L8

FEB 16 2016

abhi shelat

FFT,Median

merge-sort (A, p, r)
    if $p < r$
        $q \leftarrow \lfloor (p+r)/2 \rfloor$
        merge-sort (A, p, q)
        merge-sort $(A, q+1, r)$
        merge(A, p, q, r)

```
MERGE(A[1..n], m):
  i ← 1; j ← m + 1
  for k ← 1 to n
      if j > n
          B[k] ← A[i]; i ← i + 1
      else if i > m
          B[k] ← A[j]; j ← j + 1
      else if A[i] < A[j]
          B[k] ← A[i]; i ← i + 1
      else
          B[k] ← A[j]; j ← j + 1
  for k ← 1 to n
      A[k] ← B[k]
```

jeff erickson

Karatsuba(ab, cd)

Base case: return b*d if inputs are 1-digit

ac = Karatsuba(a,c)

bd = Karatsuba(b,d)

t = Karatsuba((a+b),(c+d)) $\left(\frac{n}{2}+1\right)$

$3T(n/2) + 2n$

mid = t - ac - bd $\quad 4n$

RETURN ac*100^2 + mid*100 + bd $\quad 3n$

Closest(P,SX,SY)

Let q be the middle-element of SX

Divide P into Left, Right according to q

delta,r,j = MIN(Closest(Left, LX, LY) Closest(Right, RX, RY))

Mohawk = { Scan SY, add pts that are delta from q.x }

For each point x in Mohawk (in order):

Compute distance to its next 15 neighbors

Update delta,r,j if any pair (x,y) is < delta

Return (delta,r,j)

Can be reduced to 7!

```
arbit+(A[1...n])
 base case if |A|<=2, …
 (lg,minl,maxl) = arbit(left(A))
 (rg,minr,maxr) = arbit(right(A))
   return max{maxr-minl,lg,rg},
          min{minl, minr},
          max{maxl, maxr}
```

$$= \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$R = P_5 + P_4 - P_2 + P_6$ $\quad$ $S = P_1 + P_2$

$T = P_3 + P_4$ $\quad$ $U = P_5 + P_1 - P_3 - P_7$

[strassen]

$$P_1 = A(F - H)$$

$$P_2 = (A + B)H$$

$$P_3 = (C + D)E$$

$$P_4 = D(G - E)$$

$$P_5 = (A + D)(E + H)$$

$$P_6 = (B - D)(G + H)$$

$$P_7 = (A - C)(E + F)$$

FFT(f=a[1,...,n])

Base case if n<=2

E[...] <- FFT(A_e) // eval Ae on n/2 roots of unity
O[...] <- FFT(A_o) // eval Ao on n/2 roots of unity

combine results using equation:

$$A(\omega_{i,n}) = A_e(\omega_{i,n}^2) + \omega_{i,n} A_o(\omega_{i,n}^2)$$

$$A(\omega_{i,n}) = A_e(\omega_{i \bmod n/2, \frac{n}{2}}) + \omega_{i,n} A_o(\omega_{i \bmod n/2, \frac{n}{2}})$$

Return n resulting values.

Fast Fourier Transform 2

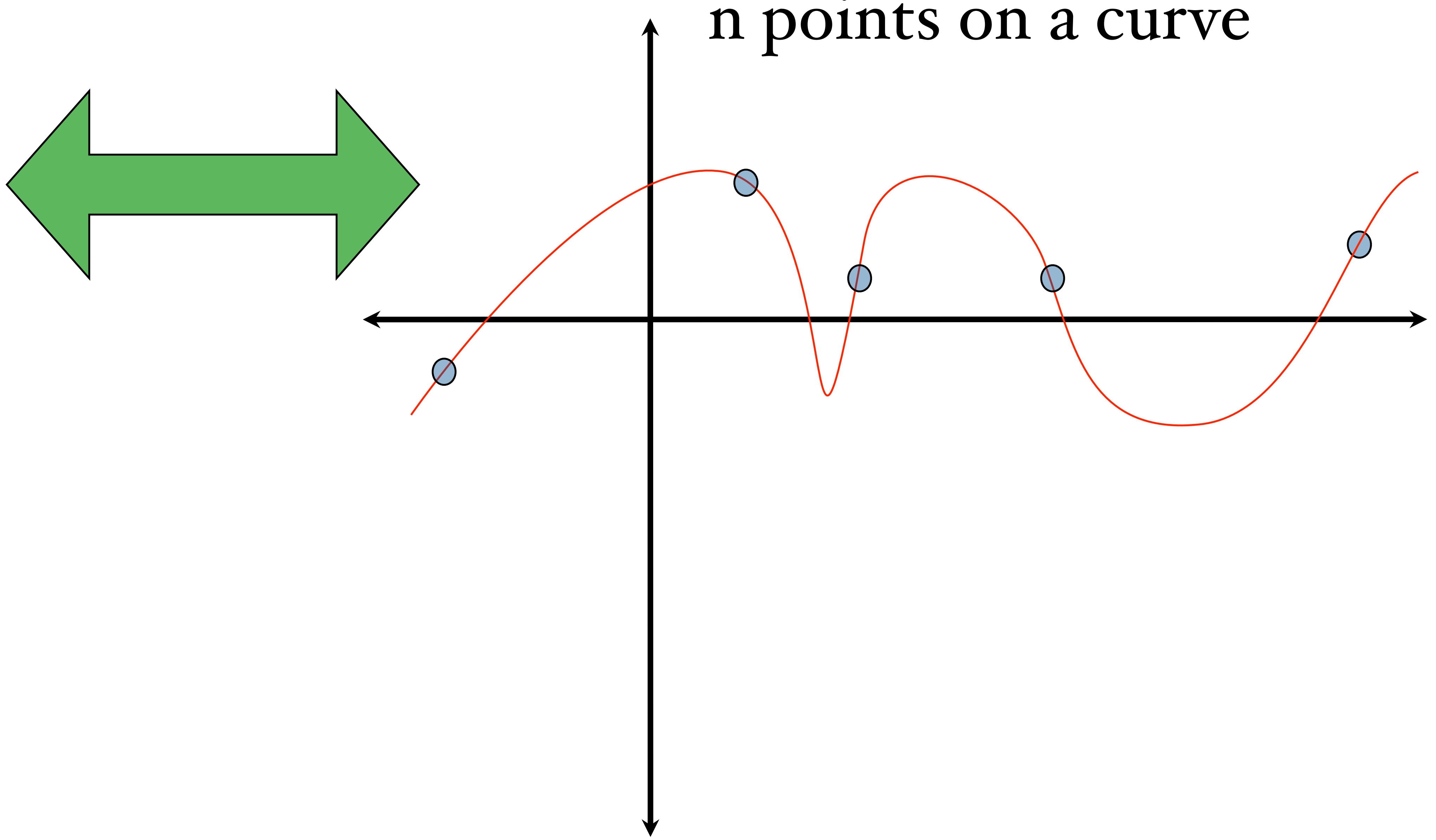
FFT

input: $a_0, a_1, a_2, \ldots, a_{n-1}$

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

output: evaluate polynomial A at (any) n different points. → n roots of unity

$A(x)$



$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

Brute force method to evaluate A at n points:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$
$$= a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-2} x^{n-2}$$
$$+ a_1 x + a_3 x^3 + a_5 x^5 + \cdots + a_{n-1} x^{n-1}$$

$$A_e(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_n x^{(n-2)/2}$$
$$A_o(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{(n-2)/2}$$

~degree $\frac{n}{2}$

deg n, ~deg n/2, ~deg n/2

$$A(x) = A_e(x^2) + x A_o(x^2)$$

2

Divide & Conquere

FFT(f=a[1,...,n])

Evaluates degree n poly on the n^{th} roots of unity

- $E \leftarrow FFT(A_e)$ // $E[1 \ldots n/2]$ $T(\frac{n}{2})$
- $O \leftarrow FFT(A_o)$ // $O[1 \ldots n/2]$ $T(\frac{n}{2})$

then compute

$\longrightarrow$ $A(x) = A_e(x^2) \pm x \cdot A_o(x^2)$ for n points $\leftarrow \Theta(n)$

$T(n) = 2T(\frac{n}{2}) + \Theta(n)$

Last remaining issue: Which points to use?

