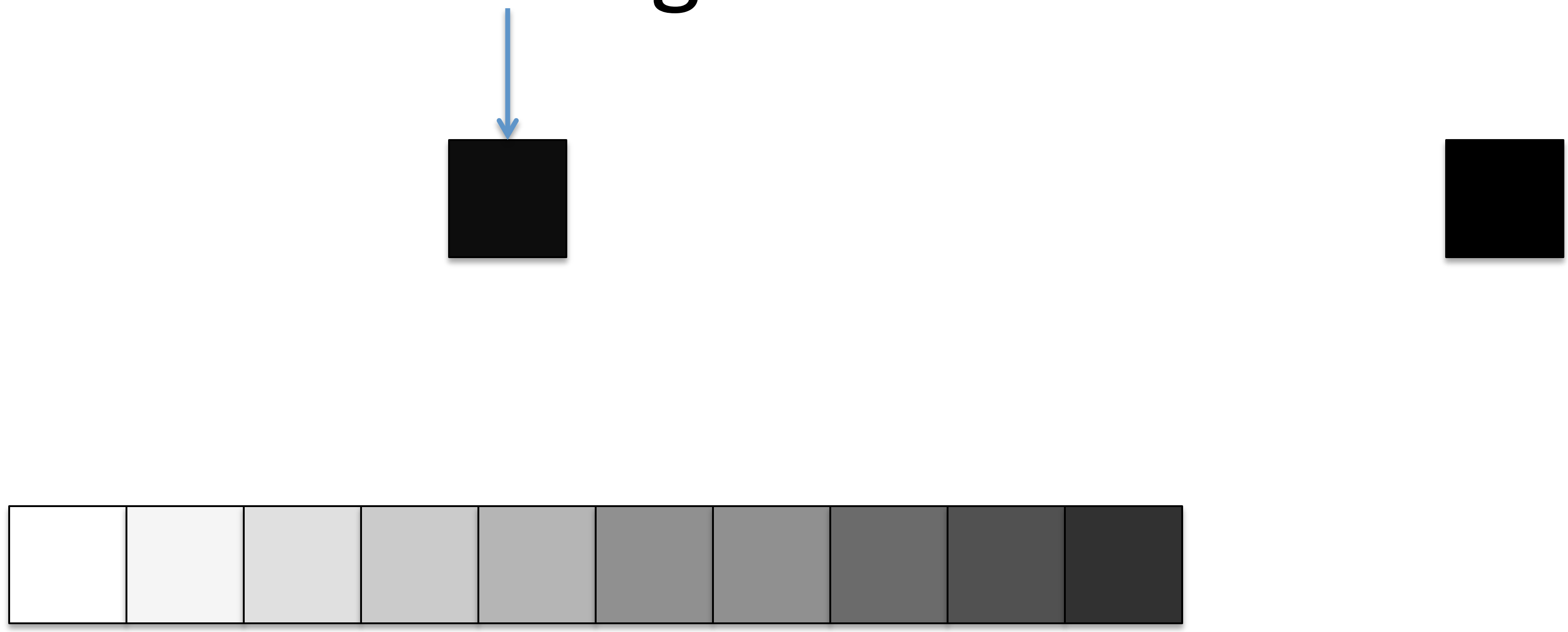


Merge Sort

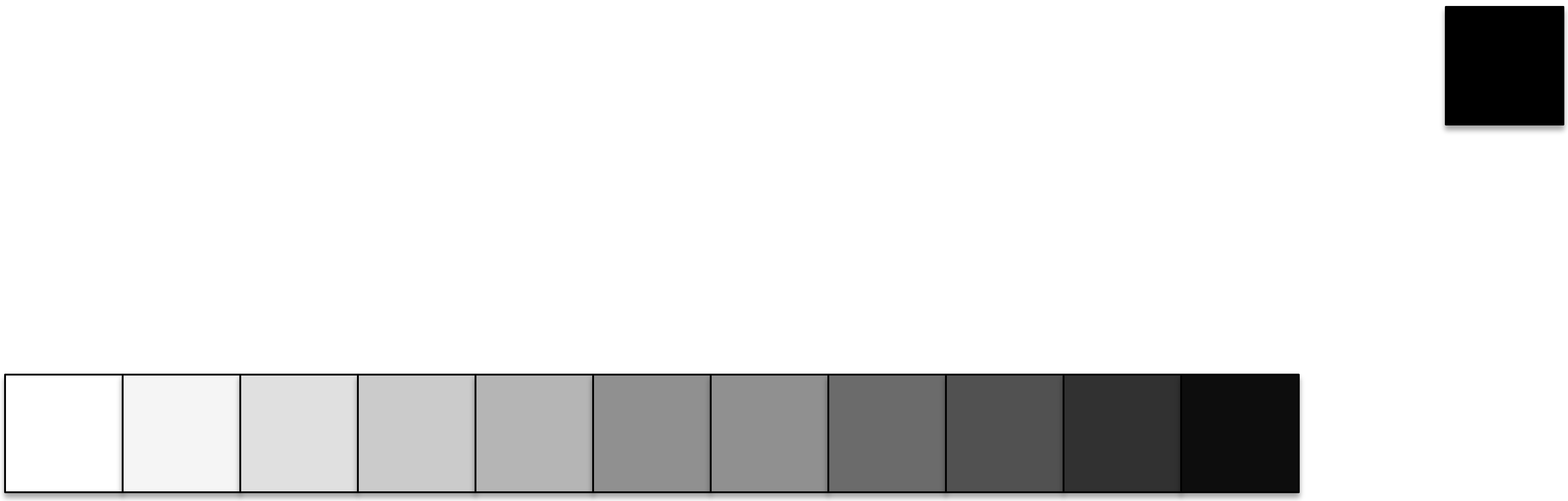


Merge Sort

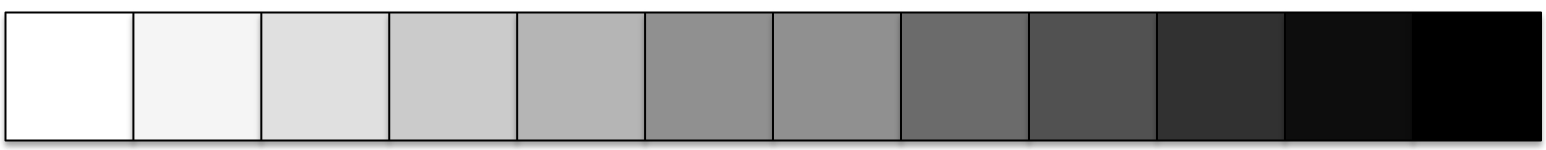
Merge Sort



Merge Sort



Merge Sort



Merge Sort

- Splitting the list: Easy! O(n)
- Two recursive calls: Easy!
- Merging two sorted lists?
  - Pick smaller of the first elements of the two lists, remove it, and add it to the final list.
  - Every iteration, length of final list grows
    - Can only iterate O(n) times
  - O(n) for merge

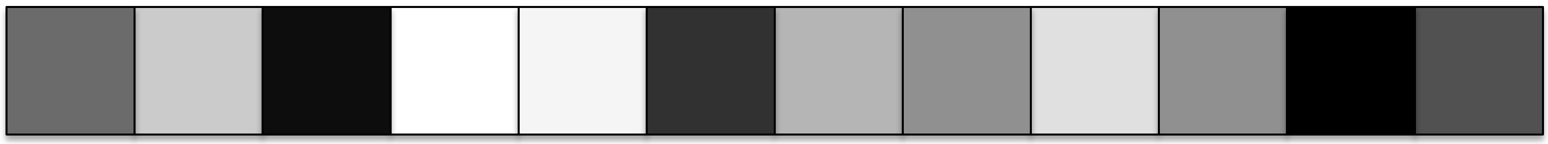
