COMP251: Divide-and-Conquer
(1)

Jérôme Waldispühl & Guilia Alberini
School of Computer Science
McGill University

Based on (Kleinberg & Tardos, 2005) and slides by K. Wayne & Snoeyink
## Algorithm design techniques

<table>
<thead>
<tr>
<th></th>
<th>Greedy choice</th>
<th>Optimal Substructure</th>
<th>Recursion</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Kruskall, Huffman, Dijkstra...</td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>Weighted interval scheduling, Bellman-Ford...</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td></td>
<td></td>
<td>✓</td>
<td>MergeSort, Karatsuba...</td>
</tr>
</tbody>
</table>
Outline

• MergeSort
  - Definition
  - Correctness
  - Complexity analysis

• Integer multiplication
  - “Naïve” recursive algorithm
  - Karatsuba

Objective: Designing a divide-and-conquer algorithm and characterizing its running time.
Divide and Conquer

• Recursive in structure
  – *Divide* the problem into sub-problems that are similar to the original but smaller in size
  – *Conquer* the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  – *Combine* the solutions to create a solution to the original problem
An Example: Merge Sort

**Sorting Problem:** Sort a sequence of $n$ elements into non-decreasing order.

- **Divide:** Divide the $n$-element sequence to be sorted into two subsequences of $n/2$ elements each
- **Conquer:** Sort the two subsequences recursively using merge sort.
- **Combine:** Merge the two sorted subsequences to produce the sorted answer.
Merge Sort - Example

Divide

Merge
Merge-Sort \((A, p, r)\)

**INPUT:** a sequence of \(n\) numbers stored in array \(A\)

**OUTPUT:** an ordered sequence of \(n\) numbers

\[
\text{MergeSort} (A, p, r) \quad // \text{sort } A[p..r] \text{ by divide & conquer}
\]

1. \textbf{if} \(p < r\)
2. \textbf{then} \(q \leftarrow \lfloor (p+r)/2 \rfloor\)
3. \textit{MergeSort} \((A, p, q)\)
4. \textit{MergeSort} \((A, q+1, r)\)
5. \textit{Merge} \((A, p, q, r)\) \(//\) merges \(A[p..q]\) with \(A[q+1..r]\)

**Initial Call:** \text{MergeSort}(A, 1, n)
**Procedure Merge**

**Merge**($A, p, q, r$)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. for $i \leftarrow 1$ to $n_1$
   4.   do $L[i] \leftarrow A[p + i - 1]$
5. for $j \leftarrow 1$ to $n_2$
   6.   do $R[j] \leftarrow A[q + j]$
7. $L[n_1+1] \leftarrow \infty$
8. $R[n_2+1] \leftarrow \infty$
9. $i \leftarrow 1$
10. $j \leftarrow 1$
11. for $k \leftarrow p$ to $r$
   12.   do if $L[i] \leq R[j]$
       13.       then $A[k] \leftarrow L[i]$
       14.       $i \leftarrow i + 1$
       15.       else $A[k] \leftarrow R[j]$
       16.       $j \leftarrow j + 1$

**Input:** Array containing sorted subarrays $A[p..q]$ and $A[q+1..r]$.

**Output:** Merged sorted subarray in $A[p..r]$.

**Sentinels**, to avoid having to check if either subarray is fully copied at each step.
Merge (example)

Execution of a call to Merge(A,p,q,r):

The first and second half of A[p,r] are already sorted.