Need points that have log(n) square roots

Roots of unity (Complex)

$$x^n = 1$$

should have n solutions

what are they?

$\mathbb{Z}_p^*$

$$x^n = 1$$

the n solutions are:

$$\underbrace{\left\{1, e^{2\pi i/n}, e^{2\pi i 2/n}, e^{2\pi i 3/n}, \ldots, e^{2\pi i(n-1)/n}\right\}}$$

because $\underbrace{e^{2\pi i} = 1}$ Euler identity

$$\left[e^{2\pi i(j/n)}\right]^n = \left(e^{2\pi i}\right)^j = 1^j = 1$$

$$x^n = 1$$

the n solutions are:

consider $e^{2\pi ij/n}$ for j=0,1,2,3,...,n-1

$$\left[e^{(2\pi i/n)j}\right]^n = \left[e^{(2\pi i/n)n}\right]^j = \left[e^{2\pi i}\right]^j = 1^j$$

$e^{2\pi ij/n}$ = $\omega_{j,n}$ is an n[th] root of unity

$$\omega_{0,n}, \omega_{2,n}, \ldots, \omega_{n-1,n}$$

What is this number?

$e^{2\pi ij/n}$ = $\omega_{j,n}$ is an n^{th} root of unity

Taylor series expansion

of a function f around point a

$$f(y) = f(a) + \frac{f'(a)}{1!}(y-a) + \frac{f''(a)}{2!}(y-a)^2 + \frac{f'''(a)}{3!}(y-a)^2 +$$

$$e^x =$$

around 0

What is this number?

$e^{2\pi ij/n}$ = $\omega_{j,n}$ is an n[th] root of unity

$$\underline{e^{ix}} = \underline{\underline{\cos(x) + i\sin(x)}}$$

$$\underline{\underline{e^{2\pi ij/n}}} = \underline{\cos}(\underline{2\pi j/n}) + i\,\underline{\sin}(\underline{2\pi j/n})$$

$e^{2\pi i j/n} = \omega_{j,n}$ is an n^{th} root of unity

$$\omega_{0,n}, \omega_{2,n}, \dots, \omega_{n-1,n}$$

Lets compute $\omega_{1,8}$

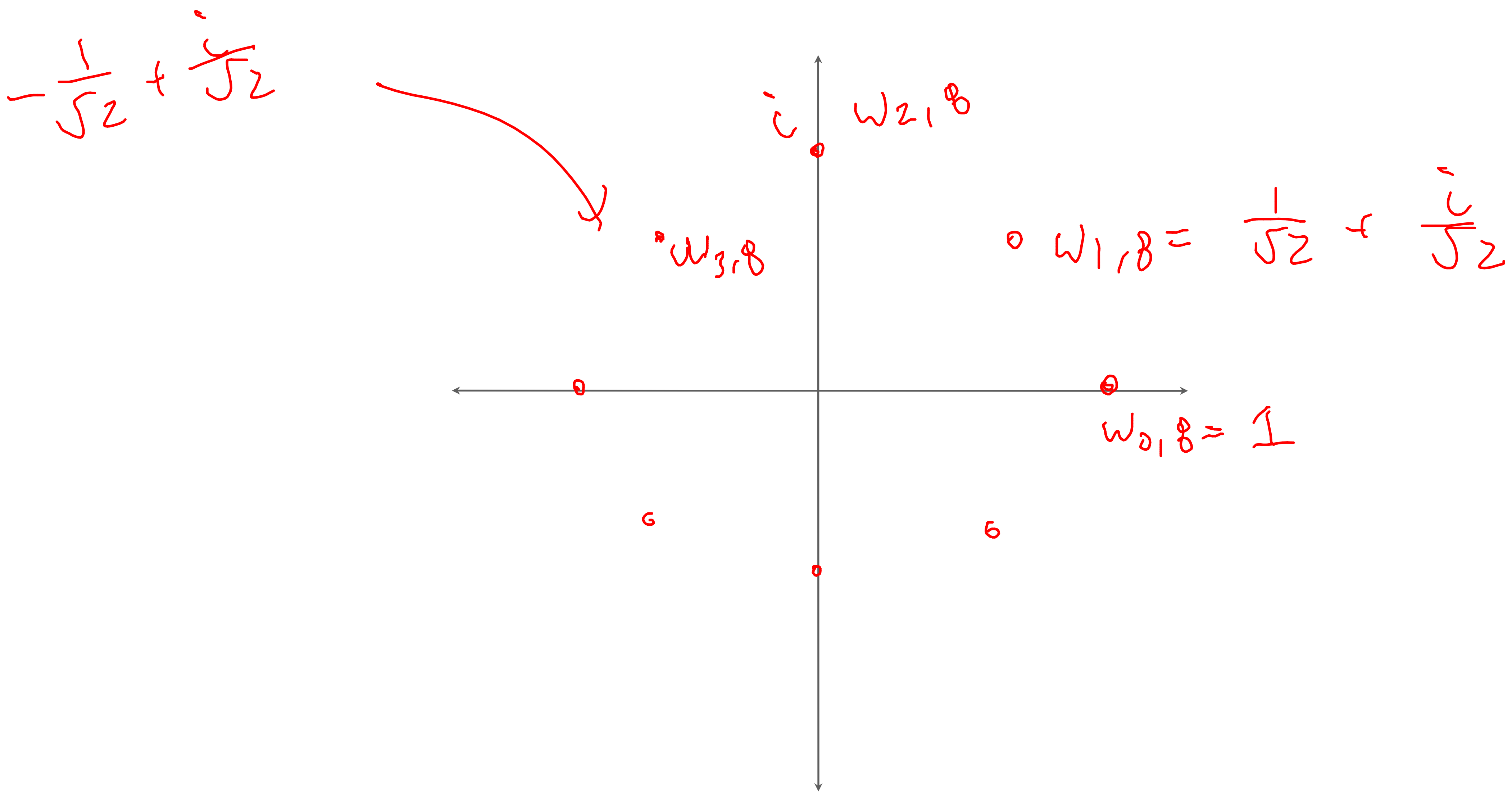
$$\omega_{1,8} = \cos(2\pi \, 1/8) + i \sin(2\pi \, 1/8)$$

