# Divide & Conquer DPV Ch2, KT Ch5

Frank

September 24, 2021

## 1 Summary

- D&C recurrences (Tool for analyzing running time of DC algorithms)
- Karatsuba's algorithm (fast integer multiplication)
- Strassen's algorithms (fast matrix multiplication)

## 2 Examples of DC Algorithms

### 2.1 Merge Sort

- Sorts an array $A[1 \dots n]$

```
procedure MERGE SORT
    if (n = 1) then
        done
    else
        split A into A_1, A_2 each of size n/2
        recursively sort A_1, A_2
        merge sorted A_1, A_2
```

### 2.2 Binary Search

- Find $x$ in $A[1 \dots n]$ where $A$ is sorted

```
procedure BINARY SEARCH
    if (n = 1) then
        trivial
    else if (A[n/2] ≥ x) then
        search A[1 ... n/2]
    else
        search A[n/2 + 1 ... n]
```

# 3 Technique for Solving DC

- Solve instance of a DC problem of size $n$

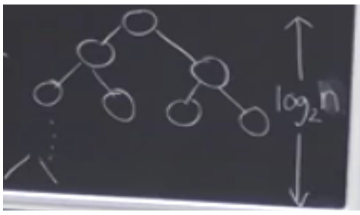
---

**procedure**<br>
    **if** ($n$ is 'small') **then**<br>
        solve directly<br>
    **else**<br>
        split input into $a$ pieces, each of size $\frac{n}{b}$ where ($a \geq 1, b > 1$ are constants)<br>
        recursively solve each of $a$ subproblems<br>
        combine solutions to $a$ subproblems into solution of original problem

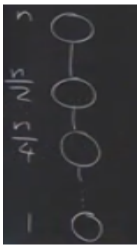
---

## 3.1 Visualizing DC

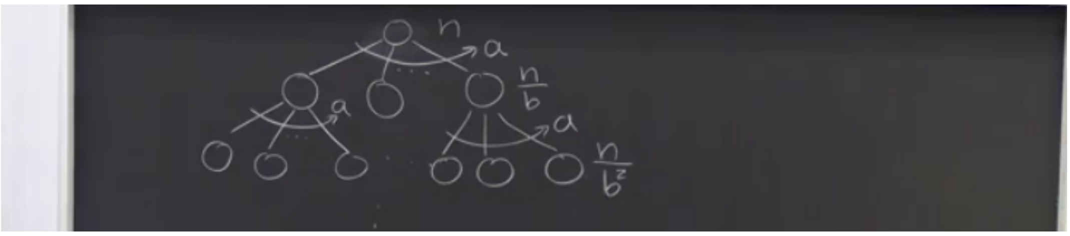
- Merge sort




- Binary search



- Generic




Our goal is to get to a case where the size of each piece is trivial

$$\frac{n}{b^k} = 1 \implies k = \log_b n$$

### 3.2 Is DC the same as Recursion?

No, not every recursive algorithm is a DC algorithm.

In a DC algorithm, we shrink the size of the input by a constant factor during each recursive call (not by a constant amount) $\implies$ guarantees that the height of the recursion tree is logarithmic (the number of problems is not logarithmic but is actually linear, $n$, so merging takes a factor of $n$

## 4 Running time of DC

### 4.1 Running time of DC (merge sort)

- Let $T(n)$ be the time for merge sort to sort an array of size $n$.

  The algorithm recursively calls 2 times to sort 2 arrays, each of size $\frac{n}{2}$ and merges using a time factor $cn$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn$$

  And let $T(1) = c$ for simplicity

  Solving this recurrance gives

$$T(n) = O(n \log n)$$

### 4.2 Running time of DC (binary search)

- Let $T(n)$ be the time for binary search to search an item in array of size $n$

  The algorithm solves a subproblem of size $\frac{n}{2}$ plus a constant amount of work for comparison, until we reach $T(1)$

$$T(n) = T\left(\frac{n}{2}\right) + c$$

  And let $T(1) = c$ for simplicity

  Solving this recurrance gives

$$T(n) = O(\log n)$$

### 4.3 Running time of generic DC algorithm

- Let $T(n)$ be the time required to solve an instance of the problem of size $n$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

then $a \cdot T\left(\frac{n}{b}\right)$ is the time to solve $a$ instances each of size $\frac{n}{b}$, and $c \cdot n^d$ is the merging time for each recursive call. Note that in binary search $d = 0$ and in merge sort $d = 1$

# 5 Master Theorem

The solution to DC recurrence is:

1. if $a < b^d$ then $T(n) = O(n^d)$
2. if $a = b^d$ then $T(n) = O(n^d \log n)$
3. if $a > b^d$ then $T(n) = O(n^{\log_b a})$