Merge creates an array for each first and second halves.
Merge (example)

Execution of a call to Merge(A,p,q,r):

Pick the lowest value in L and R and place it at the lowest index not used yet (i.e., index \( k \))

Note: We sort in-place
Merge (example)

Execution of a call to Merge(A,p,q,r):

```
A: p 1 4 12 24 q q+1 1 4 8 13 r
   k

L: 6 7 12 24 \infty
   i

R: 1 4 8 13 \infty
   j
```

Iterate the same process
Merge (example)

Execution of a call to Merge(A,p,q,r):

- **A**: Array with elements...
- **p**, **q**, **q+1**, **r**: Subarrays 1, 4, 6, 24, 1, 4, 8, 13, ...
- **k**: Middle of the array
- **L**: Left subarray
- **R**: Right subarray
- **i**, **j**: Indices of subarrays
Merge (example)

Execution of a call to Merge(A, p, q, r):

A

\[
\begin{array}{cccccccc}
\ldots & p & q & q+1 & r \\
6 & 7 & 1 & 4 & 6 & 7 & 1 & 4 & 8 & 13 & \ldots \\
\end{array}
\]

\[
A = \begin{array}{cccccccc}
\ldots & 1 & 4 & 6 & 7 & 1 & 4 & 8 & 13 & \ldots \\
\end{array}
\]

k

L

\[
\begin{array}{cccc}
6 & 7 & 12 & 24 & \infty \\
\end{array}
\]

R

\[
\begin{array}{cccc}
1 & 4 & 8 & 13 & \infty \\
\end{array}
\]

i

j
Execution of a call to $\text{Merge}(A,p,q,r)$:

<table>
<thead>
<tr>
<th>A</th>
<th>p</th>
<th>q</th>
<th>q+1</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Execution of a call to Merge(A,p,q,r):

A

\[ \begin{array}{cccccccc}
... & 1 & 4 & 6 & 7 & 8 & 12 & 8 & 13 & ... \\
\end{array} \]

k

L

\[ \begin{array}{cccc}
6 & 7 & 12 & 24 & \infty \\
\end{array} \]

i

R

\[ \begin{array}{cccc}
1 & 4 & 8 & 13 & \infty \\
\end{array} \]

j
Execution of a call to Merge(A,p,q,r):

\[
\begin{array}{cccccccc}
A & \ldots & 1 & 4 & 6 & 7 & 8 & 12 & 13 & \boxed{13} & \ldots \\
\hline
p & q & q+1 & r \\
\end{array}
\]

\[
\begin{array}{cccccccc}
L & 6 & 7 & 12 & 24 & \infty \\
R & 1 & 4 & 8 & 13 & \infty \\
\end{array}
\]
Execution of a call to Merge(A,p,q,r):

We stop when A[p,r] has been filled
## Correctness of Merge

**Merge**($A, p, q, r$)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. for $i \leftarrow 1$ to $n_1$
   4. do $L[i] \leftarrow A[p + i - 1]$
5. for $j \leftarrow 1$ to $n_2$
   6. do $R[j] \leftarrow A[q + j]$
7. $L[n_1+1] \leftarrow \infty$
8. $R[n_2+1] \leftarrow \infty$
9. $i \leftarrow 1$
10. $j \leftarrow 1$
11. for $k \leftarrow p$ to $r$
   12. do if $L[i] \leq R[j]$
       13. then $A[k] \leftarrow L[i]$
       14. $i \leftarrow i + 1$
   15. else $A[k] \leftarrow R[j]$
   16. $j \leftarrow j + 1$

**Loop Invariant property (main for loop)**

- At the start of each iteration of the for loop, Subarray $A[p..k – 1]$ contains the $k – p$ smallest elements of $L$ and $R$ in sorted order.
- $L[i]$ and $R[j]$ are the smallest elements of $L$ and $R$ that have not been copied back into $A$.

### Initialization:

Before the first iteration:
- $A[p..k – 1]$ is empty.
- $i = j = 1$.
- $L[1]$ and $R[1]$ are the smallest elements of $L$ and $R$ not copied to $A$. 
Correctness of Merge

**Maintenance:**

**Case 1:** $L[i] \leq R[j]$
- By LI, $A$ contains $p - k$ smallest elements of $L$ and $R$ in sorted order.
- By LI, $L[i]$ and $R[j]$ are the smallest elements of $L$ and $R$ not yet copied into $A$.
- Line 13 results in $A$ containing $p - k + 1$ smallest elements (again in sorted order).
- Incrementing $i$ and $k$ reestablishes the LI for the next iteration.