$$= \cos(\pi/4) + i \sin(\pi/4)$$

$$= \frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}$$

Compute all 8 roots of unity

$n = 8$



Then graph them

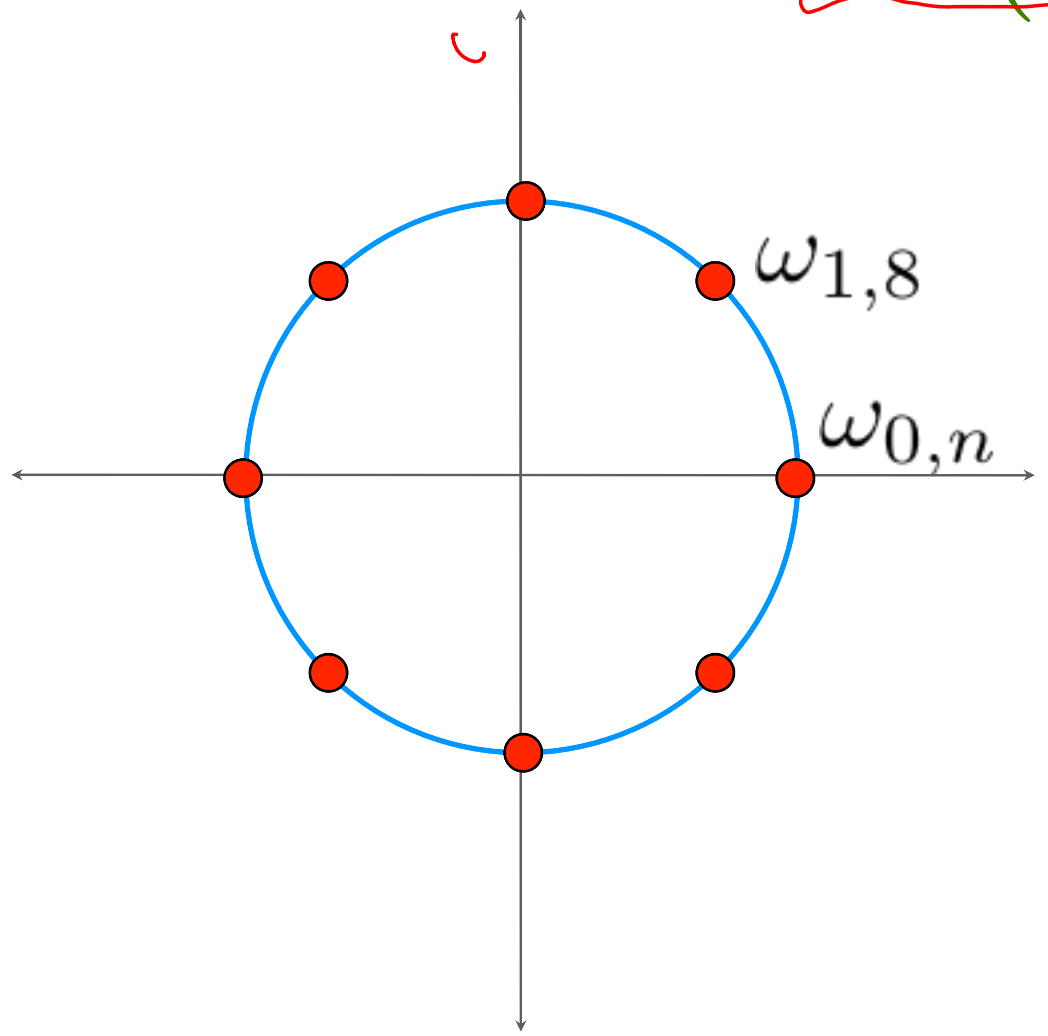
roots of unity

$$x^n = 1$$

should have n solutions

$$e^{2\pi ij/n} = \underbrace{\cos(2\pi j/n)} + i\,\underbrace{\sin(2\pi j/n)}$$

i

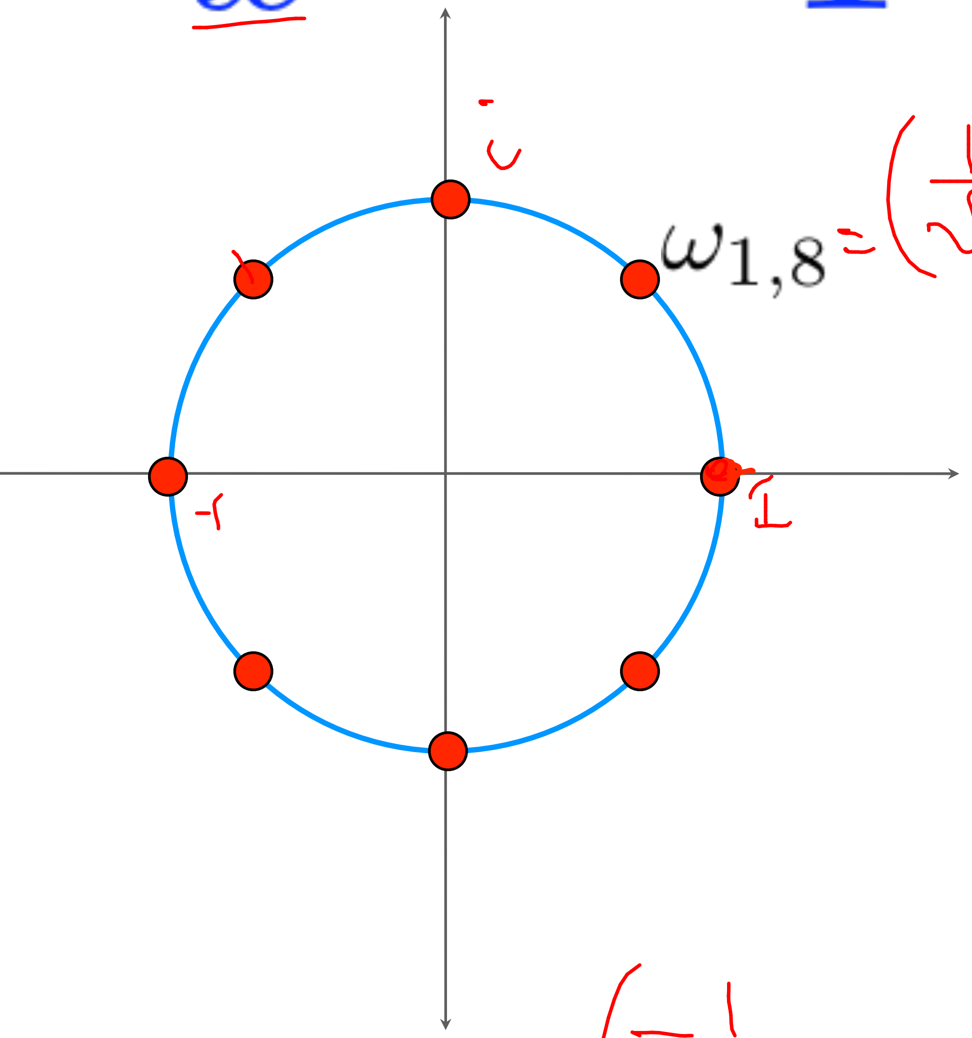


$\frac{1}{\sqrt{2}}$

squaring the n^{th} roots of unity

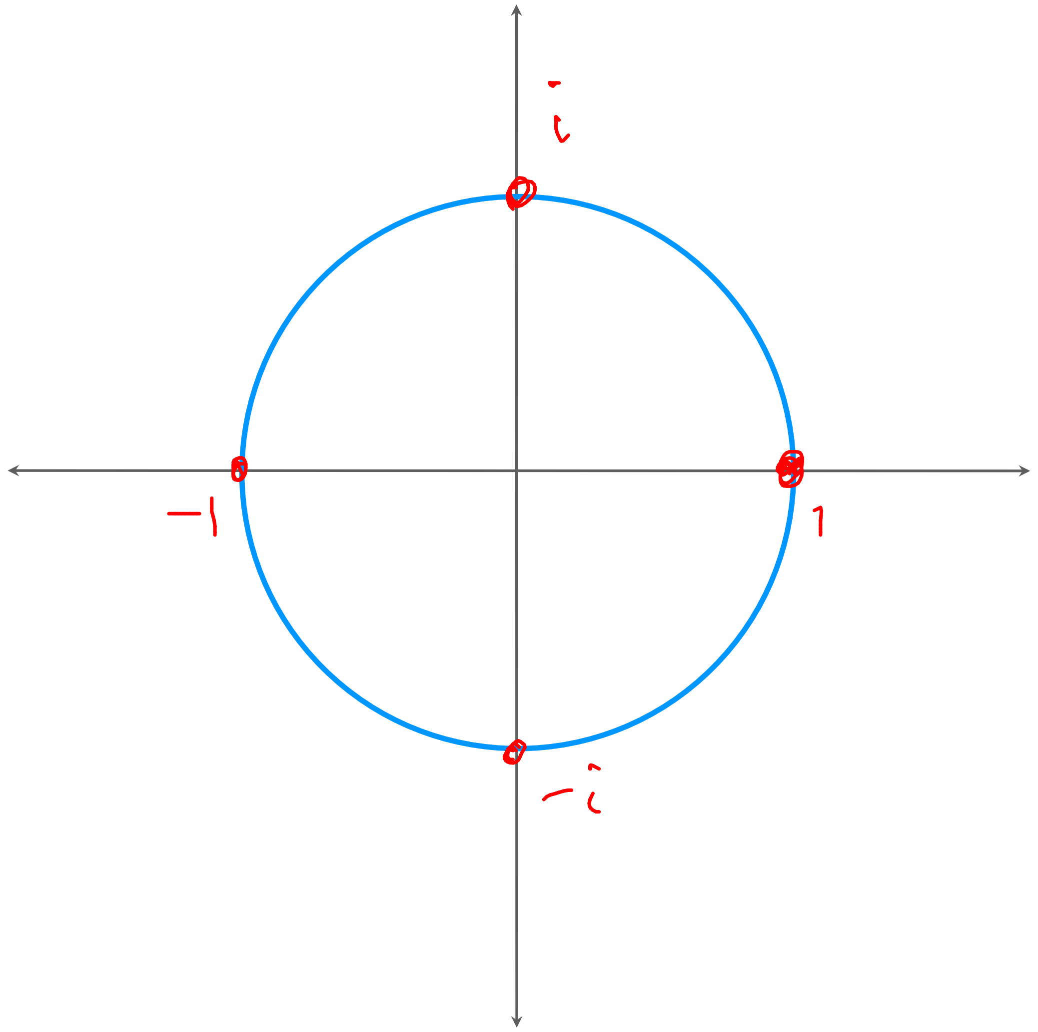
$$x^n = 1$$

$(n/2)^{th}$ roots of unity



