3 Divide and Conquer

Example: Sorting

9 8 7 6

6 7 8 9

Example: Sorting

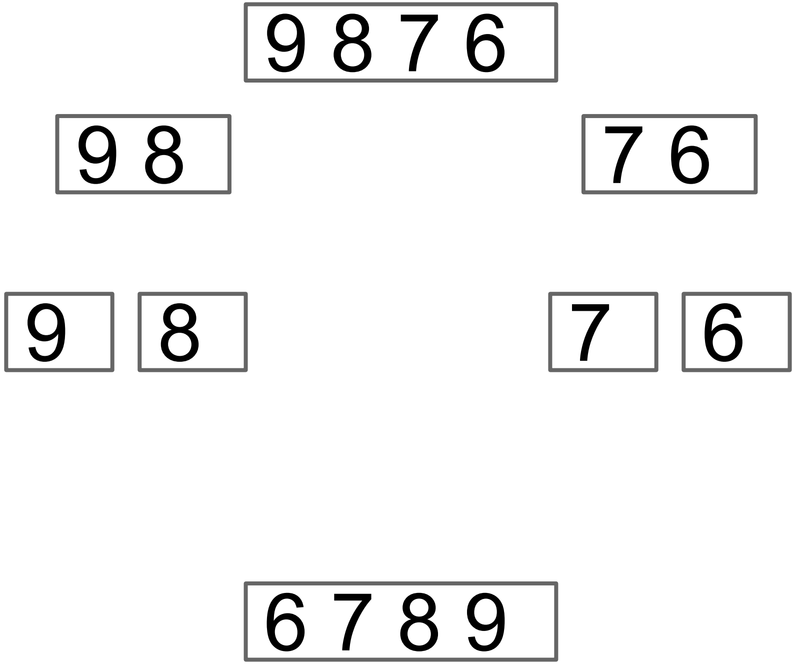
9 8 7 6

9 8

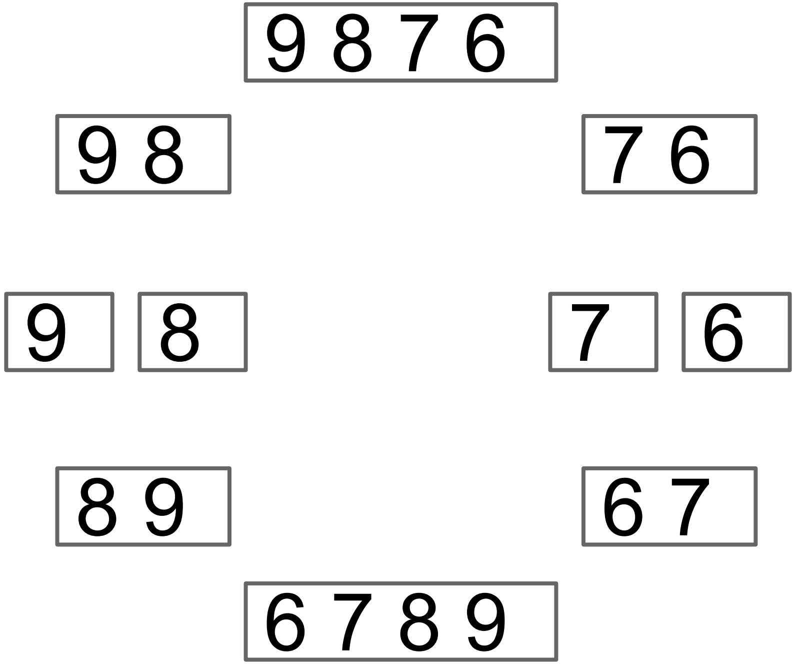
7 6

6 7 8 9

