Divide and Conquer

Algorithm Design Techniques

- Greedy
- Divide and Conquer
- Dynamic Programming
- Network Flows

Algorithm Design

	Greedy	Divide and Conquer
Formulate problem	?	?
Design algorithm	less work	more work
Prove correctness	more work	less work
Analyze running time	less work	more work

Divide and Conquer

Divide-and-conquer.

- Divide problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage:

- Problem of size n → two equal parts of size n/2
- Combine solutions in linear time.

Mergesort

Break up

| 13 | 17 | 6 | 3 | 9 | 2 | 16 | 1 |

| 13 | 17 | 6 | 3 | | 9 | 2 | 16 | 1 |

| 13 | 17 | | 6 | 3 | | 9 | 2 | | 16 | 1 |

Solve

| 13 | 17 | | 3 | 6 | | 2 | 9 | | 1 | 16 |

Combine

| 3 | 6 | 13 | 17 | | 1 | 2 | 9 | 16 |

| 1 | 2 | 3 | 6 | 9 | 13 | 16 | 17 |

Mergesort

```
mergesort(m, low, high) {
    if high == low {
        return
    }
    else if (high == low + 1) {
        sort m[low] and m[high];
        return;
    }
    else {
        middle = length(m) / 2
        mergesort(m, low, middle-1)
        mergesort(m, middle, high)
        return merge(m, low, middle, high)
    }
}
```

Solve base case

Divide

Solve recursively

Combine results

Mergesort

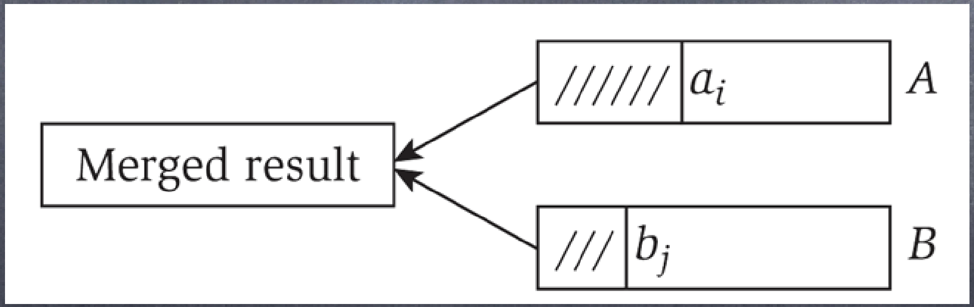
```
mergesort(m, low, high) {
    if high == low {
        return
    }
    else if (high == low + 1) {
        sort m[low] and m[high];
        return;
    }
    else {
        middle = length(m) / 2
        mergesort(m, low, middle-1)
        mergesort(m, middle, high)
        return merge(m, low, middle, high)
    }
}
```

Complexity?

Base case - O(1)
Divide - O(1)
Recursive cases ??
Merge - O(n)

Accounting: Merge Sorted Lists

- Input: sorted lists $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$
- Output: combined sorted list



Accounting: Merge Two Sorted Lists

```
i = 1, j = 1
while (both lists are nonempty) {
    if (a_i ≤ b_j) {
      append a_i to output list
      increment i
    }
    else {
      append b_j to output list
      increment j
    }
}
append remainder of nonempty list
to output list
```

Accounting scheme: each entry from input list is touched once

$\rightarrow O(n)$

mergesort Recurrence Relation

- $T(n)$ = running time for input of size n

$T(n) \leq 2\,T(n/2) + cn$ when $n > 2$

$T(2) \leq c$

Problem: How do we solve this for a O() value?

Generalized Recurrence Problem

- Instead of dividing the problem into 2 subproblems, divide it into q subproblems.
- Still have linear cost for the divide and merge steps combined.
- Consider 2 cases:
  - $q = 1$
  - $q > 2$

Summary

- Divide and conquer where:
  - $O(n)$ work is done for divide and merge combined
  - Subproblems have size $n/2$
- One subproblem on each recursion => $O(n)$
- 2 subproblems on each recursion => $O(n \log n)$
- >2 subproblems on each recursion => $O(n^{\log q})$