$$\omega_{1,8} = \left(\frac{1}{\sqrt{2}} + \frac{i}{2}\right)^2 = i$$

$$\omega_{3,8} = \left(\frac{-1}{\sqrt{2}} + \frac{i}{2}\right)^2 = -i$$



Thm: Squaring an nth root produces an n/2th root.

example: $$\omega_{1,8} = \left(\frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}\right)$$

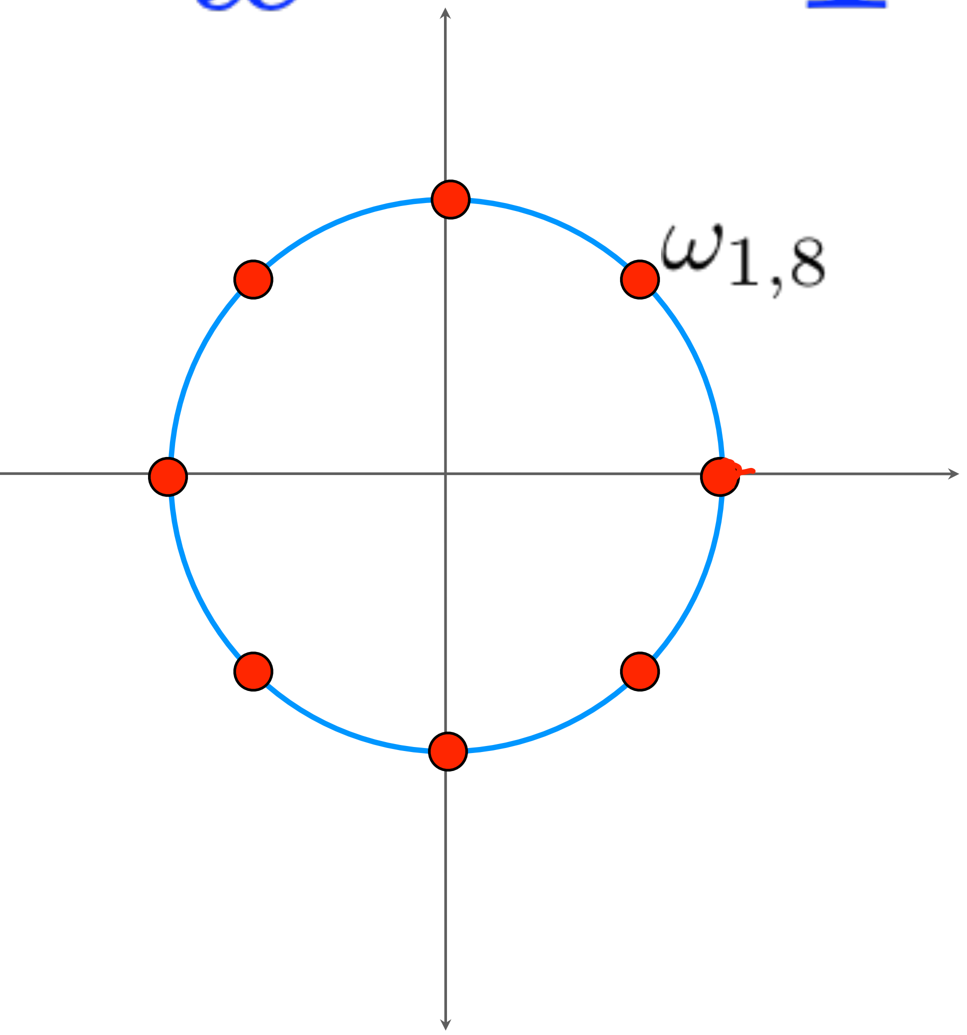
$$\omega_{1,8}^2 = \left(\frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}\right)^2 = \left(\frac{1}{\sqrt{2}}\right)^2 + 2\left(\frac{1}{\sqrt{2}}\frac{i}{\sqrt{2}}\right) + \left(\frac{i}{\sqrt{2}}\right)^2$$