Merge Sort

- Running time: $T(n) = 2\ T(n/2) + O(n)$
- Master Method:
  - $a = 2$, $b = 2$, $d = 1$
  - $a = b^d$, so $O(n^d \log n) = O(n \log n)$

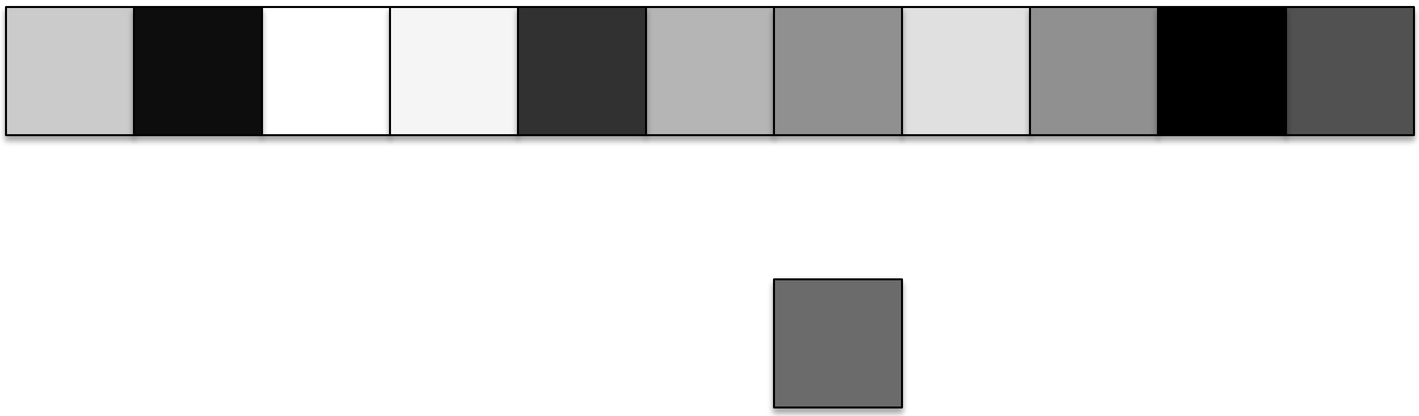
QuickSort

- What if instead of merging at end, we make sure all the elements in one list are less than all the elements in the other.
- Then we just concatenate the two lists, and are done
- To accomplish, take an element from the list, called the **pivot**, and make left list all elements less than it, right list all elements greater than it