**Case 2:** Similar arguments with $L[i] > R[j]$

**Termination:**
- On termination, $k = r + 1$.
- By LI, $A$ contains $r - p + 1$ smallest elements of $L$ and $R$ in sorted order.
- $L$ and $R$ together contain $r - p + 3$ elements including the two sentinels. So, all elements are sorted!
Analysis of Merge Sort

• Running time $T(n)$ of Merge Sort:
  • Divide: computing the middle takes $\Theta(1)$
  • Conquer: solving 2 sub-problems takes $2T(n/2)$
  • Combine: merging $n$ elements takes $\Theta(n)$
  • Total:
    \[
    T(n) = \begin{cases} 
    \Theta(1) & \text{if } n = 1 \\
    2T(n/2) + \Theta(n) & \text{if } n > 1
    \end{cases}
    \]

Solution: $T(n) = \Theta(n \log n)$

We will describe two ways to prove it. Though, to make our task easier, we will assume that $n$ is a power of 2.
Divide-and-conquer recurrence: proof by recursion tree

**Proposition.** If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2 T(n/2) + n & \text{otherwise} \end{cases}$$

**Pf 1.**

Divide-and-conquer recurrence: proof by recursion tree

\[ T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2 T(n/2) + n & \text{otherwise} \end{cases} \]

\[ n \quad = n \]
\[ 2 (n/2) \quad = n \]
\[ 4 (n/4) \quad = n \]
\[ 8 (n/8) \quad = n \]
\[ \vdots \]

\[ T(n) = n \log n \]
Proof by induction

**Proposition.** If \( T(n) \) satisfies the following recurrence, then \( T(n) = n \log_2 n \).

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2 \, T\left(\frac{n}{2}\right) + n & \text{otherwise}
\end{cases}
\]

**Pf 2.** [by induction on \( n \)]

- **Base case:** when \( n = 1 \), \( T(1) = 0 \).
- **Inductive hypothesis:** assume \( T(n) = n \log_2 n \).
- **Goal:** show that \( T(2n) = 2n \log_2 (2n) \).

\[
T(2n) = 2 \, T(n) + 2n
\]

\[
\log(a \cdot b) = \log(a) + \log(b)
\]

\[
\begin{align*}
&= 2 \, n \log_2 n + 2n \\
&= 2 \, n \left( \log_2 (2n) - 1 \right) + 2n \\
&= 2 \, n \log_2 (2n).
\end{align*}
\]

This is an example where the inductive hypothesis is used to prove \( T(2n) \) instead of \( T(n+1) \).
Arithmetic operations

Given 2 (binary) numbers, we want efficient algorithms to:

• Add 2 numbers

• **Multiply 2 numbers** (using divide-and-conquer!)
Integer addition

**Addition.** Given two $n$-bit integers $a$ and $b$, compute $a + b$.

**Subtraction.** Given two $n$-bit integers $a$ and $b$, compute $a - b$.

**Grade-school algorithm.** $\Theta(n)$ bit operations.

![Binary addition example](image)

**Remark.** Grade-school addition and subtraction algorithms are asymptotically optimal.
**Integer multiplication**

**Multiplication.** Given two $n$-bit integers $a$ and $b$, compute $a \times b$.

**Grade-school algorithm.** $\Theta(n^2)$ bit operations.

![Multiplication Diagram](image)

**Conjecture.** [Kolmogorov 1952] Grade-school algorithm is optimal.

**Theorem.** [Karatsuba 1960] Conjecture is wrong.
Divide-and-conquer multiplication

To multiply two \( n \)-bit integers \( x \) and \( y \):

- Divide \( x \) and \( y \) into low- and high-order bits.
- Multiply four \( \frac{n}{2} \)-bit integers, recursively.
- Add and shift to obtain result.

\[
m = \lfloor n / 2 \rfloor
\]
\[
a = \lfloor x / 2^m \rfloor \quad b = x \mod 2^m
\]
\[
c = \lfloor y / 2^m \rfloor \quad d = y \mod 2^m
\]

\[
(2^m a + b) (2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd
\]

Ex. \( x = 1 0 0 0 1 1 0 1 \) \( y = 1 1 1 0 0 0 0 1 \)

\[
\begin{array}{cccc}
a & b & c & d \\
1 & 2 & 3 & 4
\end{array}
\]
Divide-and-conquer multiplication

**MULTIPLY**$(x, y, n)$

**IF** $(n = 1)$

**RETURN** $x \times y$.

**ELSE**

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \mod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \mod 2^m$.

