



Divide and Conquer Select, Multiplication

CS240

Spring 2023

Rui Fan



Selection

- Given list A of n numbers, and $i \leq n$, $\text{Select}(A, i)$ finds i 'th smallest number in A .
 - Ex $A = [3, 1, 6, 7, 2]$, $\text{Select}(A, 4) = 6$.
 - Assume numbers all distinct, for simplicity.
- Select generalizes median.
 - Median of A is $\text{Select}(A, n/2)$.
- We can solve select by first sorting the numbers in $O(n \log n)$ time, then choosing the i 'th largest one.
- We show how to solve Select, and hence median, in $O(n)$ time.



Select algorithm overview

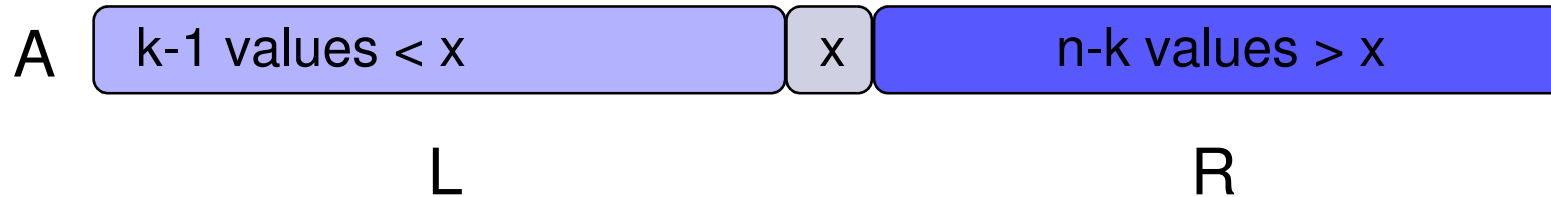
- Break A into many small groups.
- Find the median of each small group directly.
- Recursively find the median of this set of medians (using Select).
- Use the median-of-medians to partition A .
- Keep looking for target in one of the two partitions.



Select algorithm 1

- Say A has n items, and we want i 'th largest.
 - Assume for simplicity n divides 5.
- Divide A into $n/5$ groups of 5 elements each.
 - First 5 elements of A in first group, next 5 in second group, etc.
- For each group of 5 elements, find its median.
 - E.g. sort the 5 numbers, take 3rd value.
 - Let B be the set of $n/5$ medians.
- Recursively find the median in B .
 - I.e. do $\text{Select}(B, n/10)$.
 - Say the median is x .

Select algorithm 2



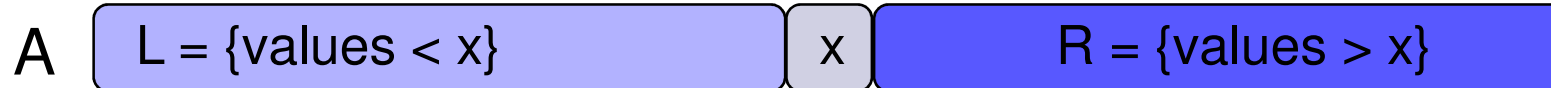
- Partition A using x as pivot.
 - Move all values $< x$ to the left of x , all other values to the right.
 - If A has n values, this takes $O(n)$ time.
- Say there are k values $\leq x$ in A , and $n - k$ values $> x$.
- If $i = k$, return x .
 - We want the i 'th largest item in A , which is x .
- If $i < k$, return $\text{Select}(L, i)$.
 - **Ex** If $i = 5$ and $k = 10$, then fifth largest item in A is fifth largest item in L .
- Else $i > k$, return $\text{Select}(R, i - k)$.
 - **Ex** if $i = 15$ and $k = 10$, then fifteenth largest item in A is fifth largest item in R .