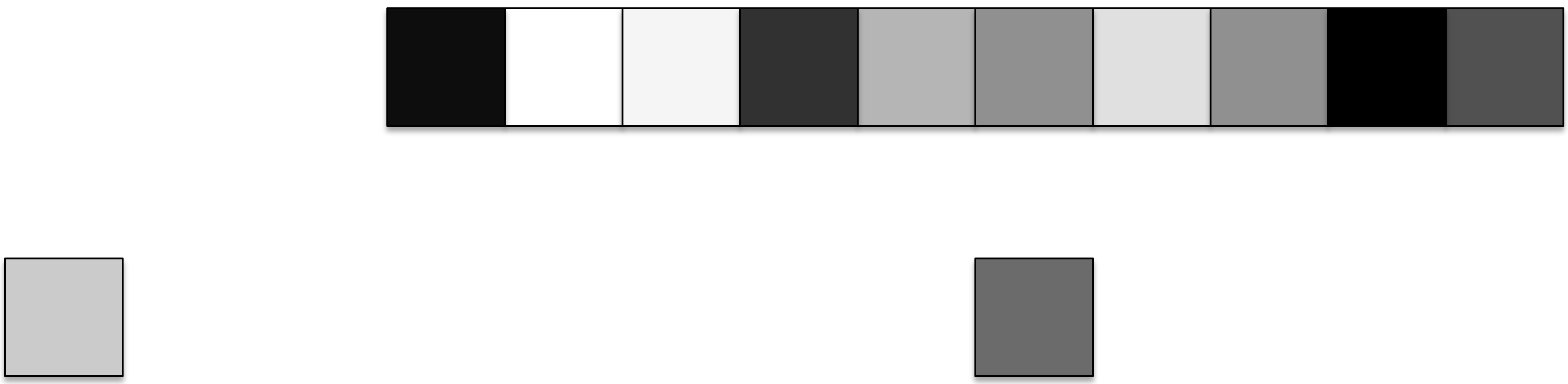
QuickSort



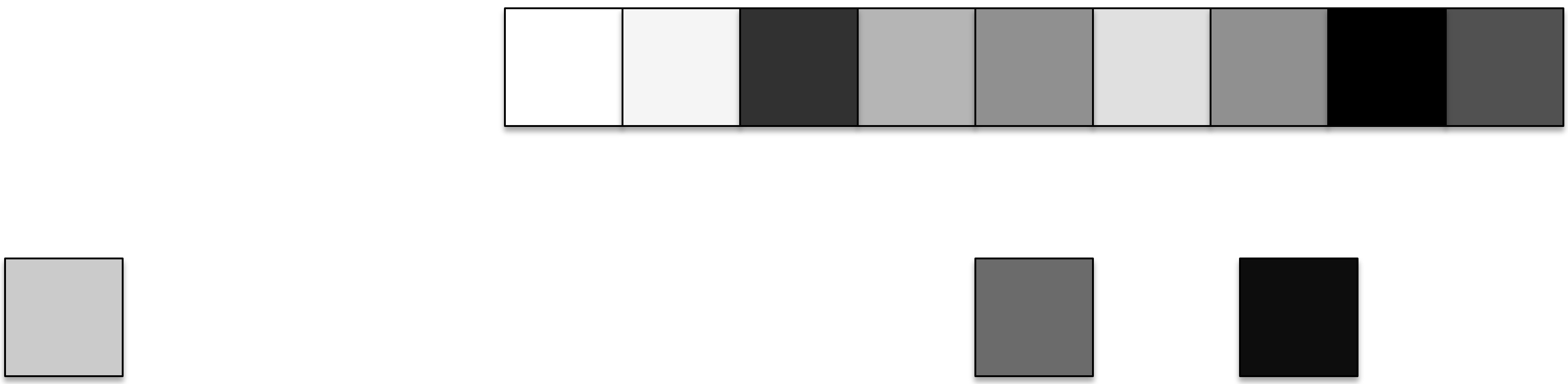
QuickSort



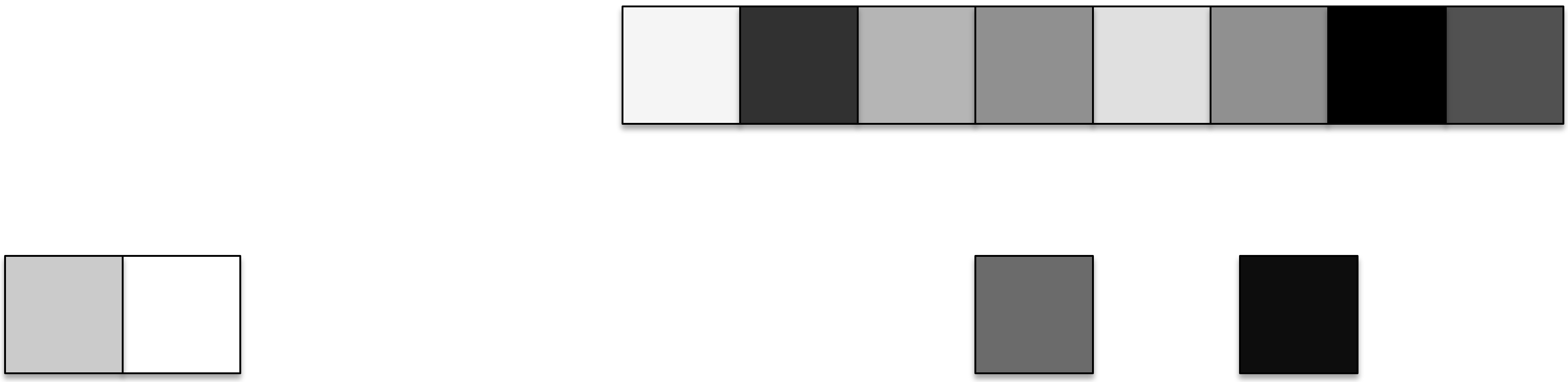
QuickSort



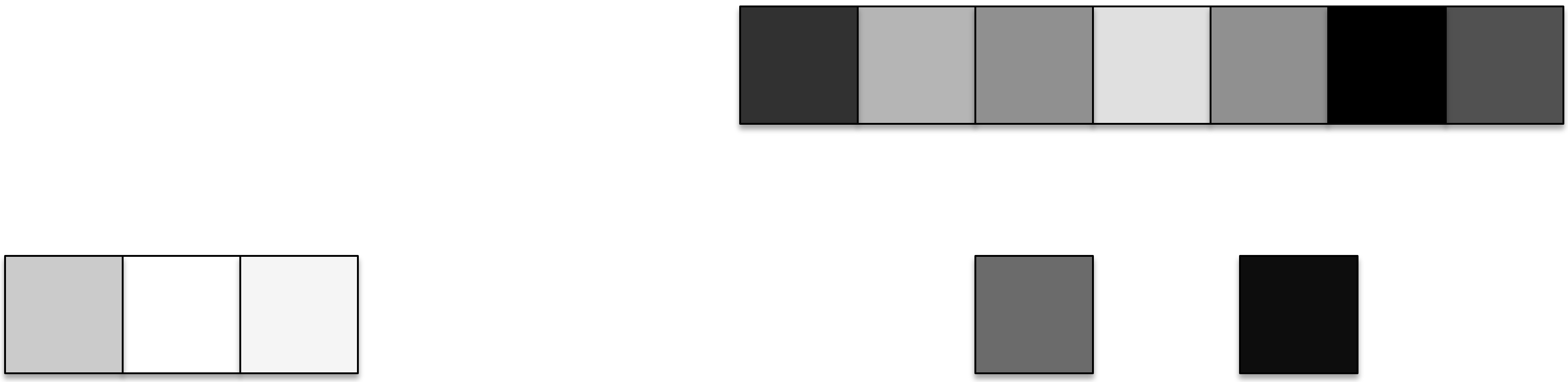
QuickSort



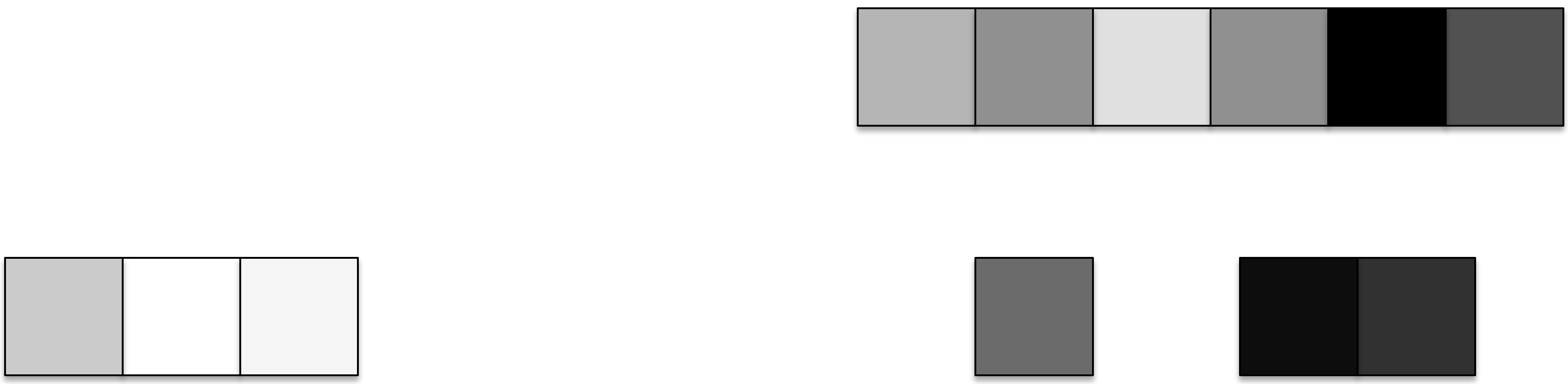
QuickSort



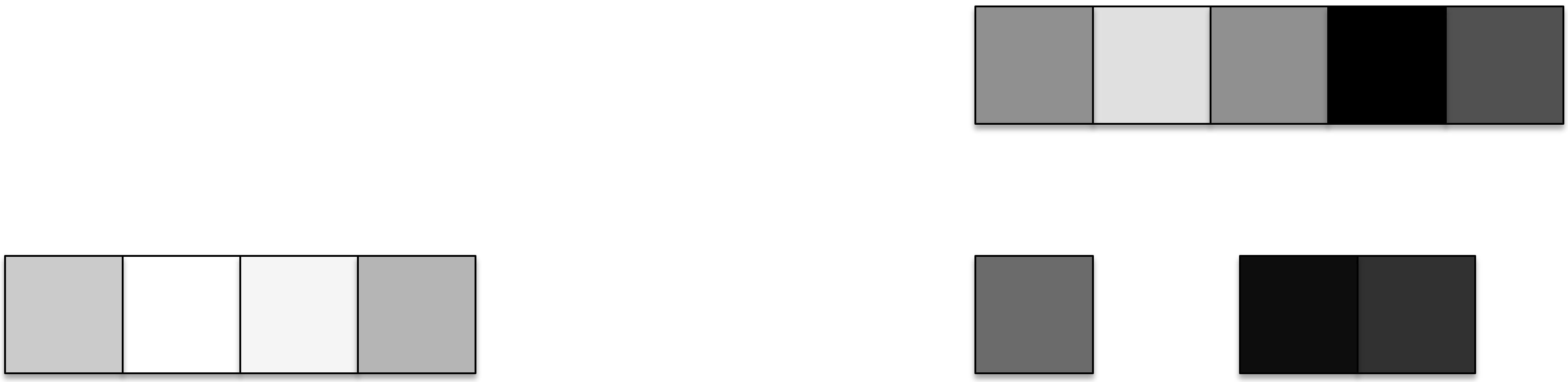
QuickSort



QuickSort

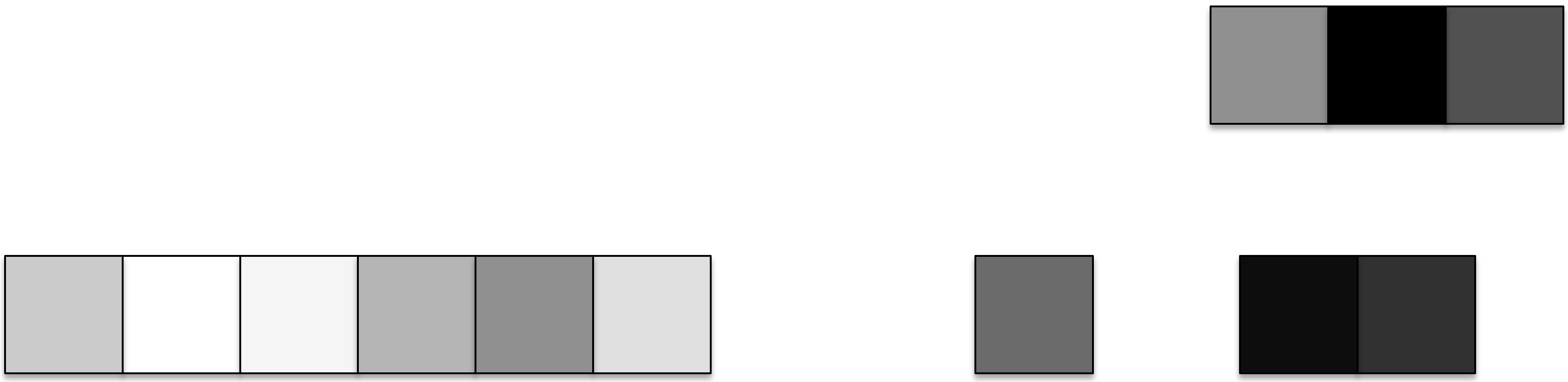


QuickSort

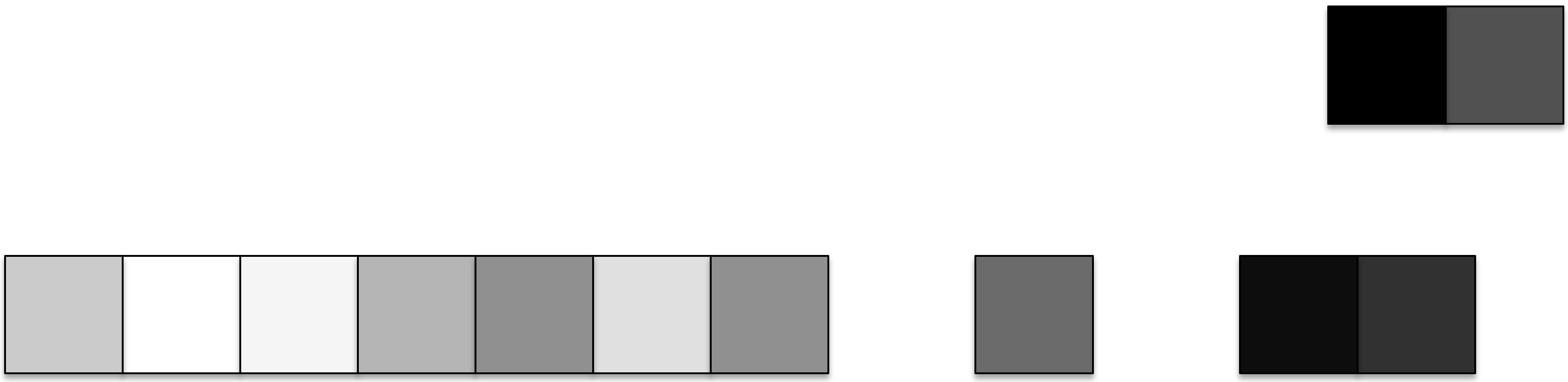


QuickSort

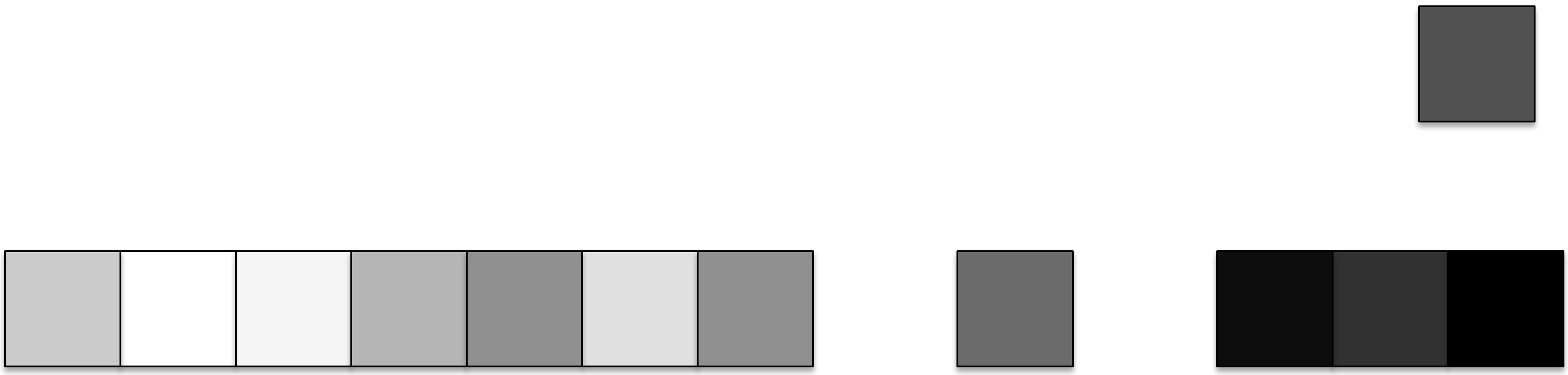
QuickSort



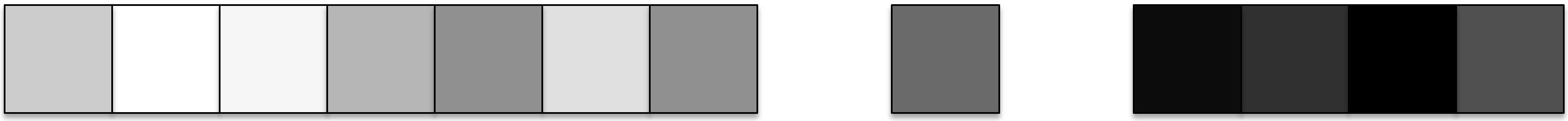
QuickSort



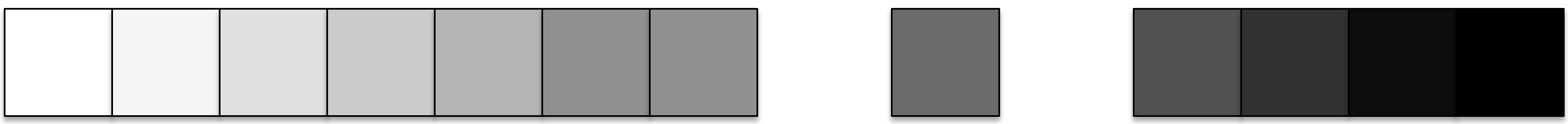
QuickSort



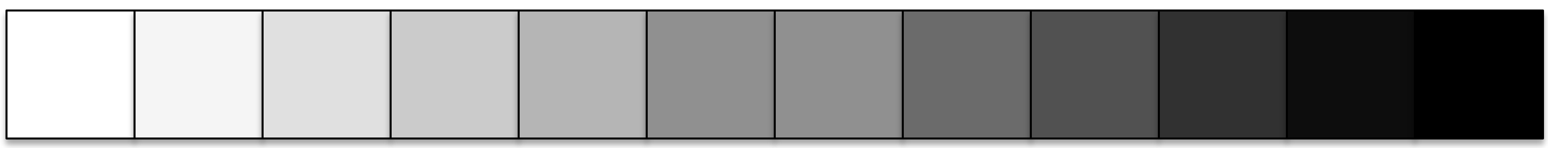
QuickSort



QuickSort



QuickSort



QuickSort Running Time

- O(n) work before collision to split lists
