Divide and Conquer

- MergeSort
  - We are given a list of n elements.
  - Split list into two lists each of length $n/2$.
    - More precisely, one with length: $\left\lceil \frac{n}{2} \right\rceil$ the other: $\left\lfloor \frac{n}{2} \right\rfloor$.
  - Sort each list using merge sort (recursive call).
  - Merge the sorted lists to get the final result.
- Merge sort analysis:
  - Let $T(n)$ = number of comparisons to sort n items in the worst case. Note: $\mathrm{T}(1) = 0$, $T(2) = 1$

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

    - Now what???

Floors and Ceilings

- In practice, floors and ceilings are often neglected.
- To be mathematically precise, we should not neglect them.
- This can nearly always be done, but details can get quite messy.
- In this course, we will almost always gloss over them. This is fine for our purposes, but in other settings it might not be (e.g. when writing a formal proof for publication).

Solving Recurrences

- Method 1: Recursion Trees
- Method 2: Master theorem
- Method 3: Guessing and Checking method

- We will study these in the next unit. For now, we solve the recurrence in an *ad-hoc* fashion.

Simple Merge Sort Analysis

- A simple analysis gives us a good guess:
  - Assume $n = 2^k$. Then:

$$
\begin{aligned}
T(n) &\le cn + 2T(n/2) \le cn + 2\{cn/2 + 2T(n/4)\} \\
&\le 2cn + 4T(n/4) = \ldots && \text{(continuing)} \\
&\le icn + 2^i\, T(n/2^i) && \text{(in general)} \\
&\le kcn + n\, T(1) && \text{(when } i \text{ is finally } k\text{)}
\end{aligned}
$$

➔ $T(n) \in O(n \log n)$ (since $k = \log n$).

A More Thorough Analysis

- Write it as

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + an & \text{if } n > 1 \end{cases}$$

- Prove $T(n) \le cn \log n$ by induction on n.
- Base case: $n = 1$. True since $T(n) = 0$.
- Inductive step: assume $T(k) \le ck \log k$ for $k < n$, and prove true for $k = n$.

Mergesort (cont.)

$$\begin{aligned}
T(n) &\le T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + an \\
&\le c\left\lfloor \frac{n}{2} \right\rfloor \log\left\lfloor \frac{n}{2} \right\rfloor + c\left\lceil \frac{n}{2} \right\rceil \log\left\lceil \frac{n}{2} \right\rceil + an \\
&\le c\left\lfloor \frac{n}{2} \right\rfloor \log\left(\frac{n}{2}\right) + c\left\lceil \frac{n}{2} \right\rceil \log n + an \\
&\le c\left\lfloor \frac{n}{2} \right\rfloor (\log n - 1) + c\left\lceil \frac{n}{2} \right\rceil \log n + an \\
&= cn \log n - c\left\lfloor \frac{n}{2} \right\rfloor + an \\
&\le cn \log n - c\left(\frac{n}{2} - \frac{1}{2}\right) + an = cn \log n + \left(a - \frac{c}{2}\right)n + \frac{c}{2} \\
&\le cn \log n \quad \text{for, say, } c = 4a
\end{aligned}$$

Divide-and-Conquer

- A general algorithmic paradigm (strategy)
  - <u>Divide</u>:
    - Separate a problem into subproblems.
  - <u>Conquer</u>:
    - Solve the subproblems (recursively).
  - <u>Combine</u>:
    - Use subproblem results to derive a final result for the original problem

    - Examples: binary search, quick sort, merge sort

When can we use D&C?

- Candidates for D&C:
  - Original problem is easily decomposable into subproblems.
    - We do not want to see “overlap” in the subproblems.
  - Combining solutions is not too costly.
  - Subproblems are about the same size.

  - We will see some examples of D&C with special emphasis on the analysis of execution time.

Multiplication of Large Integers

- We illustrate this problem with decimal digits.
  - – An actual implementation would use word-size chunks of bits in binary, but the idea is the same.
  - – Primitive operation: multiplying two digits (in practice: multiplying two words).
  - – Addition is usually less expensive so we neglect it now.
    - Later, we will need to convince ourselves that the cost of additions is not asymptotically dominant.

The "Grade School" Algorithm

				9	8	1
			1	2	3	4
			3	9	2	4
		2	9	4	3	
	1	9	6	2		
	9	8	1			
1	2	1	0	5	5	4

- If numbers have n digits, this takes $\Theta(n^2)$ time
- Can we do better?

1^{st} D&C Try at Long Multiplication

- Divide step:

  Split numbers in half, compute all products and combine with appropriate shifts. E.g.

$$0981 \times 1234 = (09 \cdot 10^2 + 81) \times (12 \cdot 10^2 + 34)$$
$$= 09 \times 12 \cdot 10^4 + 09 \times 34 \cdot 10^2 + 12 \times 81 \cdot 10^2 + 81 \times 34$$

- Conquer:
  - Recursively compute the products of numbers of size $n/2$.
- Combine:
  - If we ignore additions this is free.
  - More realistically: cost is $\Theta(n)$.

Analysis

- Assume *n* is a power of two.
  - Then all splits are exact.
- Recurrence is:

$$T(n) = 4T\left(n/2\right) + \Theta(n) \qquad T(1) = 1$$

  This has solution $T(n) \in \Theta(n^2)$.
  - You can verify this by substitution…
- So we have not done better than before! ☹
  - But we do not give up…
  - To improve our approach, we try to **reduce the number of subproblems**.

A Better D&C Algorithm

- As before, we split the first number into $w \cdot 10^{n/2} + x$, and the second number into $y \cdot 10^{n/2} + z$.
- In our first approach we computed **four** products wy, wz, xy, xz recursively.
- Improvement:
  - In the final sum both wz and xy appear with the same power of 10, i.e :

$$(w \cdot 10^{n/2} + x) \times (y \cdot 10^{n/2} + z) = wy \cdot 10^{n} + wz \cdot 10^{n/2} + xy \cdot 10^{n/2} + xz$$

  - so what we really need is: $(wz + xy) \cdot 10^{n/2}$… yet this still requires two multiplications…

A Better D&C Algorithm

- IDEA! Note that

1. $wz + xy = (w + x)(y + z) - wy - xz.$
2. Left side has 2 multiplies, right side has 3 ☹….

   but wy and xz need to be computed anyway so we have them for free! ☺

$$\boxed{(w \cdot 10^{n/2} + x) \times (y \cdot 10^{n/2} + z) = \mathbf{wy} \cdot 10^{n} + (wz + xy) \cdot 10^{n/2} + \mathbf{xz}}$$

- Now our recurrence looks like $T(n)=3T(n/2)+ \Theta(n).$
  - Note: Cost for addition in combine step is still $\Theta(n)$.

Analysis

- Recurrence is now:

$$T(n) = 3T\left(n/2\right) + \Theta(n) \qquad T(1) = 1$$

- We will eventually show that this has solution: $T(n) \in \Theta(n^{\log 3})$, that is: $T(n) \in \Theta(n^{1.585})$
  - Overhead makes this expensive for small n
  - For large n there are asymptotically better algorithms which are also more practical (though beyond the scope of this course).

What next?

- Taking stock of our progress so far:
  - We have studied various algorithms.
  - We have seen a few recursions:

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$T(n) = 4T\left(n/2\right) + \Theta(n) \qquad T(1) = 1$$

$$T(n) = 3T\left(n/2\right) + \Theta(n) \qquad T(1) = 1$$

  - As our algorithms get more complicated we will get into analysis work that is also more complicated.
    - It is reasonably clear that we are going to need a more systematic approach to solving recursions.
    - That is the topic of our next unit.