Select analysis 1

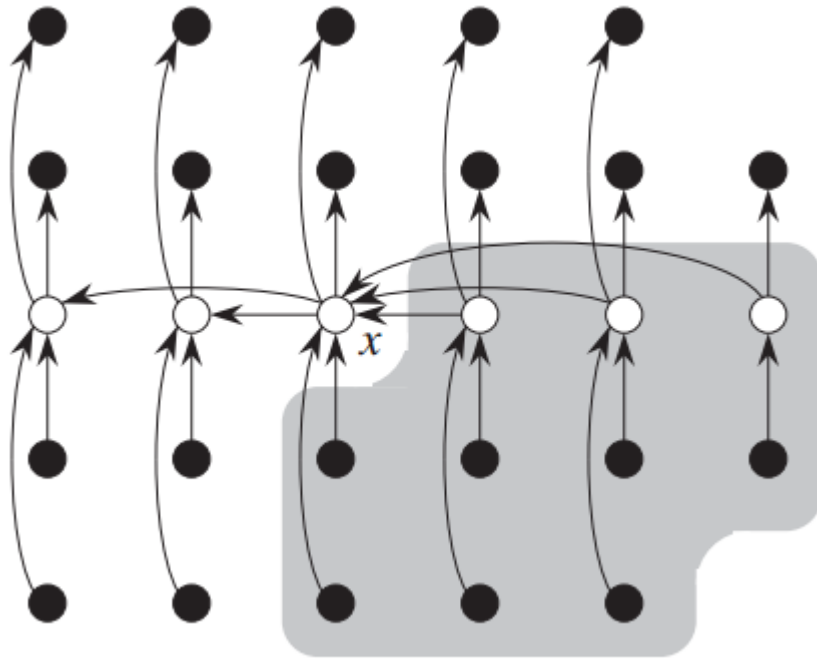
- Let $S(n)$ be an upper bound on select's running time given a list of size n .
- $S(n) = S(n/5) + S(u) + O(n)$
 - u is the size of whichever partition we recurse on.
 - $O(n)$ term comes from two parts.
 - First, for each of the $n/5$ groups of 5 numbers, find their median in constant time $\Rightarrow O(n)$ time overall.
 - Next, partition array in $O(n)$ time.
 - $S(n/5)$ to find the median of the $n/5$ group medians.
 - The key to ensuring select runs fast is ensuring u is small.
 - We will show $u \leq 7n/10$.

Select analysis 2



- Let x be the median of medians.
- We show that $|L| \geq 3n/10$, and $|R| \geq 3n/10$.
 - I.e. there are at least $3n/10$ values less than x , and $3n/10$ values larger than x .
- Then, also get $|L| \leq 7n/10$.
 - $|L| = n - 1 - |R|$.
 - Also, $|R| \leq 7n/10$.
- Thus, $S(n) \leq S(n/5) + S(7n/10) + O(n)$.

Select analysis 3



- Groups of 5 elements are shown in columns.
- Medians are shown in white.
- Medians less than x are shown to the right.

Source: *Introduction to Algorithms*
Cormen, Leiserson, Rivest, Stein

- There are $n/5 * 1/2 = n/10$ medians less than x .
- For each such median, there are two more values from the group less than the median.
 - There are 3 values from each group $< x$.
- So there are at least $3n/10$ values $< x$.
- I.e. $|L| \geq 3n/10$.
- Similarly, $|R| \geq 3n/10$.



Select analysis 4

- We have $S(n) \leq S(n/5) + S(7n/10) + O(n)$.
 - Since we want to upper bound $S(n)$, we let $S(n) = S(n/5) + S(7n/10) + O(n)$.
 - For sufficiently large n , the $O(n)$ term is bounded by bn , for some constant b .
- We guess $S(n) = O(n)$, and show this satisfies equation above, i.e. the guess is valid.
- Since $S(n) = O(n)$, $S(n) \leq cn$ for sufficiently large n , for some constant c .



Select analysis 5

- From $S(n) = S(n/5) + S(7n/10) + bn$ and $S(n) = cn$, we get
$$cn = c(n/5) + c * 7n/10 + bn.$$
- So $cn = 9cn/10 + bn$, and $c = 10b$.
- Thus, $S(n) \leq 10bn = O(n)$ for sufficiently large n , for some constant b .



Multiplying complex numbers

- Compute $(a + bi)(c + di) = x + yi$.
 - i is the imaginary number, i.e. $i^2 = -1$.
- $x = ac - bd, y = ad + bc$
 - 4 multiplications, 2 additions.
- Can we do better?
- Yes! From Gauss, we have $x = ac - bd, y = (a + b)(c + d) - ac - bd$.
 - 3 multiplications: $ac, bd, (a + b)(c + d)$.
 - 5 additions.
- Why does Gauss's method matter?
 - It's useful when addition faster than multiplication.
 - It can be used **recursively** to speed up integer multiplication.

