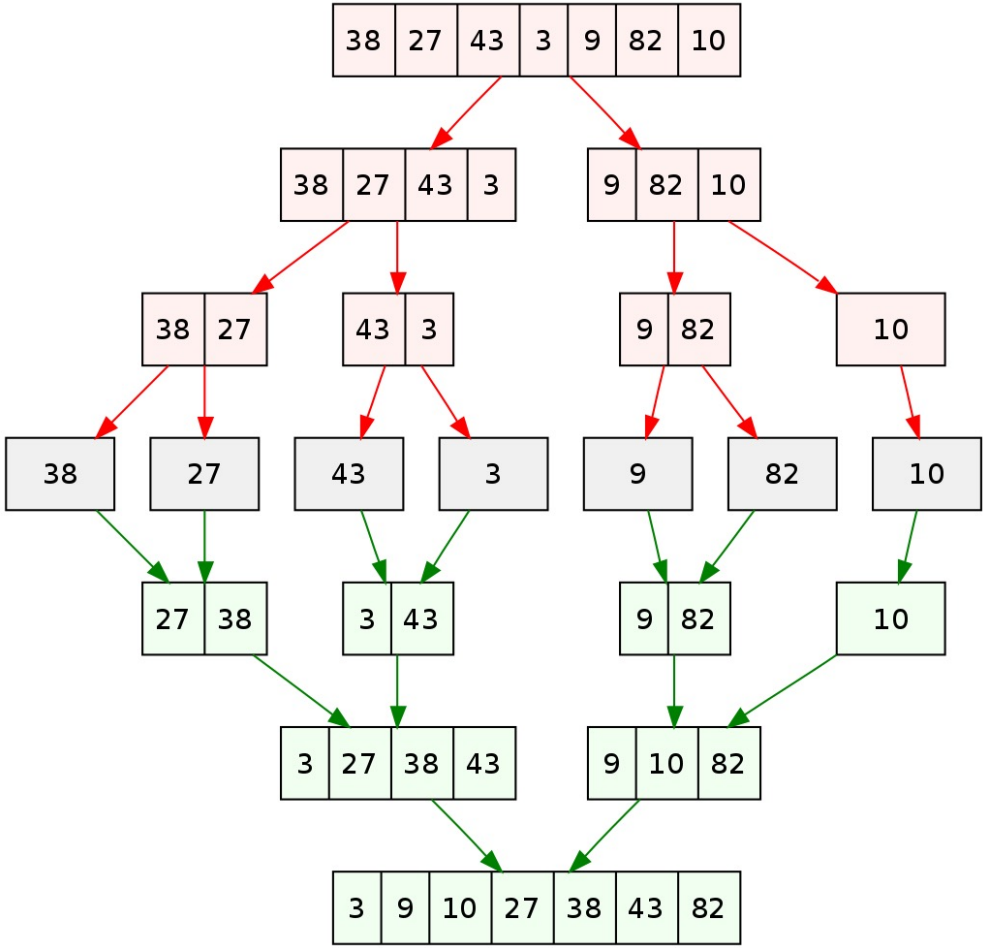
Merge sort

CLRS 2.3



Recap

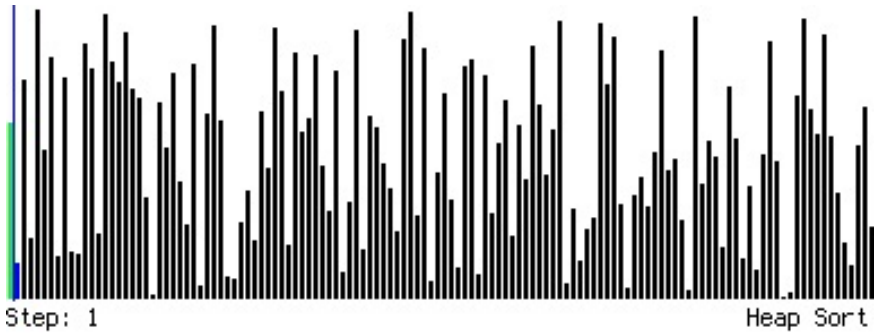
- What sorting algorithms have we covered so far?
- How efficient are they?