So once we write a DC algorithm, we just look at its $a, b, d$ values to determine its running time

### 5.1 Applying the Master Theorem (merge sort)

- $a = b = 2, d = 1 \implies a = b^d$

  $\implies T(n) = O(n \log n)$ by master theorem

### 5.2 Applying the Master Theorem (merge sort)

- $a = 1, b = 2, d = 0 \implies a = b^d$

  $\implies T(n) = O(\log n)$ by master theorem

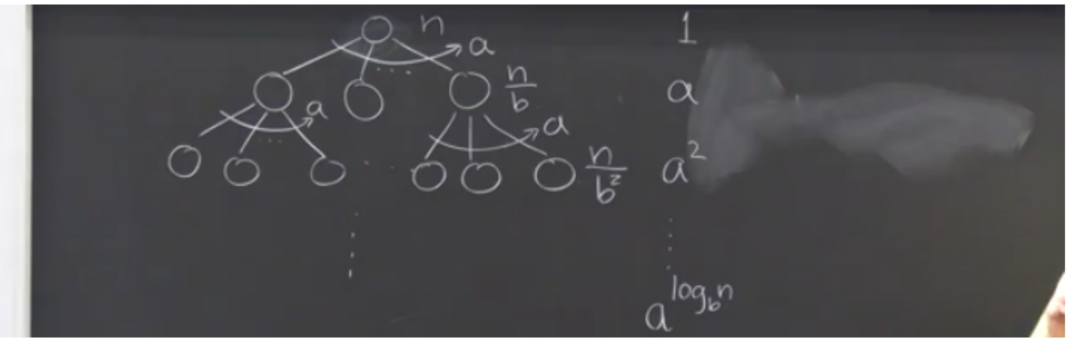
### 5.3 Proof

- Observations

  1. $n^d$ is amount of time it takes to do the dividing of the input and combining of the input. i.e. everything other than the recursive call (line 5 and 7)

     If $a < b^d$, then the amount of time the algorithm takes is dominated by the amount of time it takes to do the splitting and merging, so we have $n^d$

     If $a > b^d$, consider that the maximum depth of the recursion tree is $a^{\log_b n} = n^{\log_b a}$



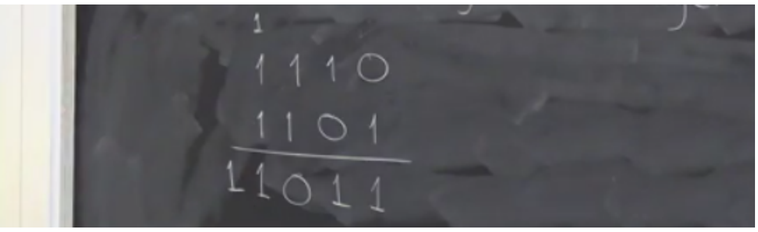

     which overwhelms the rest of the tree (the last level has almost everything) ,thus we have $O(n^{\log_b a})$, since the number of subproblems we have to solve dominates the splitting and combining process with respect to time

     If $a = b^d$, then we have a balance between the factors (depth of the tree vs. how many subproblems we have to solve)

- Rest on handout on website.

# 6 Karatsuba's Algorithm for Integer Multiplication

- Consider adding 2 binary numbers




  The running time is linear, because we are looking at every bit once and doing a constant amount of work for each.

  So $O(n)$ to add $n$-bits numbers, and similarily for subtraction. And we should not be able to improve upon that.

- Consider multiplying 2 binary numbers

  We have to cross multiply each bit, so $O(n^2)$, then perform $n-1$ additions of numbers that are $O(n)$ bits long, so also $O(n^2)$, so we end up with multiplication taking $O(n^2)$

  Can we do better?

## 6.1 DC idea 1

Given 2 $n$-bit numbers $X, Y$, and divide them into $\frac{n}{2}$ bits. Let $X_1$ be the high order bits for $X$ and $X_0$ be the low order bits for $X$, similarily for $Y$.