Complexity of multiplication

- Long multiplication of two integers
 - Multiply each digit of one number by all digits of other number.
 - Shift, carry and sum as needed.
 - If each number has n digits, takes $O(n^2)$ time.
- Can we multiply faster than $O(n^2)$ time?
- Yes! Using Karatsuba's algorithm (1962).

$$\begin{array}{r} 5127 \\ \times 4265 \\ \hline 25635 \\ 307620 \\ 1025400 \\ 20508000 \\ \hline 21866655 \end{array}$$

Divide and conquer multiplication

$$a = \underbrace{1000}_{a_1} \underbrace{1101}_{a_0} \quad b = \underbrace{1110}_{b_1} \underbrace{0001}_{b_0}$$

$$a = 2^{n/2} \cdot a_1 + a_0$$

$$b = 2^{n/2} \cdot b_1 + b_0$$

$$ab = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 b_0 + a_0 b_1) + a_0 b_0$$

- To multiply binary numbers a and b , split their digits in half and cross multiply.
 - Ex $10_2 \times 11_2 = 1*1*2^2 + (1*1+0*1)*2^1 + 0*1*2^0 = 6$
- Divide and conquer multiplication does 4 multiplications of $n/2$ bit numbers.
- It turns out this is **not faster** than long multiplication.

Karatsuba multiplication

$$a = \underbrace{10001101}_{a_1} \underbrace{}_{a_0} \quad b = \underbrace{11100001}_{b_1} \underbrace{}_{b_0}$$

$$\begin{aligned} (a + bi)(c + di) &= x + yi \\ x &= ac - bd \\ y &= (a + b)(c + d) - ac - bd \end{aligned}$$

$$a = 2^{n/2} \cdot a_1 + a_0$$

$$b = 2^{n/2} \cdot b_1 + b_0$$

$$ab = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 b_0 + a_0 b_1) + a_0 b_0$$

$$= 2^n \cdot \underbrace{a_1 b_1}_{\text{blue}} + 2^{n/2} \cdot \underbrace{(a_1 + a_0)(b_1 + b_0)}_{\text{green}} - \underbrace{a_1 b_1}_{\text{blue}} - \underbrace{a_0 b_0}_{\text{red}} + \underbrace{a_0 b_0}_{\text{red}}$$

- Karatsuba multiplication rearranges the terms in divide and conquer multiplication.
- It does 3 multiplications of $n/2$ digit numbers instead of 4.
- Applied **recursively**, this makes all the difference!
- Notice the similarity between Karatsuba's and Gauss's method.

Karatsuba complexity

$$a = 2^{n/2} \cdot a_1 + a_0$$

$$b = 2^{n/2} \cdot b_1 + b_0$$

$$ab = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 b_0 + a_0 b_1) + a_0 b_0$$