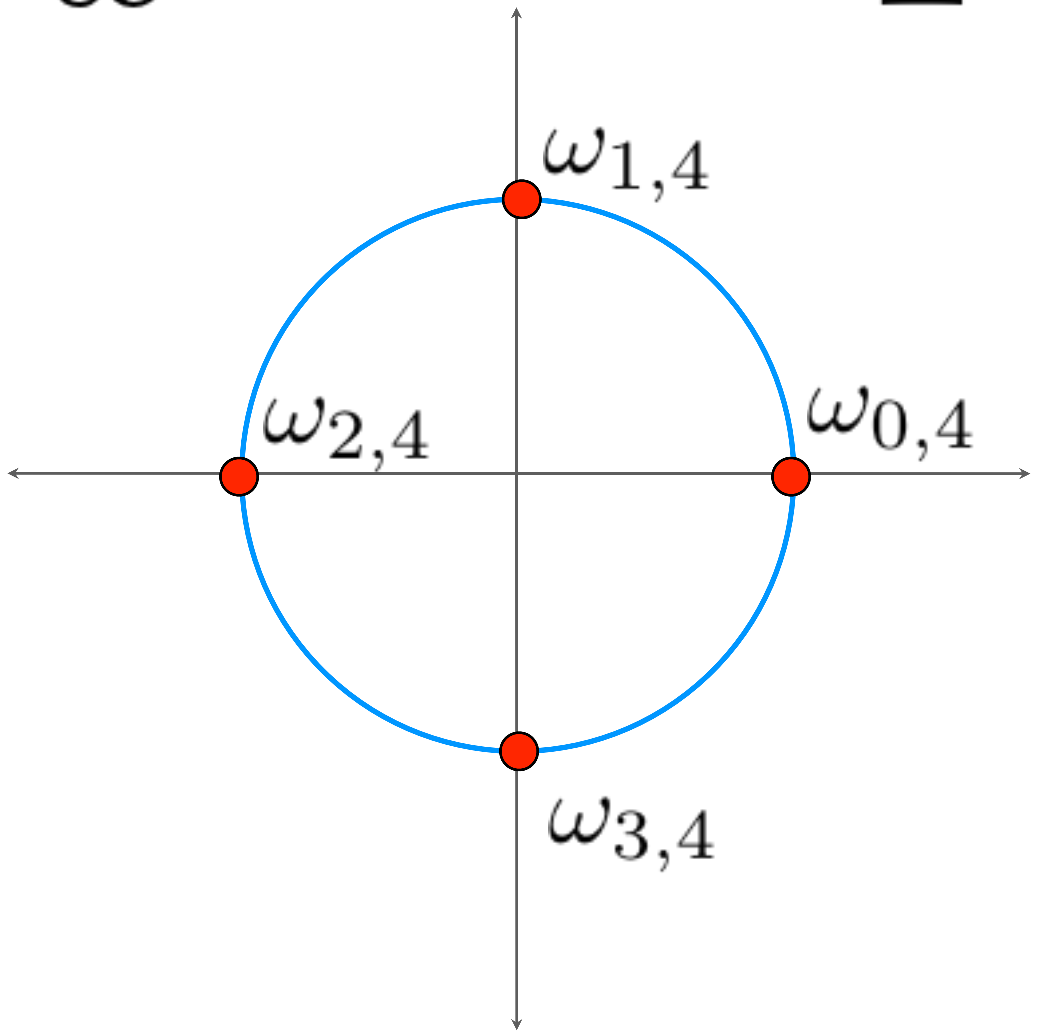
$$= 1/2 + i - 1/2$$

$$= i$$

squaring the n[th] roots of unity

$$x^n = 1 \qquad\qquad x^{n/2} = 1$$





Thm: Squaring an n^{th} root produces an $n/2^{th}$ root.

$$\left\{1, e^{2\pi i(1/n)}, e^{2\pi i(2/n)}, e^{2\pi i(3/n)}, \dots, e^{2\pi i(n/2)/n}, e^{2\pi i(n/2+1)/n}, \dots, e^{2\pi i(n-1)/n}\right\}$$

$\xrightarrow{n^{th} \text{ roots}}$

Square them

$e^{2\pi i\, 2/n}$

$$1, \quad e^{2\pi i \frac{1}{n/2}}, \quad e^{2\pi i \frac{2}{n/2}}, \quad e^{2\pi i}, \quad e^{2\pi i (n+2)/n}$$

$$= e^{2\pi i} \cdot e^{2\pi i \left(\frac{1}{n/2}\right)}$$

1

$\frac{1}{n/2} \to \left(\frac{n}{2}\right)^{th}$ root of unity

Thm: Squaring an nth root produces an n/2th root.

$$\left\{1, e^{2\pi i(1/n)}, e^{2\pi i(2/n)}, e^{2\pi i(3/n)}, \dots, e^{2\pi i(n/2)/n}, e^{2\pi i(n/2+1)/n}, \dots, e^{2\pi i(n-1)/n}\right\}$$

$1 \quad e^{2\pi i(1/(n/2))} \quad e^{2\pi i(2/(n/2))} \quad e^{2\pi i(3/(n/2))} \qquad 1$

$$e^{2\pi i((n/2)+1/(n/2))}$$

$$= e^{2\pi i(1+1/(n/2))}$$

$$= 1 \cdot e^{2\pi i(1/(n/2))}$$

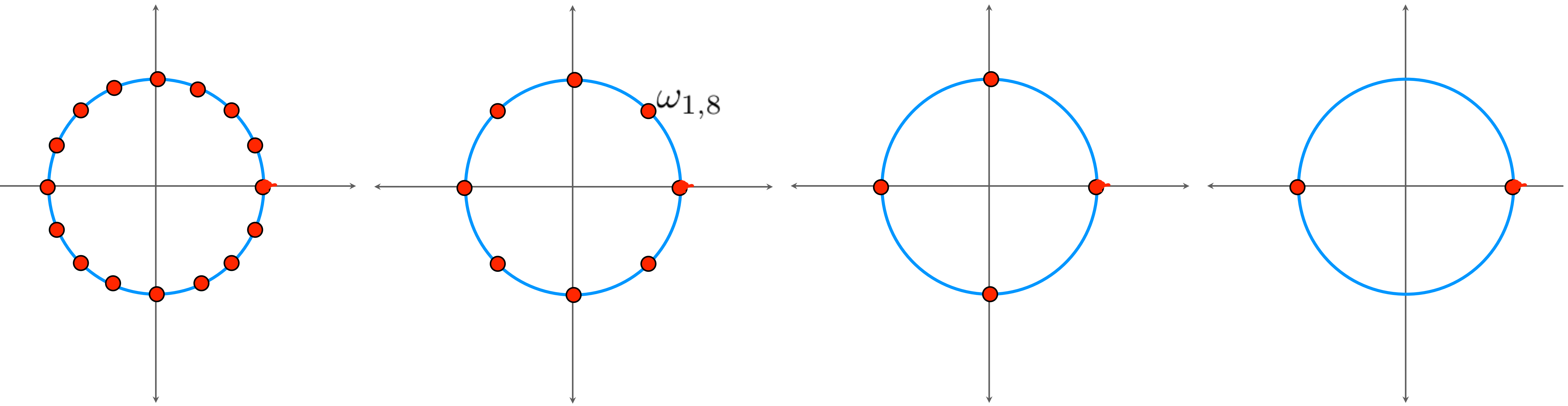
If n=16

$n = 8$

A_e, A_o

A_{eo}, A_{ee}

A_{oe}, A_{oo}



A @ the 16th roots of unity

$\frac{n}{2}$ = 8th roots of unity

4th roots of unity

2nd roots of unity

(Base case)

A_e, A_o

$$A(x) = A_e(x^2) + xA_o(x^2)$$

evaluate at a root of unity

$$A(\omega_{i,n}) = A_e(\omega_{i,n}^2) + \omega_{i,n} A_o(\omega_{i,n}^2)$$

nth root of unity

n/2th root of unity

n/2th root of unity

recursive fft call

FFT(f=a[1,...,n])

Evaluates degree n poly on the n^{th} roots of unity

$E \leftarrow FFT(A_e)$ //eval A_e of degree $\frac{n}{2}$ on the $\frac{n}{2}^{th}$ r.o-unity

$O \leftarrow FFT(A_o)$ // "

Combine these points to produce A eval @ n^{th} roots

$A(w_{0,n}) \ldots A(w_{n-1,n})$ using the equation

$$A(w_{i,n}) = \underline{A_e(w_{i,n}^2)} + w_{i,n} \cdot A_o(w_{i,n}^2)$$

FFT(f=a[1,...,n])