Divide-and-conquer approach

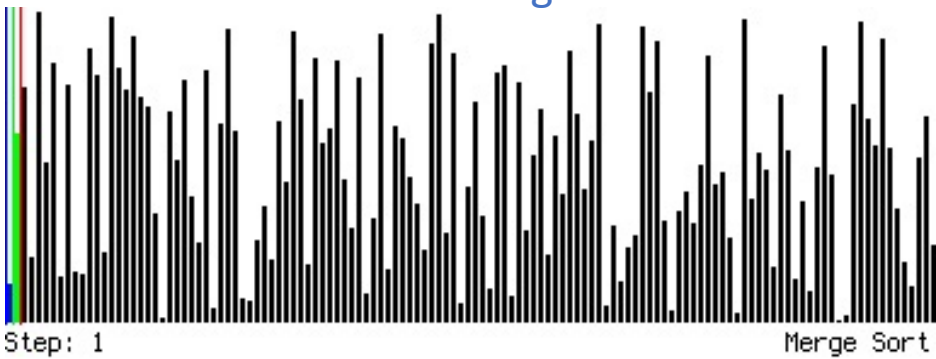
- **Divide-and-conquer** is a general algorithm design paradigm:
  - Divide: divide the input data S in two disjoint subsets S1 and S2
  - Conquer: solve the subproblems associated with S1 and S2
    - the base case for the recursion are subproblems of size 0 or 1
  - Combine: combine the solutions for S1 and S2 into a solution for S

- **Merge-sort** is a sorting algorithm based on the divide-and-conquer paradigm
  - Like heap-sort
    - Uses a comparator
    - Has $O(n \log n)$ running time
  - Unlike heap-sort
    - Accesses data in a sequential manner (suitable to sort data on a disk)

Heap sort



Merge sort



Merge sort

Merge sort works on an input sequence with *n* elements and consists of three steps:

- Divide: partition the *n*-element sequence to be sorted into two subsequences of *n/2* elements each
- Conquer: sort the two subsequences recursively using merge sort
- Combine: merge the two sorted subsequences to produce the sorted answer

The recursion "bottoms out" when the sequence to be sorted has length 1 or less.

- Q: How many recursive calls until this happens? $O(\log n)$ recursive calls

```
MERGE-SORT(A, p, r)
  if p < r                          // check for base case
      q = ⌊(p + r)/2⌋               // divide
      MERGE-SORT(A, p, q)           // conquer
      MERGE-SORT(A, q + 1, r)       // conquer
      MERGE(A, p, q, r)             // combine
```

Combine step: merge two sorted lists

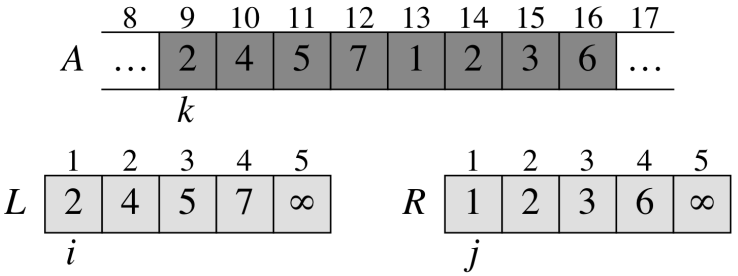
Cost of one call to merge two lists with total n items is $O(n)$

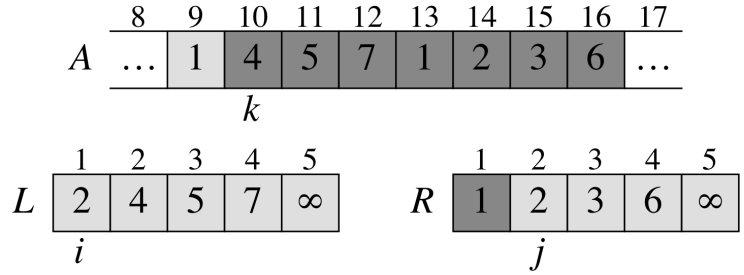
MERGE(A, p, q, r)