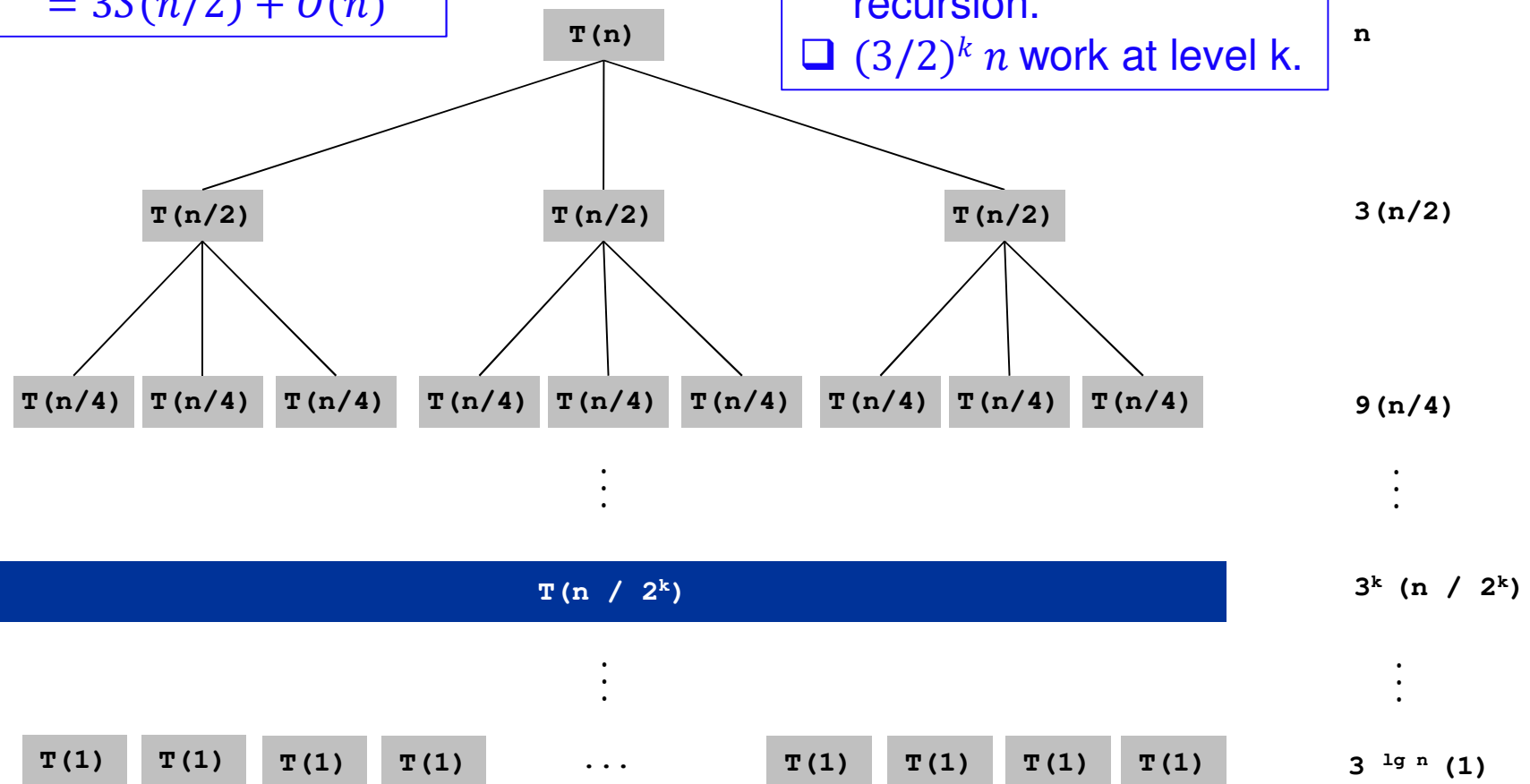
$$= 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0 + a_0 b_0$$

- $S(n)$ = time to multiply two n digit numbers using Karatsuba.
- 3 multiplications of $n/2$ digit numbers.
 - $a_0 b_0, a_1 b_1, (a_1 + a_0)(b_1 + b_0)$.
 - Takes $3S(n/2)$ time.
- Multiply by 2^n and $2^{n/2}$.
 - Done by shifting in $O(n)$ time.
 - Ex $2^4 \cdot 1011_2 = 10110000_2$.
- 5 additions / subtractions of $2n$ digit numbers.
 - $O(n)$ time.
- $S(n) = 3S(n/2) + O(n)$.
- $S(1) = 1$.

Karatsuba recursion tree

$$S(n) = 3S(n/2) + O(n)$$

- $\lg n = \log_2 n$ levels of recursion.
- $(3/2)^k n$ work at level k .



Source: <http://www.cs.bu.edu/fac/byers/courses/330/S13/handouts/05multiply.ppt>

Karatsuba complexity

- $S(n) = n + \left(\frac{3}{2}\right)n + \left(\frac{3}{2}\right)^2 n + \cdots + \left(\frac{3}{2}\right)^{\lg n} n$
- $S(n) = n \left(\frac{\left(\frac{3}{2}\right)^{1+\lg n} - 1}{\frac{3}{2} - 1} \right) = 2n \left(\frac{3n^{\lg 3}}{2 \cdot 2^{\lg n}} - 1 \right) = 3n^{\lg 3} - 2n$
 - $\lg 3 \approx 1.59$.
- So Karatsuba's method runs in $O(n^{1.59})$ instead of $O(n^2)$ time!
 - Practical for moderate $n > 100$.
- Useful for e.g. RSA cryptography.
- Even faster methods exist.
 - Schonhage-Strassen's Fast Fourier Transform based algorithm runs in $O(n \log n \log \log n)$ time.
 - Only practical for large n ($> 10,000$).