Base case if n<=2

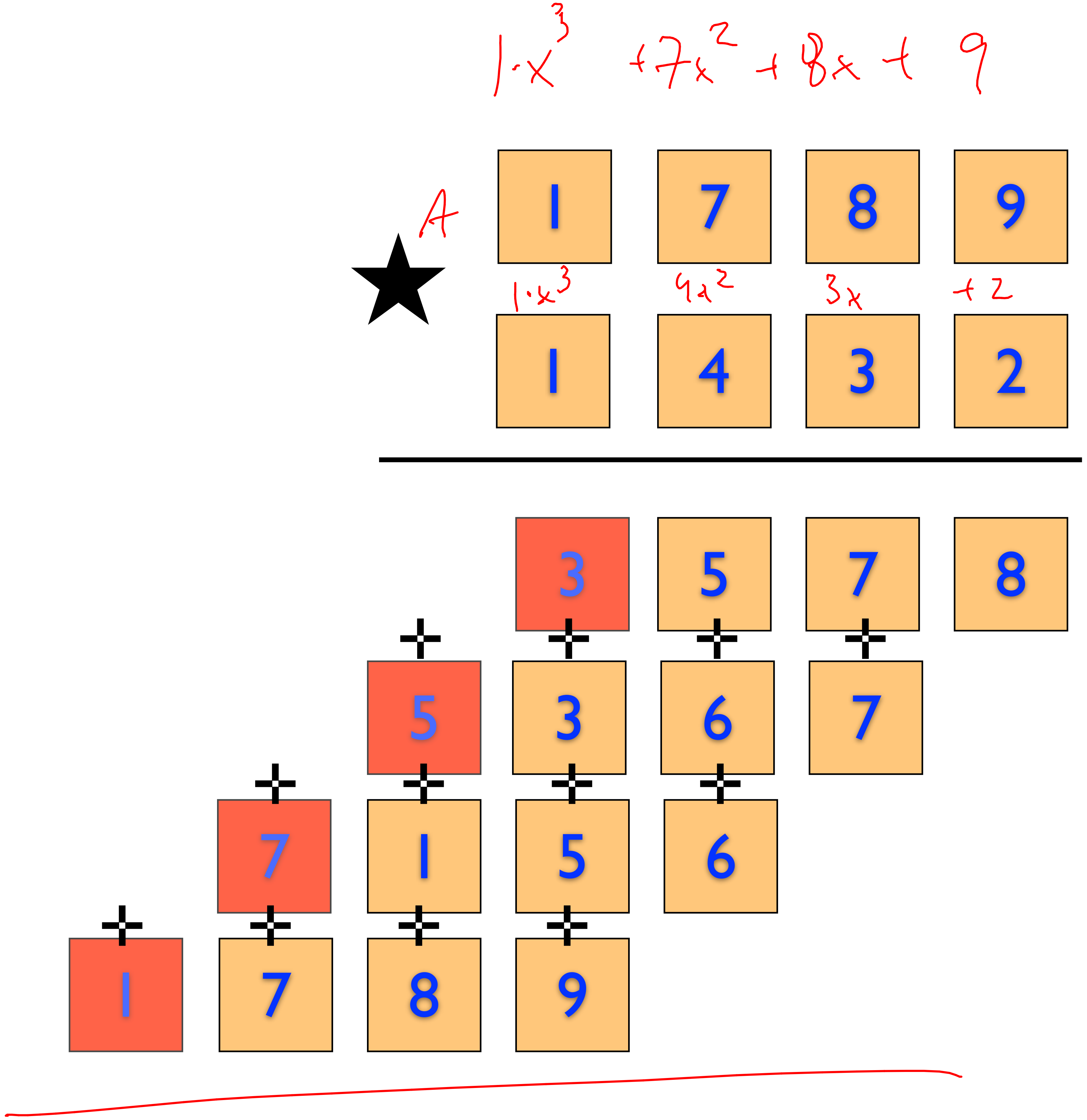
E[...] <- FFT(A_e) // eval Ae on n/2 roots of unity
O[...] <- FFT(A_o) // eval Ao on n/2 roots of unity

combine results using equation:

$$A(\omega_{i,n}) = A_e(\omega_{i,n}^2) + \omega_{i,n} A_o(\omega_{i,n}^2)$$

$$A(\omega_{i,n}) = A_e(\omega_{i \bmod n/2, \frac{n}{2}}) + \omega_{i,n} A_o(\omega_{i \bmod n/2, \frac{n}{2}})$$

Return n resulting values.

