Divide and Conquer Summary

• Divide
  – Identify one or more subproblems

• Conquer
  – Solve some or all of those subproblems

• Combine
  – Use the subproblems’ solutions to find large solution
def myDCalgo(problem):
    if baseCase(problem):
        solution = solve(problem)  # brute force if necessary
        return solution
    subproblems = Divide(problem)
    for sub in subproblems:
        subsolutions.append(myDCalgo(sub))
    solution = Combine(subsolutions)
    return solution
Karatsuba

1. Break into smaller subproblems

\[
\begin{align*}
a \ b &= 10^{\frac{n}{2}} a + b \\
\times c \ d &= 10^{\frac{n}{2}} c + d \\
10^n (a \times c) + \\
10^n (a \times d + b \times c) + \\
(b \times d)
\end{align*}
\]
Karatsuba

\[10^n(ac) + 10^{2n}(ad + bc) + bd\]

Can’t avoid these

This can be simplified

\[(a + b)(c + d) = ac + ad + bc + bd\]

\[ad + bc = (a + b)(c + d) - ac - bd\]

Two multiplications

One multiplication
def dc_mult(x, y):
    n = length of larger of x, y
    if n == 1:
        return x*y
    a = first n/2 digits of x
    b = last n/2 digits of x
    c = first n/2 digits of y
    d = last n/2 digits of y
    ac = dc_mult(a, c)
    bd = dc_mult(b, d)
    adbc = dc_mult(a+b, c+d) - ac - bd
    return ac*10^n + (adbc)*10^(n/2) + bd
Divide and Conquer Summary

• Divide
  – Identify one or more subproblems:

• Conquer
  – Solve some or all of those subproblems:

• Combine
  – Use the subproblems’ solutions to find large solution:
Generic Divide and Conquer Solution

def myDCalgo(problem):
    if baseCase(problem):
        solution = solve(problem)  # brute force if necessary
        return solution

subproblems = Divide(problem)
for sub in subproblems:
    subsolutions.append(myDCalgo(sub))

solution = Combine(subsolutions)
return solution
Karatsuba Pseudocode

def dc_mult(x, y):
    n = length of larger of x,y
    if n == 1:
        return x*y
    a = first n/2 digits of x
    b = last n/2 digits of x
    c = first n/2 digits of y
    d = last n/2 digits of y
    ac = dc_mult(a, c)
    bd = dc_mult(b, d)
    adbc = dc_mult(a+b, c+d) - ac - bd
    return ac*10^n + (adbc)*10^(n/2) + bd

\[ T(n) = 3T\left(\frac{n}{2}\right) + O(n) \]
Maximum Sum Continuous Subarray Problem

The maximum-sum subarray of a given array of integers $A$ is the interval $[a, b]$ such that the sum of all values in the array between $a$ and $b$ inclusive is maximal.

Given an array of $n$ integers (may include both positive and negative values), give a $O(n \log n)$ algorithm for finding the maximum-sum subarray.
Naïve Solution

\[
\begin{array}{cccccccccccccc}
5 & 8 & -4 & 3 & 7 & -15 & 2 & 8 & -20 & 17 & 8 & -50 & -5 & 22 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13
\end{array}
\]
What does a $n \log n$ recurrence look like?

• $T(n) =$
Tree method

\[ T(n) = 2T\left(\frac{n}{2}\right) + n \]

\[ \Rightarrow n \text{ total / level} \]

\[ \log_2 n \text{ levels of recursion} \]

\[ T(n) = \sum_{i=1}^{\log_2 n} n = n \log_2 n \]
Divide and Conquer $\Theta(n \log n)$

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>-4</td>
<td>3</td>
<td>7</td>
<td>-15</td>
<td>2</td>
<td>8</td>
<td>-20</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

- Recursively Solve on Left
- Divide in half
- Recursively Solve on Right
## Divide and Conquer $\Theta(n \log n)$

### Divide in half

Recursive Solve on Left

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>-4</td>
<td>3</td>
<td>7</td>
<td>-15</td>
<td>2</td>
</tr>
</tbody>
</table>

Recursive Solve on Right

<table>
<thead>
<tr>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>-20</td>
<td>17</td>
<td>8</td>
<td>-50</td>
<td>-5</td>
<td>22</td>
</tr>
</tbody>
</table>

### Find Largest sum that spans the cut

**Largest sum that ends here** + **Largest sum that starts here**

**19** + **25** = **44**

### Example

```
6 1 -7 -3 -6 -13 2 8 -12 5 13 -37 -42 -20
```

**Largest sum that ends here**

-7 -3 -6 -13 2 = **-27**

**Largest sum that starts here**

5 8 -4 3 7 = **16**

**Largest sum that spans the cut**

-27 + 16 = **-11**
Divide and Conquer $\Theta(n \log n)$

Return the Max of
Left, Right, Center

Recursive Solve on Left
19

Divide in half

Find Largest sum that spans the cut
19

Recursive Solve on Right
25

\[ T(n) = 2T\left(\frac{n}{2}\right) + n \]
Divide and Conquer Summary

• **Divide**
  – Break the list in half

• **Conquer**
  – Find the best subarrays on the left and right

• **Combine**
  – Find the best subarray that “spans the divide”
  – I.e. the best subarray that ends at the divide concatenated with the best that starts at the divide

Typically multiple subproblems. Typically all roughly the same size.
Generic Divide and Conquer Solution

def myDCalgo(problem):
    if baseCase(problem):
        solution = solve(problem) #brute force if necessary
        return solution

subproblems = Divide(problem)
for sub in subproblems:
    subsolutions.append(myDCalgo(sub))

solution = Combine(subsolutions)
return solution
MSCS Divide and Conquer $\Theta(n \log n)$

def MSCS(list):
    if list.length < 2:
        return list[0]  # list of size 1 the sum is maximal
    {listL, listR} = Divide(list)
    for list in {listL, listR}:
        subSolutions.append(MSCS(list))
    solution = max(solnL, solnR, span(listL, listR))
    return solution