Complexity of matrix multiplication

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Source: <http://www.cs.bu.edu/fac/byers/courses/330/S13/handouts/05multiply.ppt>

```
for (i=0; i<n; i++) {  
    for (j=0; j<n; j++) {  
        C[i,j] = 0;  
        for (k=0; k<n; k++)  
            C[i,j] = C[i,j] + A[i,k]*B[k,j]
```

- ☐ Multiplying two $n \times n$ matrices the naive way takes $\Theta(n^3)$ time.
- ☐ Can we do better?
- ☐ Yes! Use Strassen's algorithm (1969).

Block matrix multiplication

$$\begin{array}{c} C_{11} \\ \swarrow \\ \left[\begin{array}{cc|cc} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ \hline 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{array} \right] \end{array} = \begin{array}{c} A_{11} \quad A_{12} \\ \swarrow \quad \swarrow \\ \left[\begin{array}{cc|cc} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ \hline 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{array} \right] \times \left[\begin{array}{cc|cc} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ \hline 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{array} \right] \begin{array}{c} B_{11} \quad B_{21} \\ \swarrow \quad \swarrow \end{array}
 \end{array}$$

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

- Break A, B and C into **blocks**. Multiply them as in normal matrix multiplication.
- Each block can be broken into subblocks and multiplied same way.
- Leads to recursive matrix multiplication method.

Block matrix multiplication

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

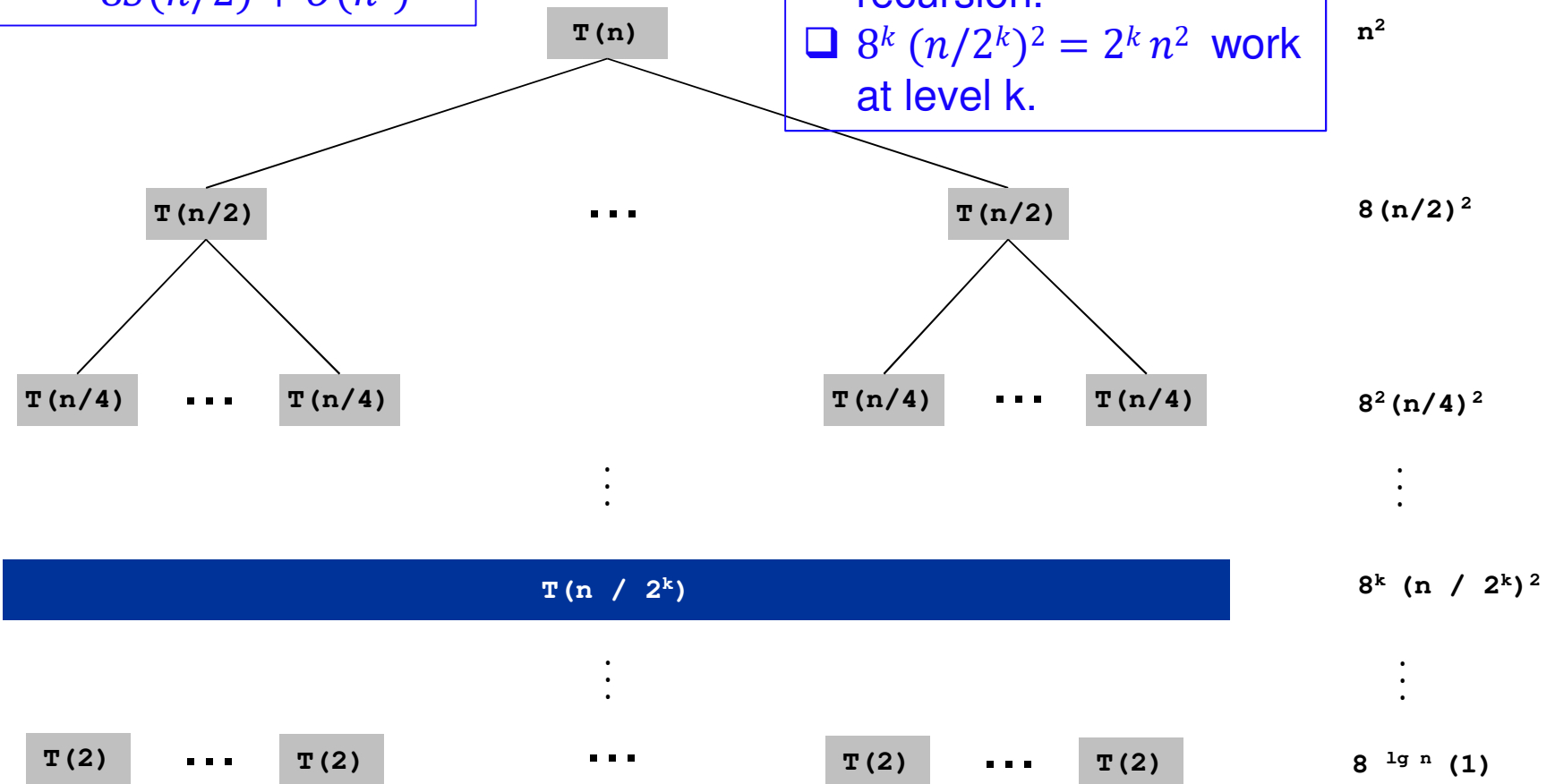
$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