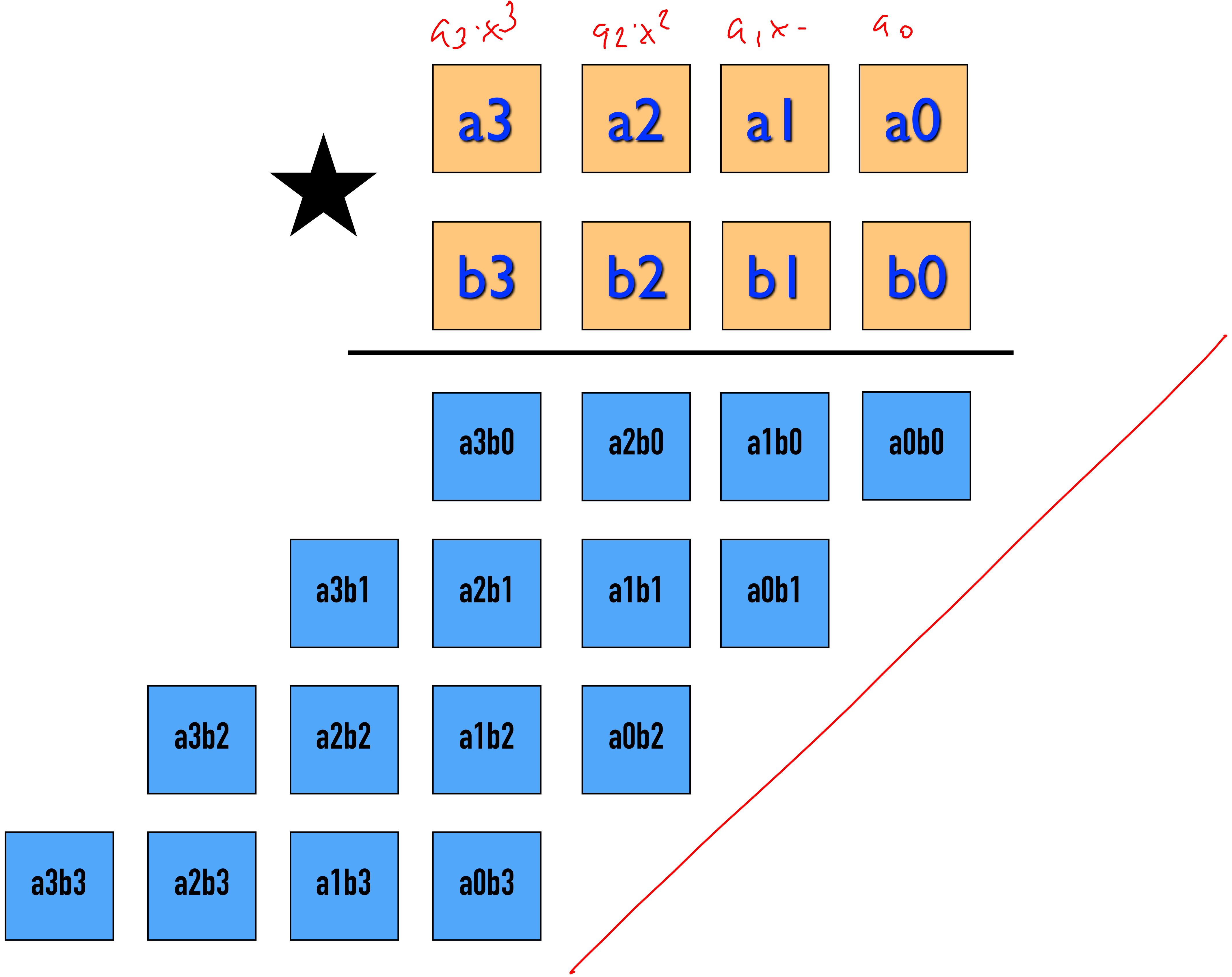


for $x = 10$

$A(x)\cdot B(x) = C(x)$

and then

return $C(10)$



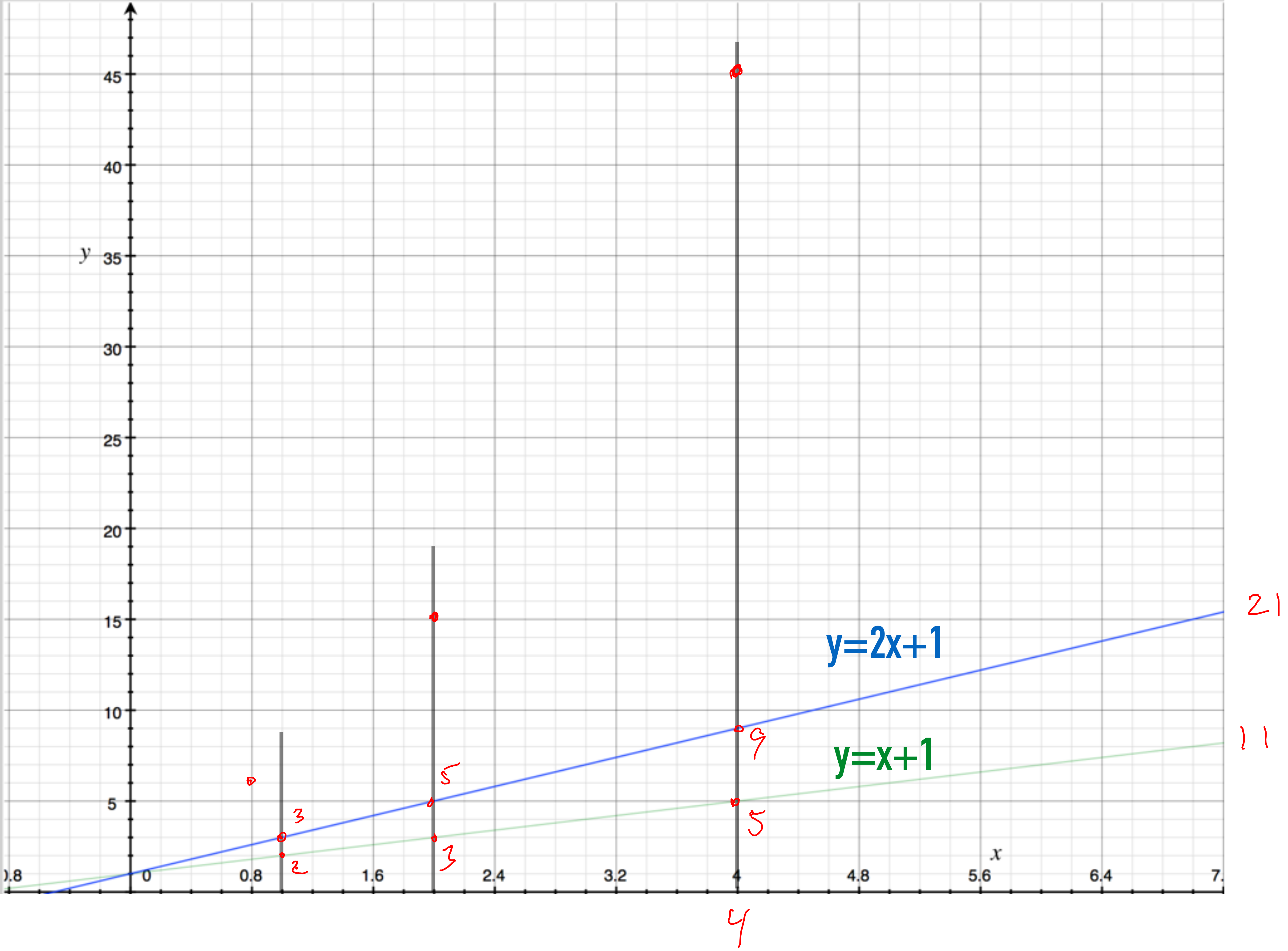
$a_3 \cdot x^3$
$a_2 \cdot x^2$
$a_1 x$
a_0
a3
a2
a1
a0
b3
b2
b1
b0
a3b0
a2b0
a1b0
a0b0
a3b1
a2b1
a1b1
a0b1
a3b2
a2b2
a1b2
a0b2
a3b3
a2b3
a1b3
a0b3

$$A(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

$$B(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

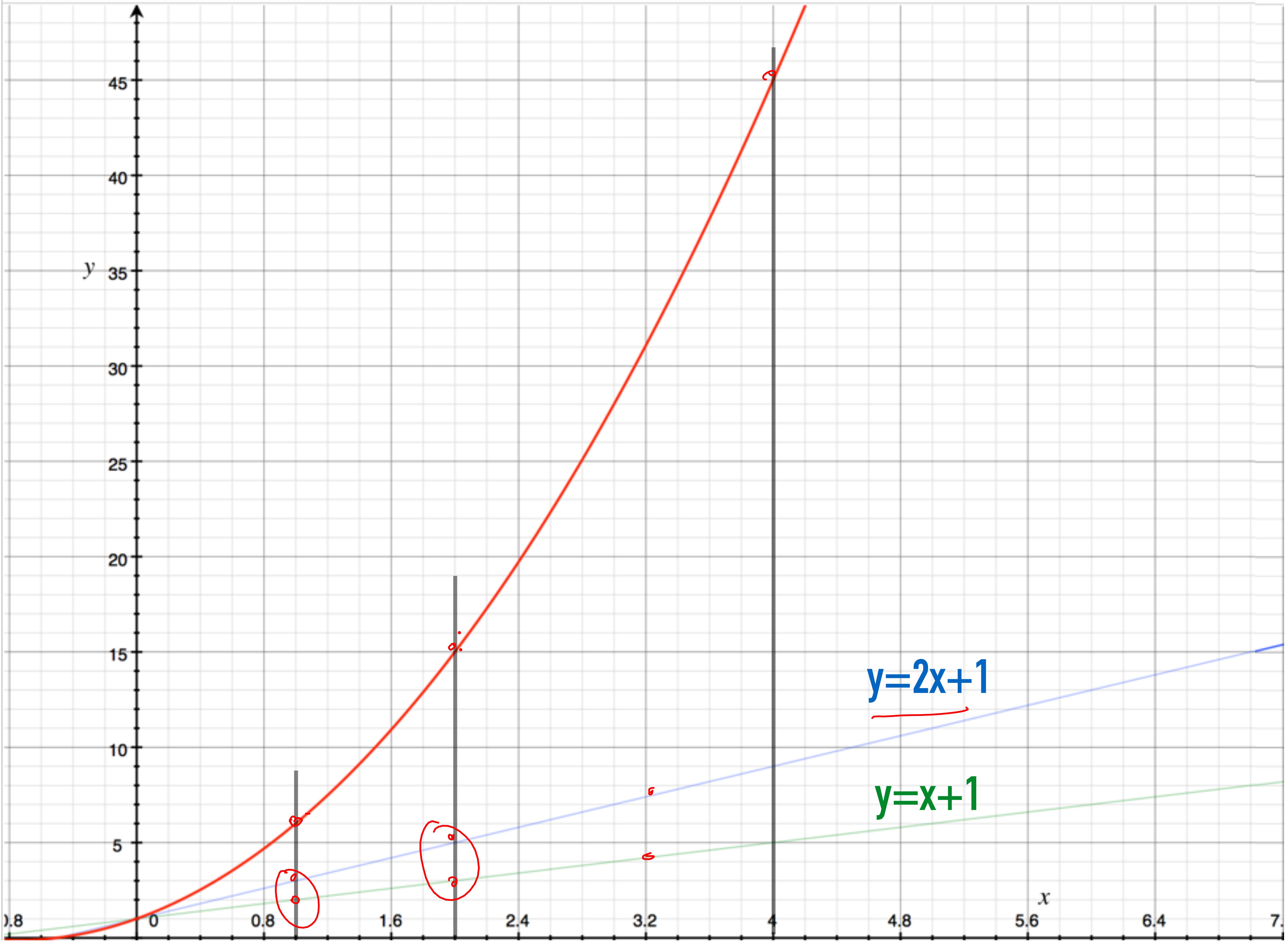
$$C(x) = \left\{ \begin{array}{l} a_3b_3x^6 + \\ (a_3b_2 + a_2b_3)x^5 + \\ (a_3b_1 + a_2b_2 + a_1b_3)x^4 + \\ (a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3)x^3 + \\ (a_2b_0 + a_1b_1 + a_0b_2)x^2 + \\ (a_1b_0 + a_0b_1)x + \\ a_0b_0 \end{array} \right.$$