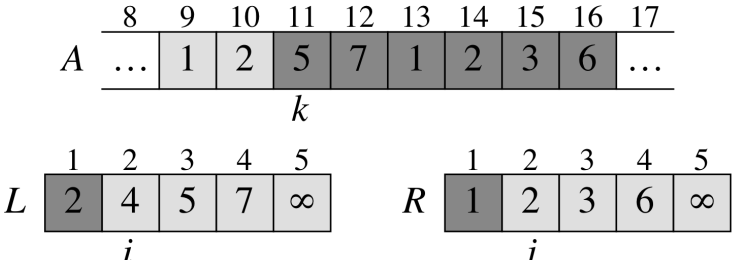
$n_1 = q - p + 1$
$n_2 = r - q$
let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays
for $i = 1$ **to** n_1
    $L[i] = A[p + i - 1]$
for $j = 1$ **to** n_2
    $R[j] = A[q + j]$
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$
for $k = p$ **to** r
    if $L[i] \le R[j]$
        $A[k] = L[i]$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

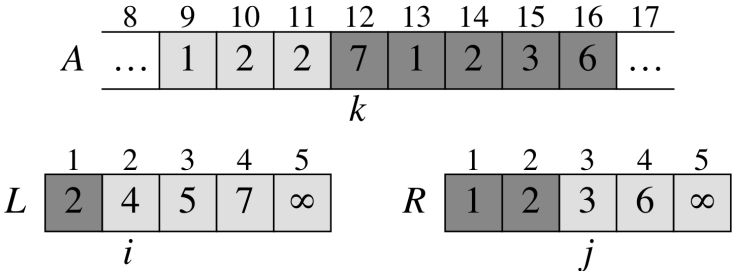
Keep track of two arrays (**L**eft and **R**ight).

Keep grabbing the smallest item from the front of either array, working up one by one.