- $S(n)$ = time for $n \times n$ block matrix multiplication.
- C_{11} requires two $(n/2) \times (n/2)$ block matrix multiplications $\Rightarrow 2S(n/2)$ time.
- C_{12}, C_{21} and C_{22} also require $2S(n/2)$ time each.
- We also add four pairs of $(n/2) \times (n/2)$ matrices.
 - Takes $O(n^2)$ time.
- So $S(n) = 8S(n/2) + O(n^2)$.
 - $S(2) = O(1)$.

Block MM recursion tree

$$S(n) = 8S(n/2) + O(n^2)$$

- $\lg n = \log_2 n$ levels of recursion.
- $8^k (n/2^k)^2 = 2^k n^2$ work at level k .



Block MM complexity

- $S(n) = n^2 + 2n^2 + 4n^2 + \dots + 2^{\lg n} n^2 = n^2 (1 + 2 + \dots + 2^{\lg n}) = n^2(2n - 1) = \Theta(n^3)$.
- So simple block matrix multiplication takes same time as naive matrix multiplication.
- Problem is each recursion does $8 (n/2) \times (n/2)$ MMs.
- Strassen's algorithm (1969) does $7 (n/2) \times (n/2)$ MMs in each recursion.
 - Complexity becomes $O(n^{\lg_2 7}) = O(n^{2.81})$.