Then

$$X = X_1 \cdot 2^{\frac{n}{2}} + X_0$$
$$Y = Y_1 \cdot 2^{\frac{n}{2}} + Y_0$$

Where we fill the spots of $X_0$ with $0's$ in $X_1$, and similarily for $Y$

Then

$$X \cdot Y = X_1 Y_1 \cdot 2^n + (X_1 Y_0 + X_0 Y_1) \cdot 2^{\frac{n}{2}} + X_0 Y_0$$

Where we are left with

4 multiplications of $\frac{n}{2}$-bit numbers
+ 3 additions of $O(n)$-bit numbers
+ 2 $O(n)$-bit shifts

Such that both addition and shifts take linear time

- So the recurrence is

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn$$

Then $a = 4$, $b = 2$, $d = 1$, where $a > b^d$, case 3 of Master Theorem

$\implies T(n) = O(n^{\log_2 4}) = O(n^2)$, but this is no better than our original method

## 6.2 DC idea 2

The previous idea gives $O(n^2)$, which is no better

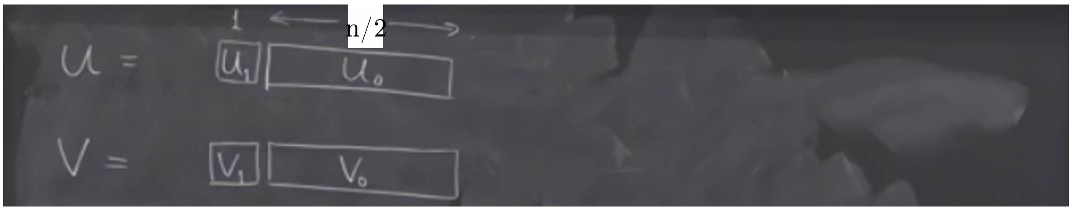
- But consider Karatsuba's Algorithm and do

$$X \cdot Y = X_1 Y_1 \cdot 2^n + [(X_1 + X_0) \cdot (Y_1 + Y_0) - X_1 Y_1 - X_0 Y_0] \cdot 2^{\frac{n}{2}} + X_0 Y_0$$

Thereby reducing the amount of multiplications, since we repeat $X_1 Y_1$ and $X_0 Y_0$. But, we increase the number of additions, which is linear work, so it doesn't matter.

- Then we are left with

3 multiplications of $\frac{n}{2}$-bit numbers (possibly $\frac{n}{2} + 1$-bit due to carry for the middle multiplication, but that doesn't matter, as follows)




Call $X_1 + X_0 = u$, and $Y_1 + Y_0 = v$, then $u$ is a single bit ($u_1$), followed by $n$ bits ($u_0$), and similarily with $v$ then

$$u = u_1 \cdot 2^{\frac{n}{2}} + u_0$$
$$v = v_1 \cdot 2^{\frac{n}{2}} + v_0$$

so

$$u \cdot v = u_1 v_1 \cdot 2^n + (u_1 v_0 + v_1 u_0) \cdot 2^{\frac{n}{2}} + u_0 v_0$$

where $u_1, v_1$ are single bits, so $u_1 v_1 \cdot 2^n$ is a $n$-shift, $u_1 v_0 \cdot 2^{\frac{n}{2}}$ and $v_1 u_0 \cdot 2^{\frac{n}{2}}$ are $\frac{n}{2}$-shifts, and finally $u_0 v_0$ is a $\frac{n}{2}$-bit multiplication

* So we really have a single $\frac{n}{2}$-bit multiplication followed by a linear amount of work.

+ 6 additions of $O(n)$-bit numbers
+ 2 $O(n)$-bit shifts

- So the recurrence is

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + cn$$

Then $a = 3$, $b = 2$, $d = 1$, where $a > b^d$, case 3 of Master Theorem

$\implies T(n) = O(n^{\log_2 3}) = O(n^{1.585\ldots})$, an improvement

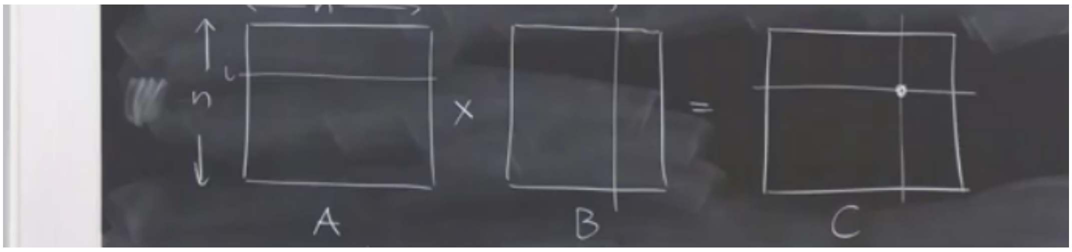
## 6.3 Difference between the two

- First algorithm is $O(n^2)$, and Karatsuba gives $O(n^{\log_2 3})$

  Karatsuba's actually does $\frac{1}{4}$ less work for every recursive step.

