EECS 336: Introduction to Algorithms	**Lecture 6**
Divide and Conquer	Mergesort, Recurrences, Integer Mult.

Reading: 5.0-5.5.

Last time:

- Dynamic Greedy
- Dijkstra, Prim.

Today:

- Divide and Conquer
- Mergesort
- Recurrences
- Integer Mult.

Divide and Conquer

- divide problem into subproblems
- solve subproblems
- merge solutions to solve original.

Example: sorting

Algorithm: Mergesort(U):

1. if $|U| \leq 1$, return U
2. split U in half: U_1, U_2
3. sort U_1 and U_2 separately:
   - $S_1 = \text{mergesort}(U_1)$
   - $S_2 = \text{mergesort}(U_2)$
4. join sorted lists:

$$S = \text{merge}(S_1, S - 2)$$

Subroutine: Merge(S_1, S_2)

5. $S = \emptyset$
6. identity S_i with minimum elt.
7. remove min from S_i and append to S
8. repeat.

Correctness: induction.

Runtime

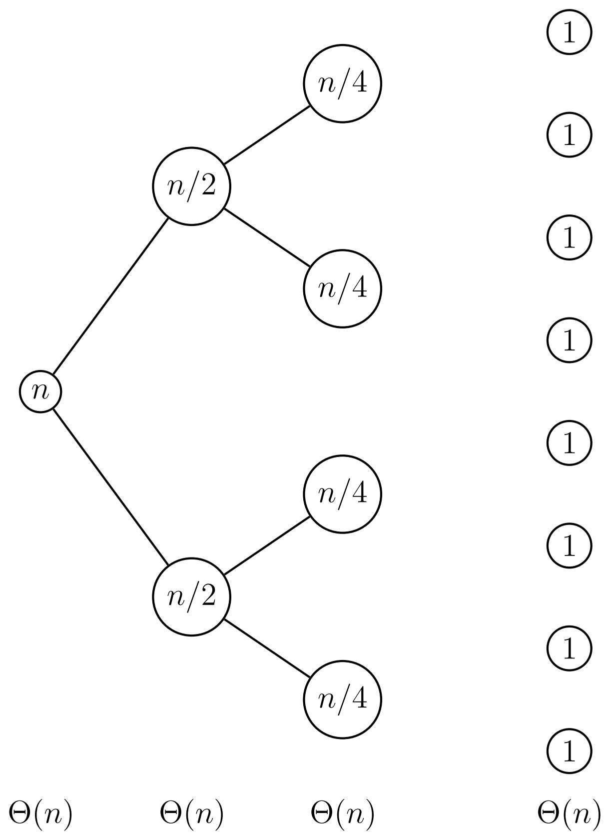
- Merge: $|S_1| + |S_2| = |S| = n$.
- Mergesort: $T(n)$

Recurrence:

- $T(n) = 2T(n/2) + n$
- $T(1) = 1$

[[*What is $T(n)$?*]]

Solving Recurrences by Unrolling



[[*how much work in each level, total?*]]

[[*how many levels?*]]

$T(n) =$ "work per level" $\times$ "number of levels" $= n \log n$.

Theorem: Mergesort runs in $O(n \log n)$.

Public Key Cryptography

"send private messages over insecure channels"

Number Theory

Easy to find large r, e, and d such that,

> **Fact:** $\forall\ m,\ m^{ed} \equiv m \pmod{r}$
>
> **Assumption:** given r, e, and $x \equiv m^e \pmod{r}$ it is hard to compute m
> [[*"discrete logarithm"*]]

Scenario: Alice wants to send private message m to Bob.

Procedure:

- Bob finds r, e, d.
  - $(r, d) =$ private key.
  - $(r, e) =$ public key.
- Bob publishes (r, e).
- Alice
  - computes $x = m^e \pmod{r}$
  - sends x to Bob.
- Bob
  - receives x
  - computes $y = x^d \pmod{r}$

From Fact: $y = m$.

Question: Can we do this efficiently?