$e \leftarrow **MULTIPLY** (a, c, m)$.

$f \leftarrow **MULTIPLY** (b, d, m)$.

$g \leftarrow **MULTIPLY** (b, c, m)$.

$h \leftarrow **MULTIPLY** (a, d, m)$.

**RETURN** $2^{2m} e + 2^m (g + h) + f$. 
Divide-and-conquer multiplication analysis

**Proposition.** The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two $n$-bit integers.

**Pf.** Apply case 1 of the master theorem to the recurrence:

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^2)$$

Next class!
Karatsuba trick

To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

$$m = \lfloor n / 2 \rfloor$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \mod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \mod 2^m$$

$$(2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

$$= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd$$

Bottom line. Only three multiplication of $n/2$-bit integers.
Karatsuba multiplication

KARATSUBA-MULTIPLY($x$, $y$, $n$)

IF ($n = 1$)
    RETURN $x \times y$.
ELSE
    \[
    m \leftarrow \left\lfloor \frac{n}{2} \right\rfloor, \\
    a \leftarrow \left\lfloor \frac{x}{2^m} \right\rfloor; \quad b \leftarrow x \mod 2^m. \\
    c \leftarrow \left\lfloor \frac{y}{2^m} \right\rfloor; \quad d \leftarrow y \mod 2^m. \\
    e \leftarrow \text{KARATSUBA-MULTIPLY}(a, c, m). \\
    f \leftarrow \text{KARATSUBA-MULTIPLY}(b, d, m). \\
    g \leftarrow \text{KARATSUBA-MULTIPLY}(a - b, c - d, m). \\
    \text{RETURN } 2^{2m} e + 2^m (e + f - g) + f.
    \]
Karatsuba analysis

Proposition. Karatsuba's algorithm requires $O(n^{1.585})$ bit operations to multiply two $n$-bit integers.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = 3 \cdot T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_3 3}) = O(n^{1.585}).$$

Practice. Faster than grade-school algorithm for about 320-640 bits.

Next class!
Integer arithmetic reductions

**Integer multiplication.** Given two $n$-bit integers, compute their product.

<table>
<thead>
<tr>
<th>problem</th>
<th>arithmetic</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer multiplication</td>
<td>$a \times b$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer division</td>
<td>$a / b, a \mod b$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer square</td>
<td>$a^2$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer square root</td>
<td>$\sqrt[n]{a}$</td>
<td>$\Theta(M(n))$</td>
</tr>
</tbody>
</table>

Integer arithmetic problems with the same complexity as integer multiplication
## History of asymptotic complexity of integer multiplication

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba-Ofman</td>
<td>$\Theta(n^{1.585})$</td>
</tr>
<tr>
<td>1963</td>
<td>Toom-3, Toom-4</td>
<td>$\Theta(n^{1.465})$, $\Theta(n^{1.404})$</td>
</tr>
<tr>
<td>1966</td>
<td>Toom-Cook</td>
<td>$\Theta(n^{1+\varepsilon})$</td>
</tr>
<tr>
<td>1971</td>
<td>Schönhage–Strassen</td>
<td>$\Theta(n \log n \log \log n)$</td>
</tr>
<tr>
<td>2007</td>
<td>Fürer</td>
<td>$n \log n \ 2^{O(\log^*n)}$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>

**Remark.** GNU Multiple Precision Library uses one of five different algorithms depending on size of operands.

These results have even been improved since 2007...