# 7 Strassen's Matrix Multiplication

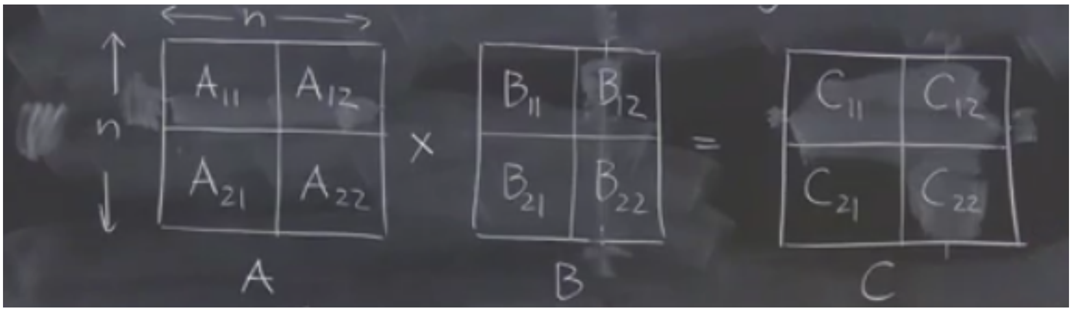
Multiply 2 square matrices, $A, B$ into $C$, by taking a dot product of every row and column




$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$

Every entry of $C$ requires $O(n)$ operations $\cdot$ $n^2$ entries $= O(n^3)$ running time for this basic algorithm

- Assume $O(1)$ time for $+$ and $*$
- Now apply DC, and divide every matrix into 4 sub matrices of equal size. $A_{11}, A_{12}, A_{21}, A_{22}$ and similarily for $B, C$




then

$$\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}$$

- We get recurrence,

$$T(n) = 8 \cdot T\left(\frac{n}{2}\right) + cn^2$$

Perform

  - 8 multiplications of $\frac{n}{2} \cdot \frac{n}{2}$ matrices
  - $c$ additions of $\frac{n}{2} \cdot \frac{n}{2} = O(n^2)$ entries

So $a = 8, b = 2, d = 2 \implies a > b^d \implies T(n) = O(n^{\log_2 8}) = O(n^3)$, again same as our basic algorithm

### 7.1 Improving our DC algorithm

Take

$$
\begin{aligned}
M_1 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
M_2 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_3 &= (A_{11} - A_{21})(B_{11} + B_{12}) \\
M_4 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_5 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_6 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_7 &= (A_{21} + A_{22}) \cdot B_{11}
\end{aligned}
$$

Then get

$$
\begin{aligned}
C_{11} &= M_1 + M_2 - M_4 + M_6 \\
C_{12} &= M_4 + M_5 \\
C_{21} &= M_6 + M_7 \\
C_{22} &= M_2 - M_3 + M_5 - M_7
\end{aligned}
$$

- Since we have 7 multiplications instead of 8, then

$$T(n) = O(n^{\log_2 7}) = O(n^{2.81\cdots})$$

- We have a lower bound of $O(n^2)$ for this operation, because we need $O(n^2)$ time to fill the $C$ matrix

# 8 Problem: TIFF Celebrity Problem

- Given $n$ patrons in a bar, one of them is a celebrity.

  A celebrity is defined as: all patrons know celebrity, but celebrity knows nobody

  The only thing we can do is go to patron $a$, and point to patron $b$ and ask, "do you know that person?" to get back a T/F answer. Proceed to another patron, perhaps the same person.

- Want to find the celebrity by asking the fewest possible patron.

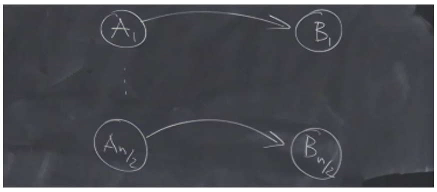
  By default, we can ask $n$ patrons $n$ times to get the answer in $O(n^2)$, but we want to do better.

### 8.1 Solution

- If the answer to the question is T, then we know that $\underline{a \text{ is not a celebrity}}$, since celebrity knows nobody.

- If the answer to the question is F, then we know that $\underline{b \text{ is not a celebrity}}$, since if $a$ is a celebrity, then $b$ cannot be a celebrity, but if $a$ is a patron, then $b$ also cannot be a celebrity because all patrons know celebrity.

- So divide the patrons into 2 groups, $A, B$. Ask patrons in group $A$ if they know a patron in $B$, and eliminate half of the patrons. with $\frac{n}{2}$ questions.




So we have

$$Q(n) = Q\left(\frac{n}{2}\right) + cn$$

eliminating half of the patrons in one recursive call and asking $\frac{n}{2}$ questions is in $O(n)$

- So $a = 1, b = 2, d = 1 \implies a < b^d$

$\therefore Q(n) = O(n)$