Strassen's algorithm

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

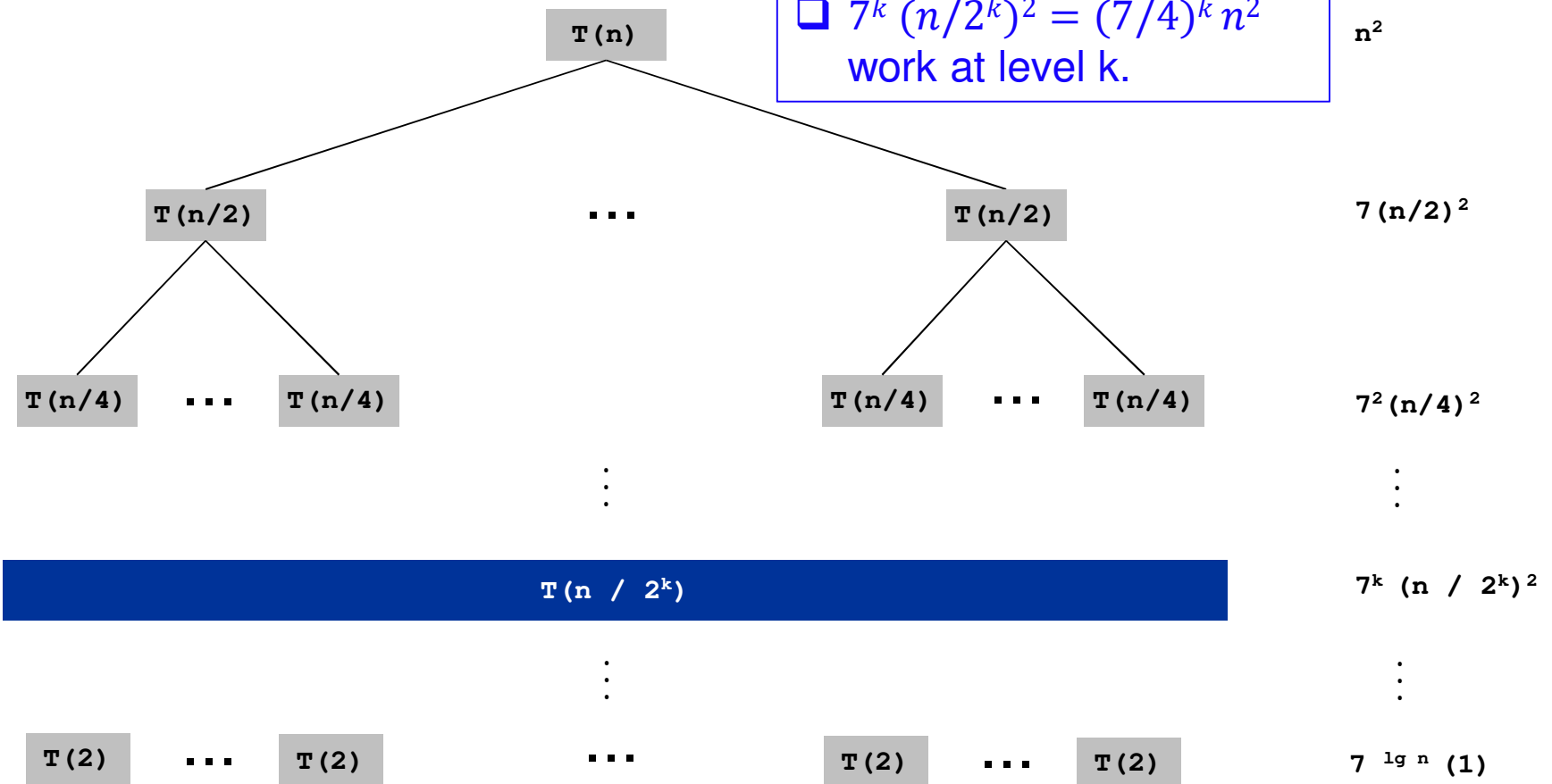
$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

- Strassen's algorithm does 7 multiplications and 18 additions / subtractions of $(n/2) \times (n/2)$ matrices.
- Let $S(n)$ be Strassen's algorithm's complexity.
- $S(n) = 7S(n/2) + O(n^2)$.
 - $O(n^2)$ comes from the matrix addition/subtractions.
- $S(2) = O(1)$.

Strassen recursion tree

$$S(n) = 7S(n/2) + O(n^2)$$

- $\lg n = \log_2 n$ levels of recursion.
- $7^k (n/2^k)^2 = (7/4)^k n^2$ work at level k.





Strassen's algorithm complexity

- $$\begin{aligned} S(n) &= n^2 + (7/4)n^2 + (7/4)^2n^2 + \dots + (7/4)^{\lg n}n^2 \\ &= n^2 (1 + (7/4) + \dots + (7/4)^{\lg n}) \\ &= n^2 ((7/4)^{\lg n+1} - 1) / (7/4 - 1) \\ &= n^2 O(7^{\lg n} / 4^{\lg n}) \\ &= n^2 O(n^{\lg 7} / n^2) \\ &= \Theta(n^{\lg 7}) \\ &= O(n^{2.81}) \end{aligned}$$



More about matrix multiplication

- Strassen's algorithm was a huge surprise. Before Strassen, it was widely believed matrix multiplication required $\Omega(n^3)$ time.
- Not clear how Strassen discovered the algorithm. Maybe inspiration from Karatsuba?
- Strassen's algorithm is practical. Beats naive method for $n > 20 - 1000$, depending on hardware architecture.
- Since Strassen, more sophisticated algorithms (but impractical) algorithms with $O(n^{2.375})$ time discovered.
- Some conjecture matrix multiplication can be done in $O(n^{2+\epsilon})$ time, for any $\epsilon > 0$.
 - This is nearly optimal, since even writing out $n \times n$ matrix output requires $O(n^2)$ time.