- Let p be the pivot, k the number of elements less than p
- One recursive call of size k, one of size n-1-k
- $T(n) = T(k) + T(n-1-k) + O(n)$

QuickSort Running Time

- Best case: $k = n/2$
- $T(n) = 2T(n/2)+O(n) \rightarrow T(n) = O(n \log n)$
- Worst case: $k = 0$ (i.e. elements are already in order)
- $T(n) = T(n-1)+T(0)+O(n)=T(n-1)+O(n)$
  - $T(n) = O(n^2)$

QuickSort Average Case

- What if input is in random order?
  - k is a random value between 0 and n-1
  - Expected running time?

$$T(n) \le O(n) + \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-k-1)\right)$$

$$\le O(n) + \frac{2}{n}\sum_{k=0}^{n-1}T(k)$$

QuickSort Average Case

$$T(n) \le cn + \frac{2}{n}\sum_{k=0}^{n-1} T(k)$$

- Claim: there is a constant d such that T(n) ≤ dn log n

QuickSort Average Case

- Proof: Assume T(k) ≤ dk log k for k < n

$$T(n) \le cn + \frac{2}{n}\sum_{k=0}^{n-1}\left(dk\log k\right)$$

$$= cn + \frac{2d}{n}\left(\sum_{k=0}^{\lceil (n-1)/2 \rceil} k\log k + \sum_{k=\lceil (n-1)/2 \rceil + 1}^{n-1} k\log k\right)$$

$$\le cn + \frac{2d}{n}\left(\log\frac{n}{2}\left(\sum_{k=0}^{\lceil (n-1)/2 \rceil} k\right) + \log n\left(\sum_{k=\lceil (n-1)/2 \rceil + 1}^{n-1} k\right)\right)$$

QuickSort Average Case

$$T(n) \le cn + \frac{2d}{n}\left(\log\frac{n}{2}\left(\sum_{k=0}^{\lceil (n-1)/2 \rceil} k\right) + \log n\left(\sum_{k=\lceil (n-1)/2 \rceil+1}^{n-1} k\right)\right)$$

$$= cn + \frac{2d}{n}\left(\log n\left(\sum_{k=0}^{n-1} k\right) - \left(\sum_{k=0}^{\lceil (n-1)/2 \rceil} k\right)\right)$$

$$= cn + \frac{2d}{n}\left(\frac{n(n-1)}{2}\log n - \frac{\lceil (n-1)/2 \rceil(\lceil (n-1)/2 \rceil - 1)}{2}\right)$$

QuickSort Average Case

$$T(n) \le cn + \frac{2d}{n}\left(\frac{n(n-1)}{2}\log n - \frac{(n-1)/2((n-1)/2-1)}{2}\right)$$

$$= cn + d(n-1)\log n - \frac{d}{8}\left(n - \frac{1}{n}\right)$$

$$\le dn\log n - \left(\frac{d}{8} - c\right)n + d\left(\frac{1}{8} - \log n\right)$$

Thus, we can set d=8c, and the desired inequality holds for log n ≥ 1/8, which holds for n ≥ 2

QuickSort Randomized

- What if we really want worst-case bounds?
- Instead of picking pivot to be the first element, pick pivot at random
- k, the number of elements below the pivot, is still a random integer form 0 to n-1
- Expected running time:

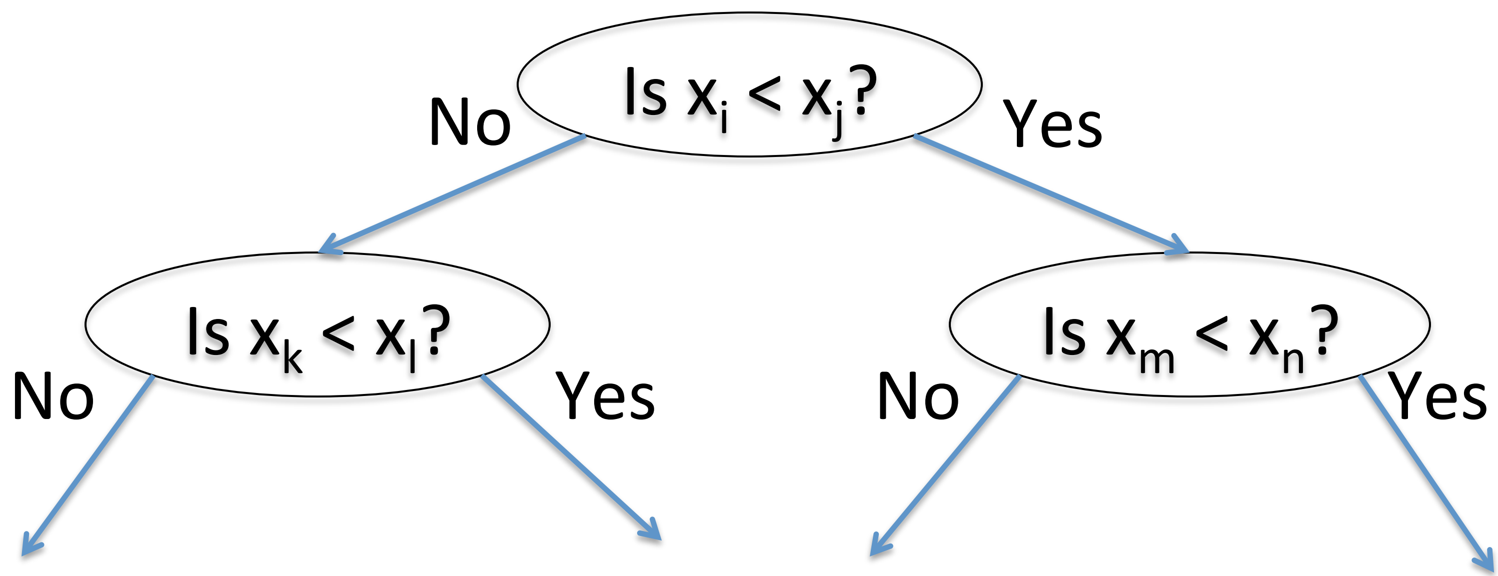
$$T(n) \leq O(n) + \frac{1}{n} \sum_{k=0}^{n-1} \left(T(k) + T(n-k-1) \right)$$

Comparison-Based Sorting

- Heap sort, Merge sort, and QuickSort all have running time $O(n \log n)$. Why?
- Theorem: Any sorting algorithm that only makes questions of the form "is $x < y$?" must make $\Omega(n \log n)$ comparisons.

Decision-Tree Model

- Any algorithm that only asks questions about the input of the form "is x < y?" can be represented as a tree



Decision-Tree Model

- Label leaves with permutations p of [1,…,n]
  - [p(1), p(2), …, p(n)]
  - Corresponds to ordering where $x_{p(1)} < x_{p(2)} < \ldots < x_{p(n)}$
- Permutation must be consistent with answers to questions
  - Let r_i and r_j be the integers such that $i = p(r_i)$ and $j = p(r_j)$
  - If $x_i < x_j$ was answered yes, then $r_i < r_j$

Decision-Tree Model

- All possible permutations must be present
  - What if permutation is missing, and we give algorithm an input with the corresponding ordering?
  - The algorithm will think we are in a different ordering, and produce the wrong output

Decision-Tree Model

- Number of permutations?
  - First pick p(1): n choices
  - Then pick p(2): n-1 choices
  - …
  - Pick p(n): 1 choice
  - Total number of choices: n!

Decision-Tree Model

- Number of permutations: n!
- Number of leaves: ≥ n!
- Depth of tree: ≥ log n!
- Number of comparisons in algorithm: ≥ log n!
- Need to asymptotically bound log n!

Bounding log n!

- log n! = O(n log n)
  - log n! = log n + log (n-1) + … + log 1

    < n log n
- Log n! = Ω(n log n)
  - log n! = log n + log (n-1) + … + log 1

    > log n + log (n-1) + … + log n/2

    > (n/2) log (n/2) = Ω(n log n)

Comparison-Based Sorting

- Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons
- One of the very few non-trivial lower bounds that we know of
- What about linear sorting algorithms?
  - Not comparison-based

n!

- We can actually do better for bounding n! using Stirling's Approximation:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Finding the Median

- Finding the smallest value of a list is easy
  - Go through the list, keeping track of the smallest element. O(n)
- Finding the kth smallest value of a list, for constant k, is easy
  - Go through the list, keeping track of the k smallest elements. O(n) if k is constant
- What about k = n/2?
  - Sort, pick out middle element. O(n log n)
  - Is there any way to get O(n)?

