**COMP251: Divide-and-Conquer (1)**

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Kleinberg & Tardos, 2005) & slides from (Snoeyink, 2004)

## 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.

### Sorting applications

- **Obvious applications.**
  - Organize an MP3 library.
  - Display Google PageRank results.
  - List RSS news items in reverse chronological order.

- **Some problems become easier once elements are sorted.**
  - Identify statistical outliers.
  - Binary search in a database.
  - Remove duplicates in a mailing list.

- **Non-obvious applications.**
  - Convex hull.
  - Closest pair of points.
  - Interval scheduling / interval partitioning.
  - Minimum spanning trees (Kruskal’s algorithm).
  - Scheduling to minimize maximum lateness or average completion time.
  - ...

### Merge Sort – Example

<table>
<thead>
<tr>
<th>Original Sequence</th>
<th>Sorted Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 26 32 6 43 15 9 1</td>
<td>1 6 9 15 18 26 32 43</td>
</tr>
<tr>
<td>18 26 32 6 43 15 9 1</td>
<td>1 6 9 15 18 26 32 43</td>
</tr>
<tr>
<td>18 26 32 6 43 15 9 1</td>
<td>1 6 9 15 18 26 32 43</td>
</tr>
<tr>
<td>18 26 32 6 43 15 9 1</td>
<td>1 6 9 15 18 26 32 43</td>
</tr>
</tbody>
</table>
Merge-Sort $(A, p, r)$

**INPUT:** a sequence of $n$ numbers stored in array $A$

**OUTPUT:** an ordered sequence of $n$ numbers

```
MergeSort(A, p, r) // sort A[p..r] by divide & conquer
1  if p < r
2     then q ← ⌊(p+r)/2⌋
3     MergeSort(A, p, q)
4     MergeSort(A, q+1, r)
5  Merge(A, p, q, r) // merges A[p..q] with A[q+1..r]
```

**Correctness of Merge**

```
Merge(A, p, q, r)
1  n1 ← q−p+1
2  n2 ← r−q−1
3  for i ← 1 to n1
4     do lij ← A[p+i−1]
5     for j ← 1 to n2
6     do lij ← A[q+j]
7     L[n1+1] ← ∞
8     R[n2+1] ← ∞
9     i ← 1
10    j ← 1
11  for k ← p to r
12     do if lij ≤ R[j]
13        then L[i] ← lij
14        else A[i] ← R[j]
15        i ← i + 1
16        j ← j + 1
```

**Analysis of Merge Sort**

- **Running time $T(n)$ of Merge Sort:**
  - Divide: computing the middle takes $\Theta(1)$
  - Conquer: solving 2 subproblems takes $2T(n/2)$
  - Combine: merging $n$ elements takes $\Theta(n)$

Total:

$T(n) = \Theta(1) \text{ if } n = 1$

$T(n) = 2T(n/2) + \Theta(n) \text{ if } n > 1$

$\Rightarrow T(n) = \Theta(n \log n)$
A useful recurrence relation

Definition. If \( T(n) \) is the maximum number of compares to merge sort a list of size \( n \).

Note. \( T(n) \) is monotone nondecreasing.

Merge sort recurrence.

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

Solution. \( T(n) \) is \( \Theta(n \log_2 n) \).

Assorted proofs. We describe several ways to prove this recurrence.

Initially we assume \( a \) is a power of 2 and replace \( a \) with \( n \).

Proof by induction

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

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

Proof. If \( T(a) \) is \( \Theta(\log n) \).

\[
T(2a) = 2T(a) + a
\]

Case 1: \( a \) is a power of 2.

\[
a = \log_2 n
\]

Divide-and-conquer recurrence: proof by recursion tree

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

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

Proof. \( T(2a) = T(a) + T(a) + a \).

\[
T(a) = 0, T(a) + T(a) + a = 2T(\frac{n}{2}) + a
\]

Analysis of merge sort recurrence

Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \log_2 n \).

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

Proof. [By strong induction on \( n \)]

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

- Base case: \( n = 1 \).
- Inductive hypothesis: assume \( T(n) = n \log_2 n \).
- Goal: show that \( T(n) = n \log_2 n \).

\[
T(2n) = 2T(n) \leq 2n \log_2 n = 2n \log_2 (2n) - 2n = 2n \log_2 (2n) - 2n
\]

Arithmetic operations

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

- Add 2 numbers
- Multiply 2 numbers (here, we will use a divide-and-conquer method!)

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.** \( (a_i) \) bit operations.

- 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.

```
(a)  1 2 3 4
(b)  5 6 7 8
```

```
1 2 3 4  
5 6 7 8  
10 12 14 17 18 20 24 26
```

Conjecture. [Zakosarenko 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 ½n-bit integers, recursively.
- Add and shift to obtain result.

```
a = \lfloor x/2 \rfloor
b = x \mod 2^n

(2^n a + b)(2^n c + d) = 2^{2n} ac + 2^n (bc + ad)
```

```
a = \lfloor x/2 \rfloor
b = x \mod 2^n

\Rightarrow

\begin{align*}
2^n a + b &= x \\
2^n c + d &= y \\
2^n ac + 2^n (bc + ad) + bd &= x \times y
\end{align*}
```

Divide-and-conquer multiplication analysis

**Proposition.** The divide-and-conquer multiplication algorithm requires $\Theta(n^\log_2 3)$ bit operations to multiply two n-bit integers.

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

```
T(n) = 4T(n/2) + \Theta(n)
```

```
\Rightarrow

T(n) = \Theta(n^\log_2 3)
```

Karatsuba trick

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

```
h = ad - ac \mod 2^n - (a - h)(c - d)
```

```
h = \lfloor x/2 \rfloor
a = \lfloor x/2 \rfloor
b = x \mod 2^n

\Rightarrow

\begin{align*}
2^n a + b &= x \\
2^n c + d &= y \\
2^n ac + 2^n (bc + ad) + bd &= x \times y
\end{align*}
```

Bottom line. Only three multiplication of $\times 2$-bit integers.

Karatsuba multiplication

```
a = \lfloor x/2 \rfloor
b = x \mod 2^n

\Rightarrow

\begin{align*}
2^n a + b &= x \\
2^n c + d &= y \\
2^n ac + 2^n (bc + ad) + bd &= x \times y
\end{align*}
```
Karatsuba analysis

Proposition. Karatsuba’s algorithm requires $O(n^{\log_2 3})$ bit operations to multiply two $n$-bit integers.

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

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

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

Integer arithmetic reductions

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer addition</td>
<td>$a + b$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Integer division</td>
<td>$a / b$, $a \mod b$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Integer square</td>
<td>$a^2$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Integer square root</td>
<td>$\sqrt{a}$</td>
<td>$O(n)$</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>7</td>
<td>Best case</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba-Ofman</td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>1963</td>
<td>Toom-3, Toom-4</td>
<td>$O(n^{1.71})$</td>
</tr>
<tr>
<td>1965</td>
<td>Toom-Cook</td>
<td>$O(n^{1.70})$</td>
</tr>
<tr>
<td>1971</td>
<td>Schönhage-Strassen</td>
<td>$O(n \log n \log \log n)$</td>
</tr>
<tr>
<td>2007</td>
<td>Fürer</td>
<td>$O(n \log^2 n)$</td>
</tr>
<tr>
<td>2007</td>
<td>GMP</td>
<td>$O(n \log(n \log n))$</td>
</tr>
</tbody>
</table>

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