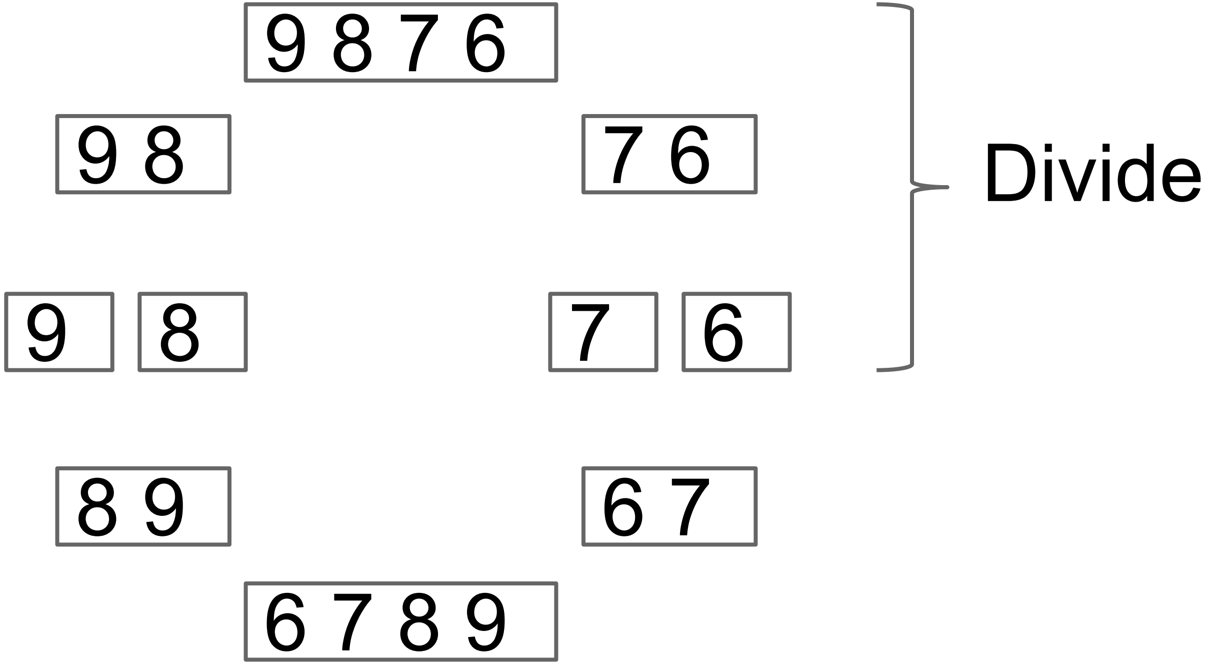
Example: Sorting



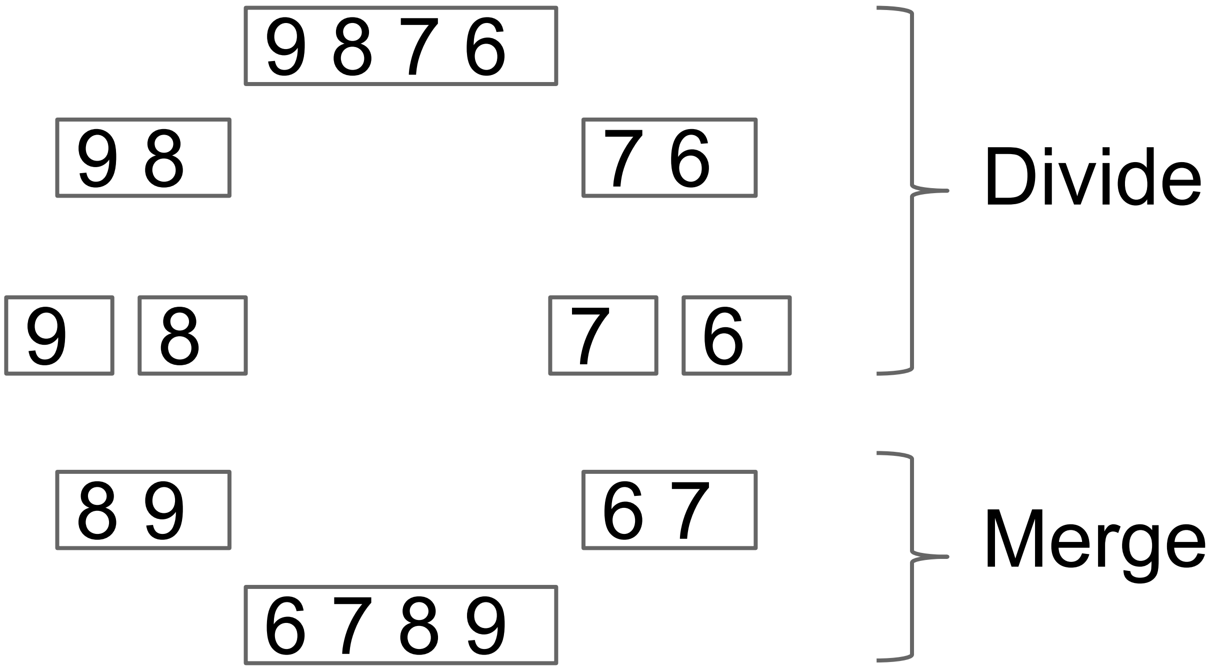
Example: Sorting



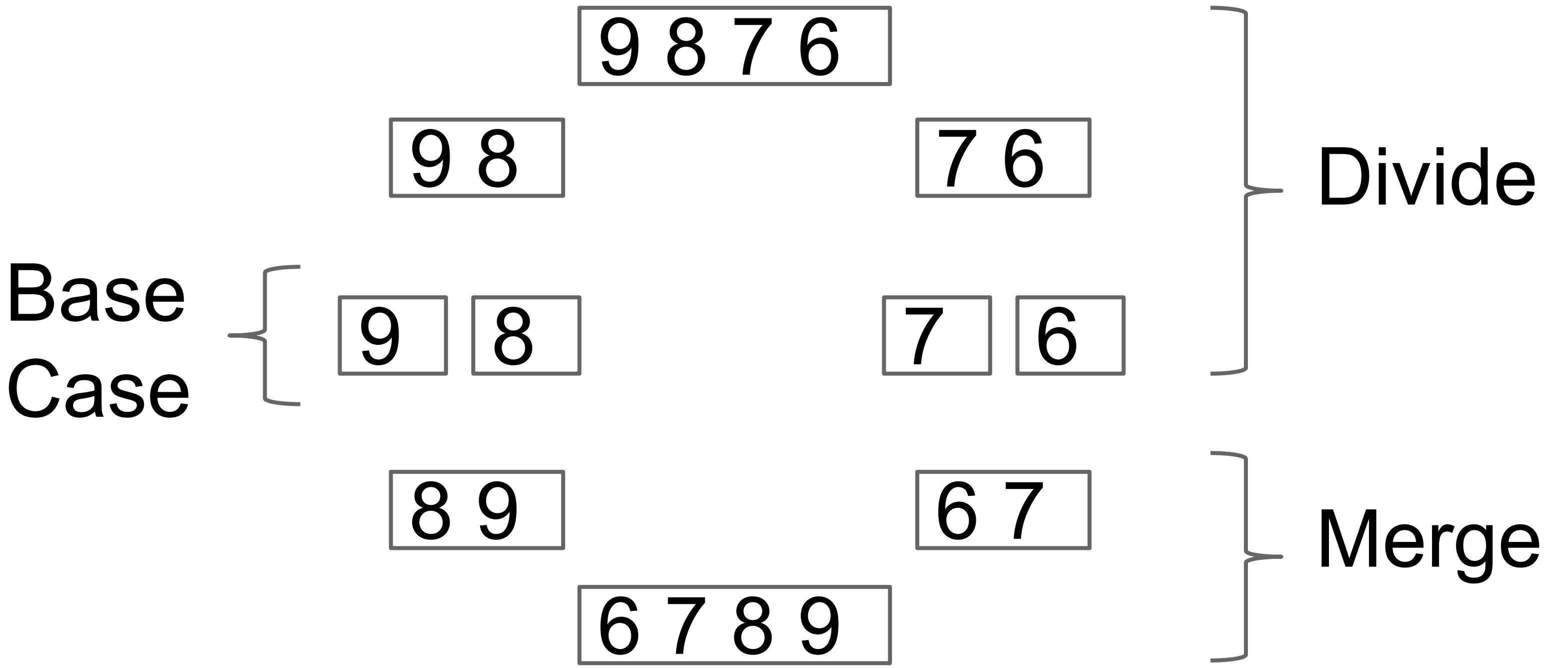
Example: Sorting



Example: Sorting



Example: Sorting



Example: Sorting

Merging - key insight:

- only compare p1, p2
- select and advance accordingly

sort(0, 1)	sort(2, 3)	sort(0, 3)
8 9	6 7
p1	p2

Example: Sorting

Merging - key insight:

- only compare p1, p2
- select and advance accordingly

sort(0, 1)	sort(2, 3)	sort(0, 3)
8 9	6 7	6
p1	p2	

Example: Sorting

Merging - key insight:

- only compare p1, p2
- select and advance accordingly

sort(0, 1) sort(2, 3) sort(0, 3)

8 9	6 7	6 7

p1 p2

Example: Sorting

Merging - key insight:

- only compare p1, p2
- select and advance accordingly

sort(0, 1)	sort(2, 3)	sort(0, 3)
8 9	6 7	6 7 8 9
p1	p2

Example: Sorting

Merging - key insight:

- only compare p1, p2
- select and advance accordingly
- very efficient - at most *n - 1 steps*

Divide and Conquer

1. Recursively *divide* into smaller problems

2. Solve small problems

3. Merge solutions **efficiently** (*conquer*)

Divide and Conquer

1. Recursively *divide* into smaller problems

2. Solve small problems

3. Merge solutions **efficiently** (*conquer*)

4. Looks **deceivingly simple**!

Binary Search

- find 2 in [1, 2, 3, 4, 5, 6]
  - find(0, 5) -> find(0, 3) ->
    find(2, 3) -> find(2, 2) -> found!
  - log(n) steps - fast lookup on sorted, random access data structures
- java.util.Collections.binarySearch(...)

Divide and Conquer

Speed-up problems (if applicable):

- Sort input data
- Use binary search to speed up lookup
- Total complexity (n items, k lookups):
  - O(nlog(n) + klog(n))

Exact Sum [11057]

Given a sequence of numbers, find a *pair* which adds to a given number S.

2 40 40 80 5 10 2 6 8 4 10	Peter should buy books whose prices are 40 and 40. Peter should buy books whose prices are 4 and 6.

Exact Sum - Solution

Idea 1: Complete Search?

- O(n^2) - n up to 10^5, too slow

Idea 2: Binary Search?

- Sort input data
- Iterate through it (i1) and binary search for (S - i1)
- O(n logn) - fast enough!

Lessons Learned

- Binary search solutions usually have edge cases - test extensively!
- Use the API of your programming language for binary search:
  - Collections.binarySearch / Arrays.binarySearch
  - STL: binary_search / upper_bound / lower_bound

Solve It 10341

Solve

$p * e^{-x} + q * \sin(x) + r * \cos(x) + s * \tan(x) + t * x^2 + u = 0$,

where $0 <= x <= 1$,

$0 <= p,r <= 20$ and $-20 <= q,s,t <= 0$

0 0 0 0 -2 1 1 0 0 0 -1 2 1 -1 1 -1 -1 1	0.7071 No solution 0.7554

Solve It - Bisection Method

f - continuous on [a..b], sign(f(a)) != sign(f(b))

⇒ f(r) = 0, for some r in [a..b]

Solve It - Bisection Method

f - continuous on [a..b], sign(f(a)) != sign(f(b))

⇒ f(r) = 0, for some r in [a..b]

```
findRoot(low, high):
  mid = (low + high) / 2
  if abs(f(r)) < EPS  return c       // found root
  if sign(f(r)) = sign(f(low))
   return findRoot(mid, high)        // root is in the other interval
  return findRoot(low, high)
```

Solve It [10341]

Solve

$p * e^{-x} + q * \sin(x) + r * \cos(x) + s * \tan(x) + t * x^2 + u = 0$,

where $0 <= x <= 1$,

$0 <= p,r <= 20$ and $-20 <= q,s,t <= 0$

- don't know if sign(f(0)) != sign(f(1))

Solve It 10341

Solve

$p * e^{-x} + q * \sin(x)+r * \cos(x) + s * \tan(x) + t * x^2 + u = 0,$

where $0 <= x <= 1$,

$0 <= p,r <= 20$ and $-20 <= q,s,t <= 0$

- don't know if sign(f(0)) != sign(f(1))
- but f is **decreasing**

⇒ no sol if sign(f(0)) == sign(f(1))

Lessons Learned

- Divide and conquer can be used
  - **directly** to design new algorithms
  - **to speed-up** lookups in sorted, random access DS
  - **to find answers** (limited precision/range)