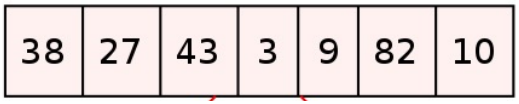




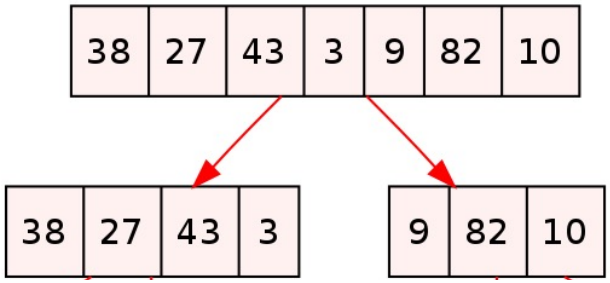




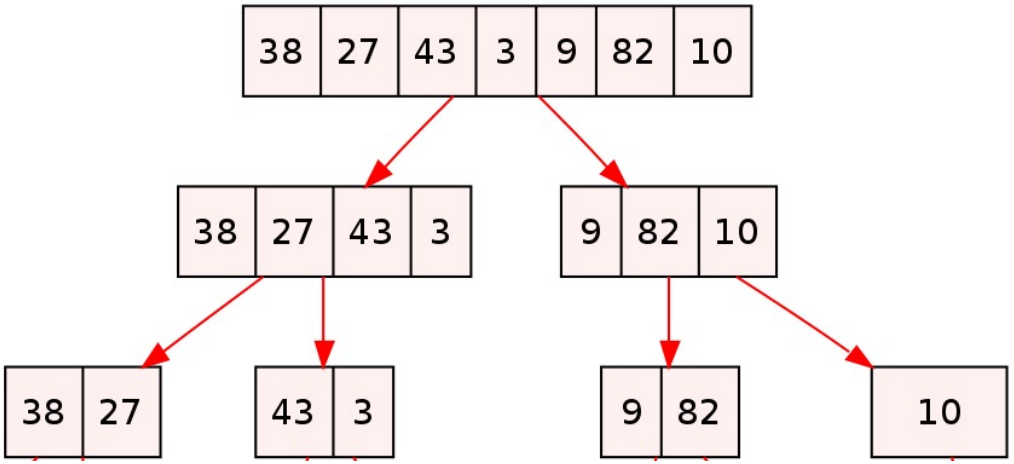
Example



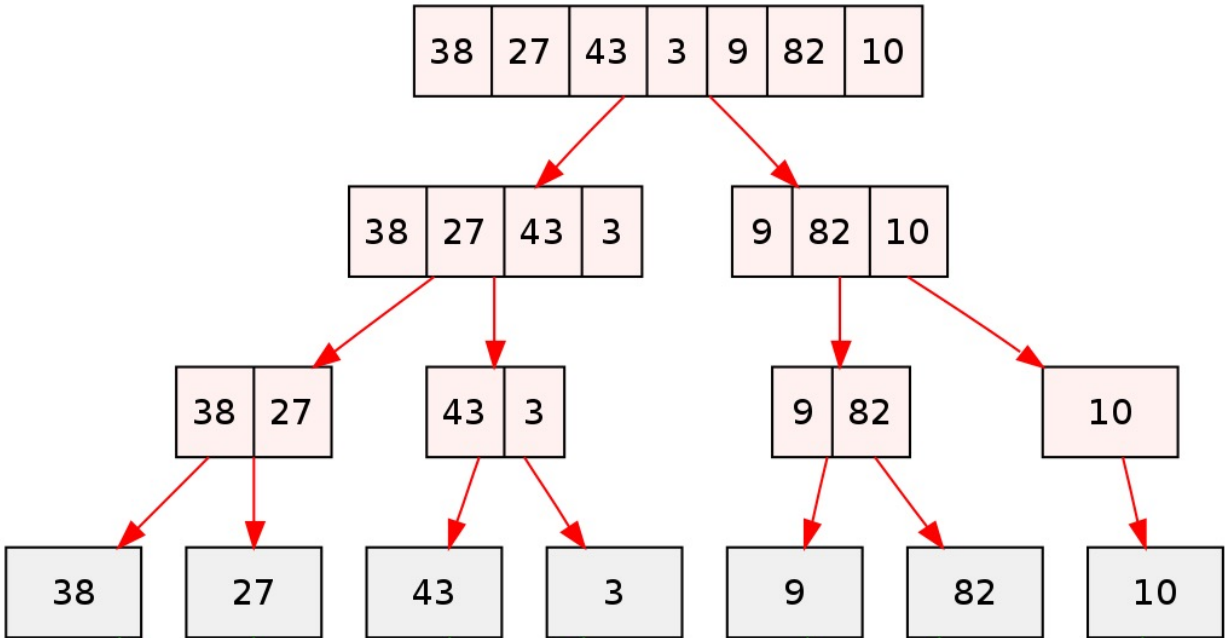
