Algorithm Design Techniques

- **Divide & Conquer**
  - Reduce problem to one or more sub-problems of the same type
  - Typically, each sub-problem is at most a constant fraction of the size of the original problem
  - e.g. Mergesort, Binary Search, Strassen's Algorithm, Quicksort (kind of)

### Fast Exponentiation

**Power(a, n)**

- **Input:** integer n and number a
- **Output:** a^n

**Obvious algorithm**

- n-1 multiplications

**Observation:**

- if n is even, n=2m, then a^n = a^m * a^m

### Divide & Conquer Algorithm

**Power(a, n)**

- if n=0 then return(1)
- else if n=1 then return(a)
- else
  - x ← Power(a, \lfloor n/2 \rfloor)
  - if n is even then return(x * x)
  - else return(a * x * x)

### Analysis

- Worst-case recurrence
  - T(n)=T(n/2)+2 for n=1
  - T(1)=0
  - Time
  - $T(n)=T(n/2)+2 = T(n/4)+2+2 = \ldots = T(1)+2\log_2 n = 2\log_2 n$ (log n copies)

- More precise analysis:
  - $T(n) = \lceil \log_2 n \rceil + \# of 1's in n's binary representation$

### A Practical Application - RSA

- Instead of $a^n$ want $a^n \mod N$
  - $a^n \mod N = ((a^i \mod N)\cdot(a^{n-i} \mod N)) \mod N$

- same algorithm applies with each x-y replaced by
  - $(x \mod N)\cdot(y \mod N) \mod N$

- In RSA cryptosystem (widely used for security)
  - need $a^n \mod N$ where a, n, N each typically have 1024 bits
  - Power: at most 2048 multiplies of 1024 bit numbers
  - relatively easy for modern machines
  - Naive algorithm: $2^{1024}$ multiplies
Binary search for roots (bisection method)

- Given:
  - continuous function \( f \) and two points \( a < b \) with \( f(a) = 0 \) and \( f(b) > 0 \)
- Find:
  - approximation to \( c \) s.t. \( f(c) = 0 \) and \( a < c < b \)

Bisection method

\[
\text{Bisection}(a, b, \epsilon) \quad \begin{cases} 
\text{if } (a-b) < \epsilon \text{ then } & \text{return}(a) \\
\text{else } & c \leftarrow (a+b)/2 \\
& \text{if } f(c) = 0 \text{ then } \text{return}(\text{Bisection}(c, b, \epsilon)) \\
& \text{else } \text{return}(\text{Bisection}(a, c, \epsilon))
\end{cases}
\]

Time Analysis

- At each step we halved the size of the interval
- It started at size \( b-a \)
- It ended at size \( \epsilon \)
- \# of calls to \( f \) is \( \log_2 \left( \frac{b-a}{\epsilon} \right) \)

Mergesort (review)

Mergesort: (recursively) sort 2 half-lists, then merge results.

\[
\begin{align*}
T(n) &= 2T(n/2) + cn, \quad n \geq 2 \\
T(1) &= 0 \\
\text{Solution: } &\Theta(n \log n)
\end{align*}
\]

Why Balanced Subdivision?

- Alternative "divide & conquer" algorithm:
  - Sort first \( n-1 \)
  - Sort last 1
  - Merge them
- Recurrence
  - \( T(n) = T(n-1) + T(1) + 3n \) for \( n \geq 2 \)
  - \( T(1) = 0 \)
- Solution:
  - \( 3n + 3(n-1) + 3(n-2) \ldots = \Theta(n^2) \)

Another D&C Approach

- Suppose we've already invented DumbSort, taking time \( n^2 \)
- Try Just One Level of divide & conquer:
  - DumbSort(first \( n/2 \) elements)
  - DumbSort(last \( n/2 \) elements)
  - Merge results
- Time:
  - \( (n/2)^2 + (n/2)^2 + n = n^2/2 + n \)
  - Almost twice as fast!
Some Divide & Conquer morals

Moral 1:
- Two problems of half size are better than one full-size problem, even given the $O(n)$ overhead of recombining, since the base algorithm has superlinear complexity.

Moral 2:
- If a little's good, then more's better
  - 2 levels of D&C would be almost 4 times faster, 3 levels almost 8, etc., even though overhead is growing.
  - Best is usually full recursion down to some small constant size (balancing "work" vs "overhead").

Divide & Conquer morals

Moral 3: unbalanced division less good:
- $(.1n)^2 + (.9n)^2 + n = .82n^2/2 + n$
- The 18% savings compounds significantly if you carry recursion to more levels, actually giving $O(n \log n)$, but with a bigger constant.
- Worth doing if you can’t get 50-50 split, but balanced is better if you can.
- This is intuitively why Quicksort with random splitter is good – badly unbalanced splits are rare, and not instantly fatal.
- $(1)^2 + (n-1)^2 + n = n^2 - 2n + 2 + n$
- Little improvement here.

Sometimes two sub-problems aren't enough

More general divide and conquer
- You've broken the problem into a different sub-problems
- Each has size at most $n/b$
- The cost of the break-up and recombining the sub-problem solutions is $O(n^k)$

Recurrence
- $T(n) = aT(n/b) + c \cdot n^k$

Master Divide and Conquer Recurrence
- If $T(n) = a \cdot T(n/b) + c \cdot n^k$ for $n > b$ then
  - if $a \cdot b^k$ then $T(n)$ is $O(n^{\log_b a})$
  - if $a = b^k$ then $T(n)$ is $O(n^k)$
  - if $a = 1$ then $T(n)$ is $O(n^k \log n)$
- Works even if it is $\lceil n/b \rceil$ instead of $n/b$.

Proving Master recurrence

Problem size
\begin{align*}
n & \rightarrow \text{aT(n/b)+cn}^k & \text{# probs} \\
n/b & \rightarrow \text{a} & 1 \\
n/b^2 & \rightarrow \text{a} & 1 \\
b & \rightarrow \text{ad} & \text{a} \\
1 & \rightarrow \text{T(1)=c} & \text{a} \\
\end{align*}
Proving Master recurrence

Geometric Series

Total Cost