- e is a large number (n bits, e.g., 256) [[$2^{256} \approx 10^{22}$]]
- $m^e = \underbrace{m \cdot m \cdots m}_{e \text{ times}}$
- brute force algorithm runs in $e = 2^n$ steps

  $\Rightarrow$ exponential!!

Problem 1: modular exponentiation

Input: number x, modulus r, exponent e

Output: $z \equiv x^e \pmod r$

$\left[\left[\text{\textit{if we didn't take modulus, number would get very big}}\right]\right]$

[[*How can we divide and conquer?*]]

Idea:

- if $e = e_1 + e_2$ then $x^e \equiv x^{e_1} x^{e_2} \pmod r$
- if $e_1 = e_2$ can solve x^{e_1} and square.

Algorithm: Repeated Squaring

1. if $e = 1$ return x.
2. $e' = \lfloor e/2 \rfloor$.
3. $y = \text{repeated-square}(x, e')$.
4. if e odd

   return $y \cdot y \cdot x \pmod r$

5. else

   return $y \cdot y \pmod r$

Runtime

Let $T_m(e)$ = number of multiplies.

$$\begin{aligned} &T_m(e) = T_m(\lfloor e/2 \rfloor) + 2 \\ &T_m(1) = 0 \end{aligned}$$

Solving Recurrence by Guessing

Guess $T_m(e) \leq d \log e$ [[*for some d*]]

Inductively Verify:

base: $T_m(1) = 0 \leq d \log 1 = 0$.

I.H.: assume true for $e' < e$

I.S.:

$$\begin{aligned} T_m(e) &= T_m(\lfloor e/2 \rfloor) + 2 \\ &\leq d \log(e/2) + 2 \\ &= d \log e - d \log 2 + 2 \\ &= d \log e - d + 2 \\ &= d \log e. \quad (\text{choose } d = 2) \end{aligned}$$

Recall: $n = \log e$.

Theorem: repeated squaring on an n bit number takes $O(n)$ multiplies.

Problem 2: Integer multiplication

input: n bit integers x, y.

output: $2n$ bit integer $z = x \cdot y$.

Algorithm: elementary school multiply

```
        101101
      x 010110
    ----------
        000000
       101101
      101101
     000000
    101101
 + 000000
--------------
whatever
```

Runtime: $T(n) = O(n^2)$.

[[*can we do better?*]]

Idea:

1. separate high order from low order bids
   - $k = n/2$ [[*assume n even*]]
   - x_H = high k bits of x
   - x_L = low k bits of x

   $\Rightarrow x = x_H 2^k + x_L$.

2. $x \cdot y = (x_H 2^k + x_L)(y_H 2^k + y_L)$

   $= x_H y_H 2^n + (x_L y_H + x_H y_L) 2^k + x_L y_L$

$\Rightarrow$ one n bit mult requires 4 $n/2$ bit mults

[[*mult by 2^k is bit shift (easy)*]]

$\Rightarrow T(n) = 4T(n/2) + cn$

[[*additions require cn time*]]

$= O(n^2)$.

[[*need a better idea!*]]

- let $H = x_H y_H$; $L = x_L y_L$; and $Z = x_H y_L + x_L y_H$

[[**Q:** *compute H, L, and Z in < 4 mults?*]]

Idea:

- $P = (x_H + x_L)(y_H + y_L)$

  $= x_H y_H + x_H y_L + x_L y_H + x_L y_L$

  $= H + Z + L$

3. Rearrange: $Z = P - H - L$

$\Rightarrow xy = H2^n + (P - H - L)2^k + L$

$\Rightarrow$ 3 size $n/2$ mults needed.

Runtime: $T(n) = 3T(n/2) + cn$

$= O(n^{\log_2 3}) = O(n^{1.59})$.

[[*THIS SHOULD BE SURPRISING!*]]

(Google: Arthur Benjamin does "Math-emagic")

```
35 x 51
= 15x100 + (8 * 6 - 15 - 5)x10 + 5
=           \_____ 28 _____/
= 1785
```