Divide and Conquer

Divide and Conquer

1. Multiplying two large numbers (thousands of bits long)
    - Useful in applications like RSA
2. Median
    - Find the median without first sorting the list
3. Fast Fourier Transform
    - Most important numerical algorithm of our lifetime
    - Complex numbers
    - Recursive approach

Divide and Conquer: Overview

1. Examples:
    - MergeSort
    - Fast modular exponentiation algorithm
    - Euclid's GCD algorithm
    - Now: multiplying n-bit integers
        - Given n-bit integers x and y
        - Goal: compute z = xy
        - Faster than $O(n^2)$ time
    - Median
    - FFT

Multiplying Complex Numbers

1. Setting: Multiplication is expensive
    - Adding/subtracting is cheap
2. Two complex numbers: a + bi, c + di
    - (a + bi)(c + di)
    - ac - bd + i(bc + ad)
        - 4 real number multiplications, 3 additions/subtractions
        - Compute bc + ad without individual terms

Improved Approach

1. Two complex numbers: a + bi, c + di
    - (a + bi)(c + di)
    - ac - bd + i(bc + ad)
    - (a + b)(c + d) = ac + bd + (bc + ad)
    - (bc + ad) = (a + b)(c + d) - ac - bd
        - Substitute this in
    - (a + bi)(c + di) = ac - bd + ((a + b)(c + d) - ac - bd)
        - Compute ac, bd, and (a+b)(c+d)
        - Only 3 expensive multiplications

Divide and Conquer: Naive Approach

1. Input: n-bit integers x and y
2. Goal: Compute z = xy (running time in terms of n)
3. D&C idea: Break input into 2 halves
    - $x = x_l$ and x_r
        - Break x into first n/2 bits and last n/2 bits

- $y = y_l$ and y_r
  - Break y into first n/2 bits and last n/2 bits
- $x = 182 = 10110110$
  - $x_l = 1011$, $x_r = 0110$
  - $182 = 11 * 2^4 + 6$

Naive: Recursive Idea

1. Partition x and y
   - $x = x_l + 2^{n/2} + x_r$
   - $y = y_l + 2^{n/2} + y_r$
   - $xy = 2^n x_l y_l + 2^{n/2}(x_l y_r) + x_r y_r$

Naive: Pseudocode

```
def EasyMultiply(x,y):
    # input: n-bit integers x and y, n = 2^k
    # output: z = xy
    xl = first n/2 bits of x, xr = last n/2 bits of x
    yl = first n/2 bits of y, yr = last n/2 bits of y
    A = EasyMultiply(xl, yl)
    B = EasyMultiply(xr, yr)
    C = EasyMultiply(xl, yr)
    D = EasyMultiply(xr, yl)
    z = A * 2 ** n + (C + D) * 2 ** (n/2) + B
    return z
```

Naive: Running Time

1. Partitioning x and y is O(n)
2. Calls to EasyMultiply is 4T(n/2)
3. Calculating z is O(n)
4. Let T(n) = worst-case running time of EasyMultiply on input of size n
   - $T(n) = 4T(n/2) + O(n) = O(n^2)$

Divide and Conquer: Improved Approach

1. $xy = 2^n x_l y_l + 2^{n/2}(x_l y_r) + x_r y_r$
   - $(x_l + x_r)(y_l + y_r) = x_l y_l + x_r y_r + (x_l y_r + x_r y_l)$
   - $(x_l y_r + x_r y_l) = (x_l + x_r)(y_l + y_r) - x_l y_l - x_r y_r$

Improved: Pseudocode

```
def FastMultiply(x,y):
    # input: n-bit integers x and y, n = 2^k
    # output: z = xy
    xl = first n/2 bits of x, xr = last n/2 bits of x
    yl = first n/2 bits of y, yr = last n/2 bits of y
    A = FastMultiply(xl, yl)
    B = FastMultiply(xr, yr)
    C = FastMultiply(xl+xr, yl+yr)
    z = A * 2 ** n + (C - A - B) + B
    return z
```

Improved: Running Time

1. $T(n) = 3T(n/2) + O(n)$
   - $<= cn + 3T(n/2)$
   - $<= cn + 3(cn/2 + 3T(n/2^2)$
   - $<= cn(1 + 3/2) + 3^2(cn / 2^2 + 3T(n/2^3))$
   - $= O(n * (3/2)^{\log_2 n})$
   - $= O(3^{\log_2 n})$
   - $= O(n^{\log_2 3})$
   - $\log_2 3 = 1.59$
2. Running time is $O(n^{\log_2 3})$

Improved: Summary

1. Example: x = 182, y = 154
   - x = 10110110
   - y = 10011010
   - $x_l = 1011 = 11$
   - $x_r = 0110 = 6$
   - $y_l = 1001 = 9$
   - $y_r = 1010 = 10$
   - 11 * 9 = 99
   - 6 * 10 = 60
   - (ll + 6)(9 + 10) = 323
   - 182 * 154 = 99 * 256 _ (323 - 99 - 60) * 16 + 60 = 28028
   - Strassen's algorithm: Similar idea for multiplying matrices