Example



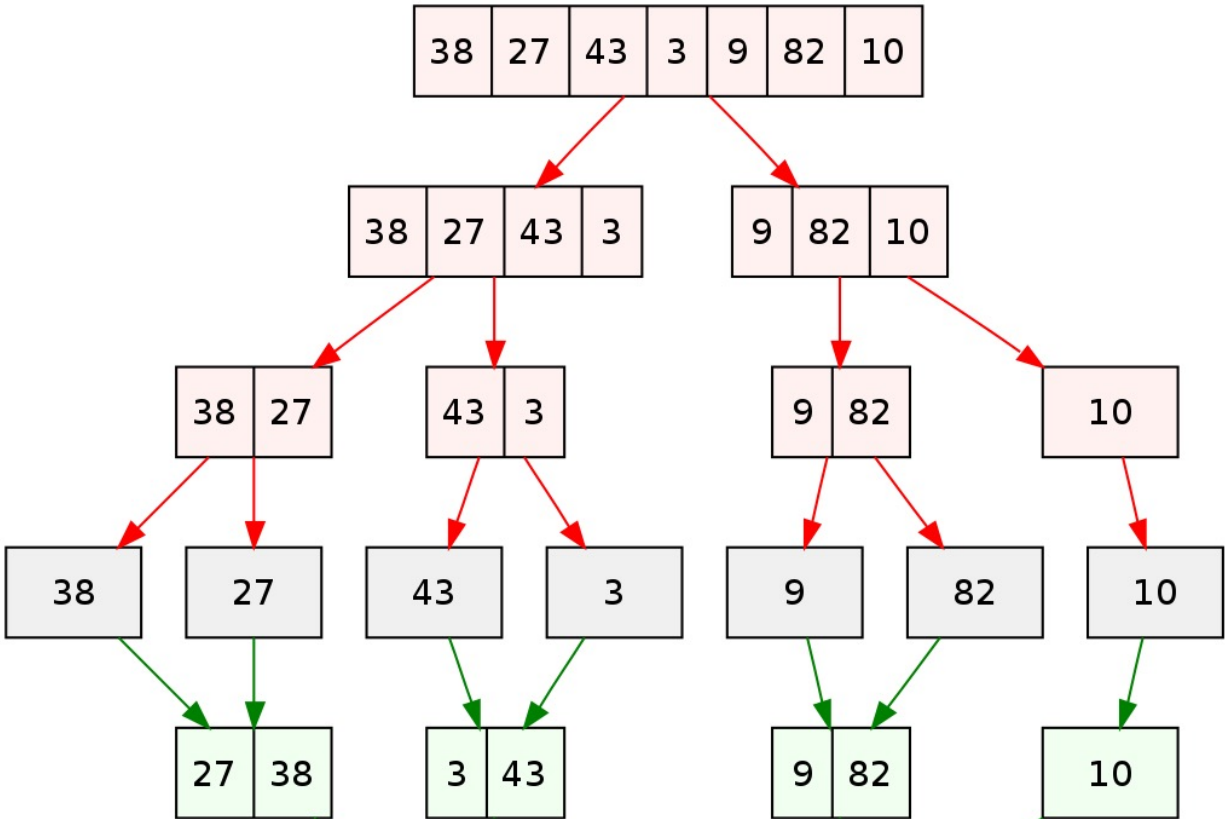
Example



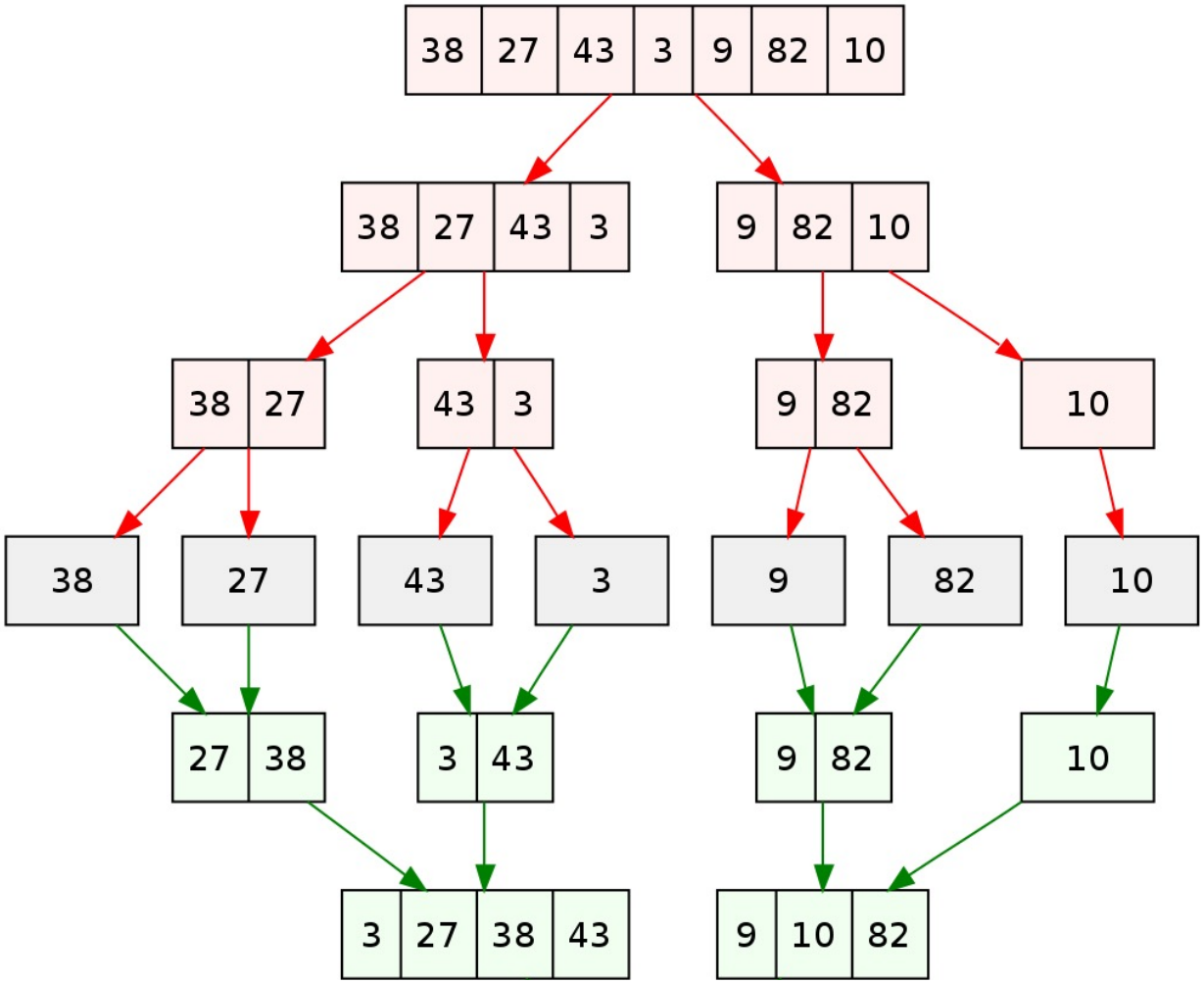
Example



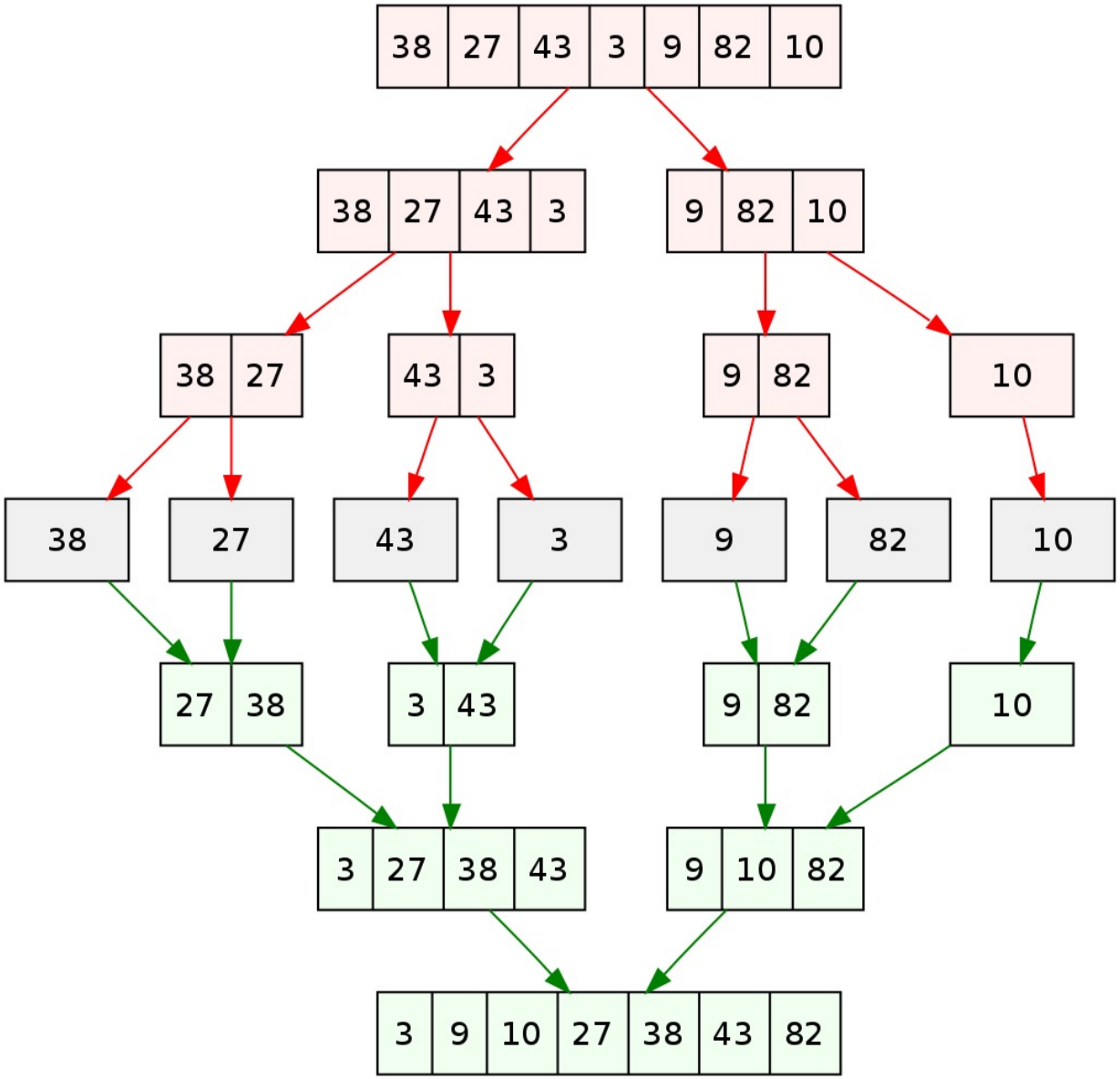
Example


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10

Example


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82

Example


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Analyzing divide-and-conquer algorithms

Let $T(n)$ be the running time on a problem of size n.

Divide-and-conquer algorithms often take the following form:

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

Where

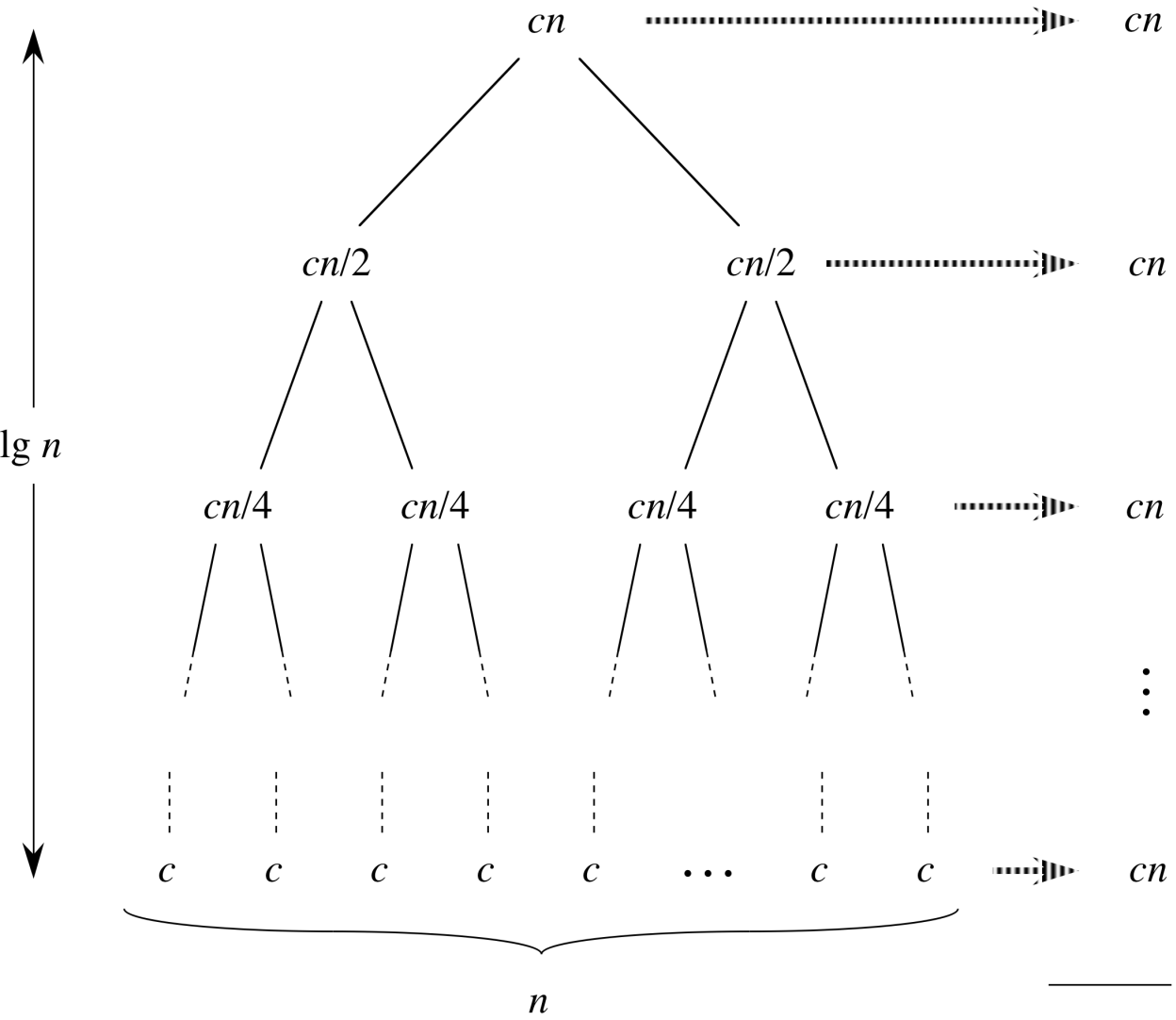
- a is the number of subproblems, each of which is of size n/b,
- $D(n)$ is the time to divide the problems into subproblems, and
- $C(n)$ is the time to combine solutions to subproblems into solutions to original problem

This recursive form is so common that we'll have an approach to immediately find the closed form theta notation using the **master theorem** (we'll see in CLRS ch. 4).

- Until then, let's see how to solve it using a recursion tree…

Analyzing merge sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{otherwise} \end{cases}$$

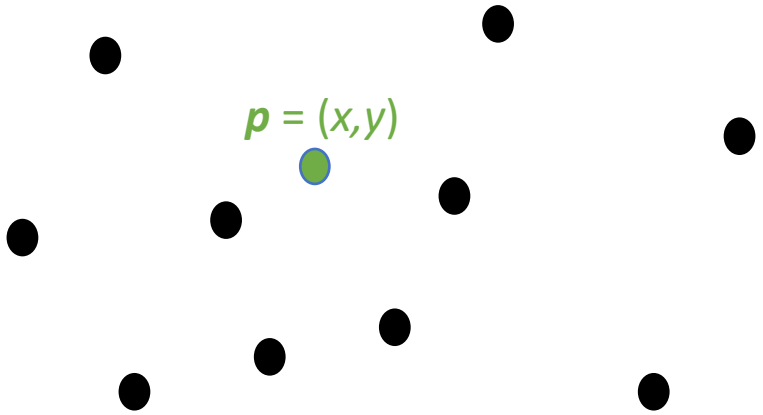


Total: $cn \lg n + cn$

Merge sort runs in $\Theta(n \log n)$ time

Other

You are given a query point ***p*** and a set ***S*** of ***n*** other points in two dimensional space. Find ***k*** points out of the ***n*** points which are nearest to ***p***.



Design an algorithm that will solve in the most efficient way as possible.
Spoiler: There are at least four approaches, all with different running times ☺