Divide-and-conquer

<table>
<thead>
<tr>
<th>Divide-and-Conquer (Input: Problem P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Solve P:</td>
</tr>
<tr>
<td>1. Divide P into smaller problems P₁, P₂, P₃,...,Pₖ.</td>
</tr>
<tr>
<td>2. Conquer by solving the (smaller) subproblems recursively.</td>
</tr>
<tr>
<td>3. Combine solutions to P₁, P₂,...,Pₖ into solution for P.</td>
</tr>
</tbody>
</table>

1 MergeSort

- Can we design better than \(n^2\) (quadratic) sorting algorithm?
- We will do so using one of the most powerful algorithm design techniques.
- Using divide-and-conquer, we can obtain a mergesort algorithm.
  - Divide: Divide \(n\) elements into two subsequences of \(n/2\) elements each.
  - Conquer: Sort the two subsequences recursively.
  - Combine: Merge the two sorted subsequences.
- Assume we have procedure Merge(A, p, q, r) which merges sorted A[p..q] with sorted A[q+1...r]
- We can sort A[p...r] as follows (initially p=1 and r=n):

  ```plaintext
  Merge Sort(A,p,r)
  If p < r then
    q = ⌊(p + r)/2⌋
    MergeSort(A,p,q)
    MergeSort(A,q+1,r)
    Merge(A,p,q,r)
  ```

- How does Merge(A, p, q, r) work?
  - Imagine merging two sorted piles of cards. The basic idea is to choose the smallest of the two top cards and put it into the output pile.
  - Running time: \((r - p)\)
Implementation is a bit messier..

Mergesort Example:

```
5 2 4 6 1 3 2 6
5 2 4 6 1 3 2 6
5 2 4 6 1 3 2 6
1 2 3 4 5 6 6
2 4 5 6 1 2 3 6
2 5 4 6 1 3 2 6
125 4 6 3 2 6
```

1.1 Mergesort Correctness

- Prove that Merge() is correct (what is the invariant?)
- Assuming that Merge is correct, prove that Mergesort() is correct.
  - Induction on $n$

1.2 Mergesort Analysis

- To simplify things, let us assume that $n$ is a power of 2, i.e $n = 2^k$ for some $k$.
- Running time of a recursive algorithm can be analyzed using a recurrence equation/relation. Each “divide” step yields two sub-problems of size $n/2$.

$$T(n) \leq c_1 + T(n/2) + T(n/2) + c_2 n \leq 2T(n/2) + (c_1 + c_2 n)$$

- Next class we will prove that $T(n) \leq cn \log_2 n$. Intuitively, we can see why the recurrence has solution $n \log_2 n$ by looking at the recursion tree: the total number of levels in the recursion tree is $\log_2 n + 1$ and each level costs linear time.
- Note: If $n \neq 2^k$ the recurrence gets more complicated, but the solution is the same. (We will often assume $n = 2^k$ to avoid complicated cases).
2 Matrix Multiplication

- Let $X$ and $Y$ be $n \times n$ matrices

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix}$$

- We want to compute $Z = X \cdot Y$

$$z_{ij} = \sum_{k=1}^{n} X_{ik} \cdot Y_{kj}$$

- Naive method uses $\Rightarrow n^2 \cdot n = \Theta(n^3)$ operations

- Divide-and-conquer solution:

$$Z = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} (A \cdot E + B \cdot G) & (A \cdot F + B \cdot H) \\ (C \cdot E + D \cdot G) & (C \cdot F + D \cdot H) \end{bmatrix}$$

- The above naturally leads to divide-and-conquer solution:
  - Divide $X$ and $Y$ into 8 sub-matrices $A, B, C,$ and $D$.
  - Do 8 matrix multiplications recursively.
  - Compute $Z$ by combining results (doing 4 matrix additions).
- Let's assume $n = 2^c$ for some constant $c$ and let $A, B, C$ and $D$ be $n/2 \times n/2$ matrices.
  - Running time of algorithm is $T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$
  - But we already discussed a (simpler/naive) $O(n^3)$ algorithm! Can we do better?

2.1 Strassen’s Algorithm

- Strassen observed the following:

$$Z = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} (S_1 + S_2 - S_4 + S_6) & (S_4 + S_5) \\ (S_6 + S_7) & (S_2 + S_3 + S_5 - S_7) \end{bmatrix}$$

where

$$S_1 = (B - D) \cdot (G + H)$$
$$S_2 = (A + D) \cdot (E + H)$$
$$S_3 = (A - C) \cdot (E + F)$$
$$S_4 = (A + B) \cdot H$$
$$S_5 = A \cdot (F - H)$$
$$S_6 = D \cdot (G - E)$$
$$S_7 = (C + D) \cdot E$$
− Let's test that $S_6 + S_7$ is really $C \cdot E + D \cdot G$

\[
S_6 + S_7 = D \cdot (G - E) + (C + D) \cdot E = DG - DE + CE + DE = DG + CE
\]

+ This leads to a divide-and-conquer algorithm with running time $T(n) = 7T(n/2) + \Theta(n^2)$
  − We only need to perform 7 multiplications recursively.
  − Division/Combination can still be performed in $\Theta(n^2)$ time.

+ Let's solve the recurrence using the iteration method

\[
T(n) = 7T(n/2) + n^2
\]

\[
= n^2 + 7T(\frac{n}{2^2}) + \left(\frac{n}{2}\right)^2
\]

\[
= n^2 + \left(\frac{7}{2^2}\right)n^2 + 7^2T(\frac{n}{2^3})
\]

\[
= n^2 + \left(\frac{7}{2^2}\right)n^2 + 7^2\left(\frac{n}{2^3}\right)^2 + 7^3T(\frac{n}{2^3})
\]

\[
= n^2 + \left(\frac{7}{2^2}\right)n^2 + \left(\frac{7}{2^2}\right)^2n^2 + \left(\frac{7}{2^2}\right)^3n^2 + \ldots + \left(\frac{7}{2^2}\right)^{\log n}n^2 + 7^\log n
\]

\[
= \sum_{i=0}^{\log n - 1} \left(\frac{7}{2^2}\right)^i n^2 + 7^\log n
\]

\[
= n^2 \cdot \Theta\left((\frac{7}{2^2})^{\log n - 1}\right) + 7^\log n
\]

\[
= n^2 \cdot \Theta\left(\frac{7^\log n}{n^2}\right) + 7^\log n
\]

\[
= n^2 \cdot \Theta\left(\frac{7^\log n}{n^2}\right) + 7^\log n
\]

\[
= \Theta\left(7^\log n\right)
\]

− Now we have the following:

\[
7^\log n = \frac{7^\log n}{\log 7}
\]

\[
= \left(7^\log n\right)(1/ \log 7)
\]

\[
= n(1/ \log 7)
\]

\[
= n\log 7
\]

− Or in general: $a^{\log_k n} = n^{\log_k a}$
So the solution is $T(n) = \Theta(n^{\log_7}) = \Theta(n^{2.817...})$

- Note:
  - We are 'hiding' a much bigger constant in $\Theta()$ than before.
  - Currently best known bound is $O(n^{2.376...})$ (another method).
  - Lower bound is (trivially) $\Omega(n^2)$.