Select

- Find the kth smallest element in a list
- Divide and conquer approach?
  - Pick a pivot p
  - Create two lists, l1 with all elements less than p, and l2 with all elements greater than p
  - If k is at most |l1|, then recursively call on l1
  - If k = |l1|+1, return p
  - Otherwise, call on l2 with k' = k - |l1|-1

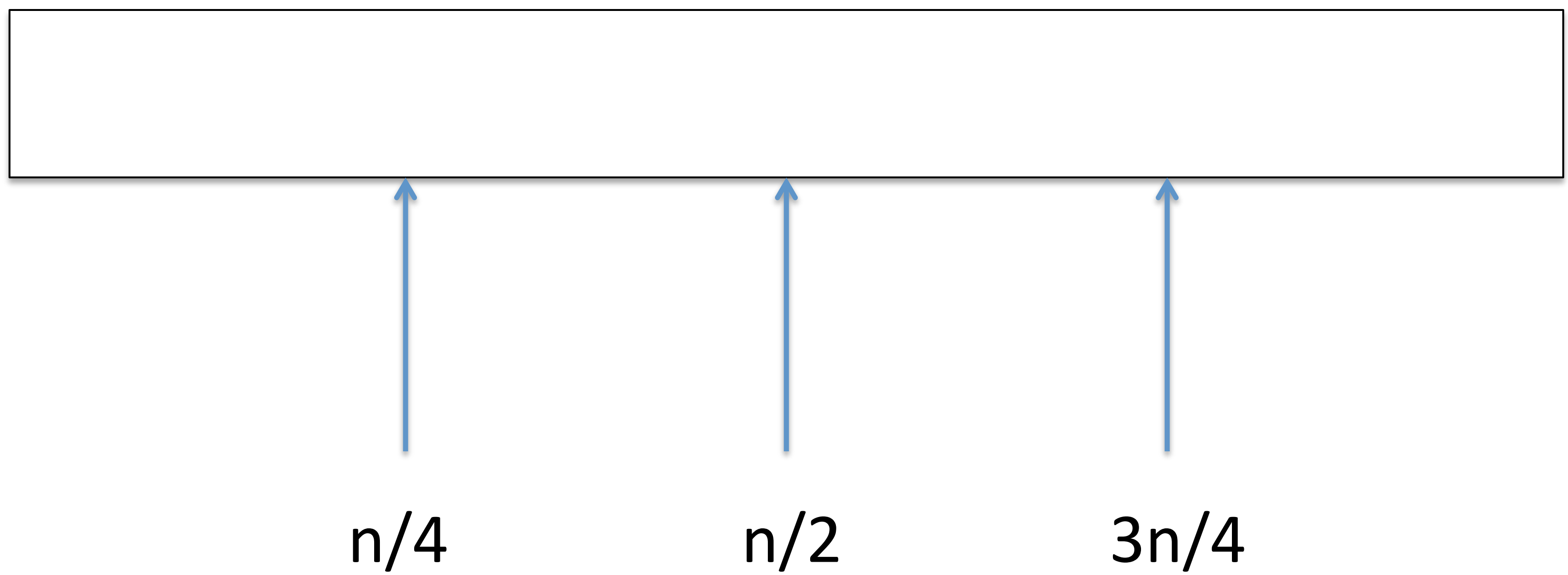
Select

- Running Time?
  - Best case: p happens to be the median
    - $T(n) = T(n/2) + O(n) \rightarrow T(n) = O(n)$
  - Worst case: p is the smallest or largest element
    - $T(n) = T(n-1) + O(n) \rightarrow T(n) = O(n^2)$

Select Expected Running Time

- If we choose pivot randomly, the number of elements smaller than it will be a random integer from 0 to n-1
- Can write recurrence for expected run time:
  - $T(n) = T(3n/4) + g\ O(n)$
  - g = expected number of of recursive calls until the list has size 3n/4

Select Expected Running Time



- How many splits until pivot between n/4 and 3n/4? g = 2

Select Expected Running Time

- $T(n) = T(3n/4) + O(n)$
  - $a = 1, b = 4/3, d = 1$
  - $a < b^d$, so $T(n) = O(n^d) = O(n)$

Worst Case Linear Time?

- How can we get a linear time worst case select?
- Idea: want to pick pivot close to the median
  - Can use select to pick good pivot

Median-of-Medians

- Group elements off arbitrarily into n/5 groups of 5
- Find median of each group
- Find and output median of medians

Median-of-Medians

- Finding median of 5 elements: O(1) since a fixed number of comparisons
- Finding medians of all n/5 groups: O(n)
- Finding median of n/5 medians: T(n/5)

Median-of-Medians

- How good is the median-of-medians?
  - The median of each group is larger than 2 elements
  - The median-of-medians is larger than (n/5)/2 = n/10 group medians, as well as the elements these medians are larger than
  - Median-of-medians is larger than 3n/10
  - Also smaller than 7n/10
  - Therefore, next recursive call has size at most 7n/10

Worst-case Linear Select

- Select(l,k) =
  - Arbitrarily group elements into groups of five
  - Construct l1, the list of medians of each group
  - Let p = Select(l1,|l1|/2)
  - Construct l2 and l3, the lists of elements smaller and greater than p
  - If |l2| ≤ k, call Select(l2,k)
  - If |l2| = k+1, return p
  - Otherwise, call Select(l3,k-|l2|-1)

Worst-case Linear Select

- Running Time:
  - O(n) for grouping and constructing list of medians
  - T(n/5) for computing pivot
  - O(n) for constructing l2 and l3
  - At most T(7n/10) for recursive call to Select
  - $T(n) = T(n/5) + T(7n/10) + O(n)$

Akra-Bazzi Method

$$T(n) = \sum_i a_i T(n / b_i) + O(n^d)$$

- Let f be the solution to $\sum_i \frac{a_i}{b_i^f} = 1$

- Then:
  - If $f < d$, $T(n) = O(n^d)$
  - If $f > d$, $T(n) = O(n^f)$
  - If $f = d$, $T(n) = O(n^d \log n)$

Worst-case Linear Select

- $T(n) = T(n/5) + T(7n/10) + O(n)$
  - $a_1 = a_2 = 1$
  - $d = 1$
  - $b_1 = 5$, $b_2 = 10/7$
  - Solve $$1 = \sum_i \frac{a_i}{b_i^f} = \left(\frac{1}{5}\right)^f + \left(\frac{7}{10}\right)^f$$
  - $f \approx 0.84$, do $d > f$
  - Therefore, $T(n) = O(n^d) = O(n)$