$y=2x+1$


y
45
40
35
30
25
20
15
10
5
0
0.8
1.6
2.4
3.2
4
4.8
5.6
6.4
x
y=2x+1
y=x+1
21
11
9
5
3
5
3
2
4

$y=2x^2+3x+1$

2 3 1



21·11

1, 2, 4

a_3 a_2 a_1 a_0 ★ b_3 b_2 b_1 b_0

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

write input as an n-d poly

$$B(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

a_3 a_2 a_1 a_0 ★ b_3 b_2 b_1 b_0

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

n points

$A(\omega_0)$ $A(\omega_1)$ $A(\omega_2)$ .... $A(\omega_7)$

FFT

a3 a2 a1 a0 ★ b3 b2 b1 b0

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

$A(\omega_0)$	$A(\omega_1)$	$A(\omega_2)$	....	$A(\omega_7)$	**FFT**
$B(\omega_0)$	$B(\omega_1)$	$B(\omega_2)$	....	$B(\omega_7)$	**FFT**

$C(\omega_0)$ $C(\omega_1)$ . . $C(\omega_8)$ multiply

a_3 a_2 a_1 a_0 ★ b_3 b_2 b_1 b_0

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

$A(\omega_0)$ $A(\omega_1)$ $A(\omega_2)$ .... $A(\omega_7)$ **FFT**

$B(\omega_0)$ $B(\omega_1)$ $B(\omega_2)$ .... $B(\omega_7)$ **FFT**

$C(\omega_0)$ $C(\omega_1)$ $C(\omega_2)$ .... $C(\omega_7)$

a3 a2 a1 a0 ★ b3 b2 b1 b0

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

$A(\omega_0)$ $A(\omega_1)$ $A(\omega_2)$ .... $A(\omega_7)$ **FFT** n log n

$B(\omega_0)$ $B(\omega_1)$ $B(\omega_2)$ .... $B(\omega_7)$ **FFT** n log n

$C(\omega_0)$ $C(\omega_1)$ $C(\omega_2)$ .... $C(\omega_7)$ multiply $\tilde{O}(n)$

$$C(x) = c_0 + c_1 x + c_2 x^2 + \cdots c_7 x^7$$

IFFT n log n

application to mult

karatsuba

1 7 8 9 ★ 1 4 3 2

a b ★ c d

$$\Theta(n^{\log_2 3})$$

application to mult

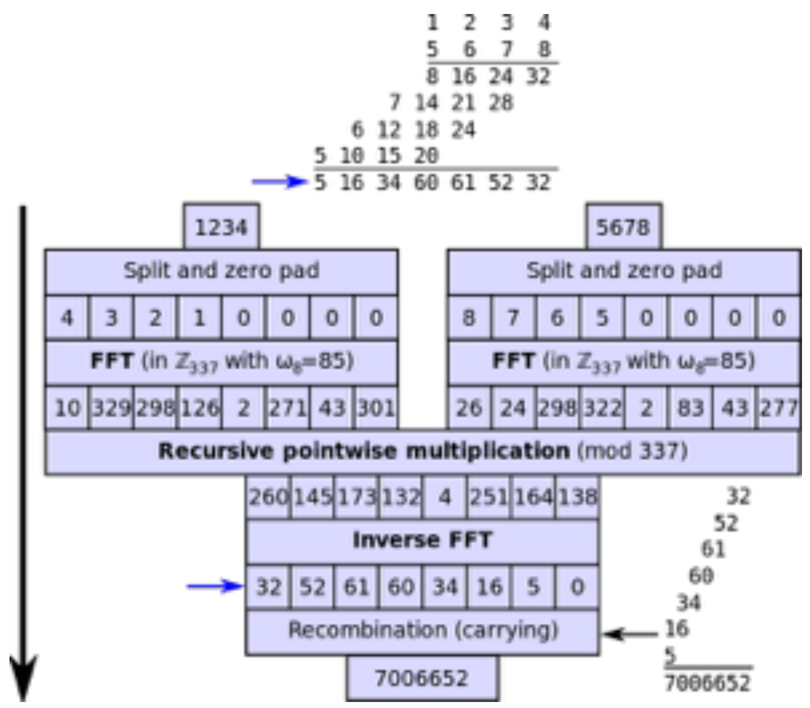
karatsuba

1	7	8	9	★	1	4	3	2
a		b			c		d	

$$T(n) = 3T(n/2) + 6O(n)$$

$$\Theta(n^{\log_2 3})$$

Multiplying n-bit numbers



https://en.wikipedia.org/wiki/File:Integer_multiplication_by_FFT.svg

Schönhage–Strassen '71 $O(n \log n \log \log n)$

Fürer '07 $O(n \log(n) 2^{\log^*(n)})$

A GMP-BASED IMPLEMENTATION OF SCHÖNHAGE-STRASSEN'S LARGE INTEGER MULTIPLICATION ALGORITHM

~2007

PIERRICK GAUDRY, ALEXANDER KRUPPA, AND PAUL ZIMMERMANN

ABSTRACT. Schönhage-Strassen's algorithm is one of the best known algorithms for multiplying large integers. Implementing it efficiently is of utmost importance, since many other algorithms rely on it as a subroutine. We present here an improved implementation, based on the one distributed within the GMP library. The following ideas and techniques were used or tried: faster arithmetic modulo $2^n + 1$, improved cache locality, Mersenne transforms, Chinese Remainder Reconstruction, the $\sqrt{2}$ trick, Harley's and Granlund's tricks, improved tuning. We also discuss some ideas we plan to try in the future.

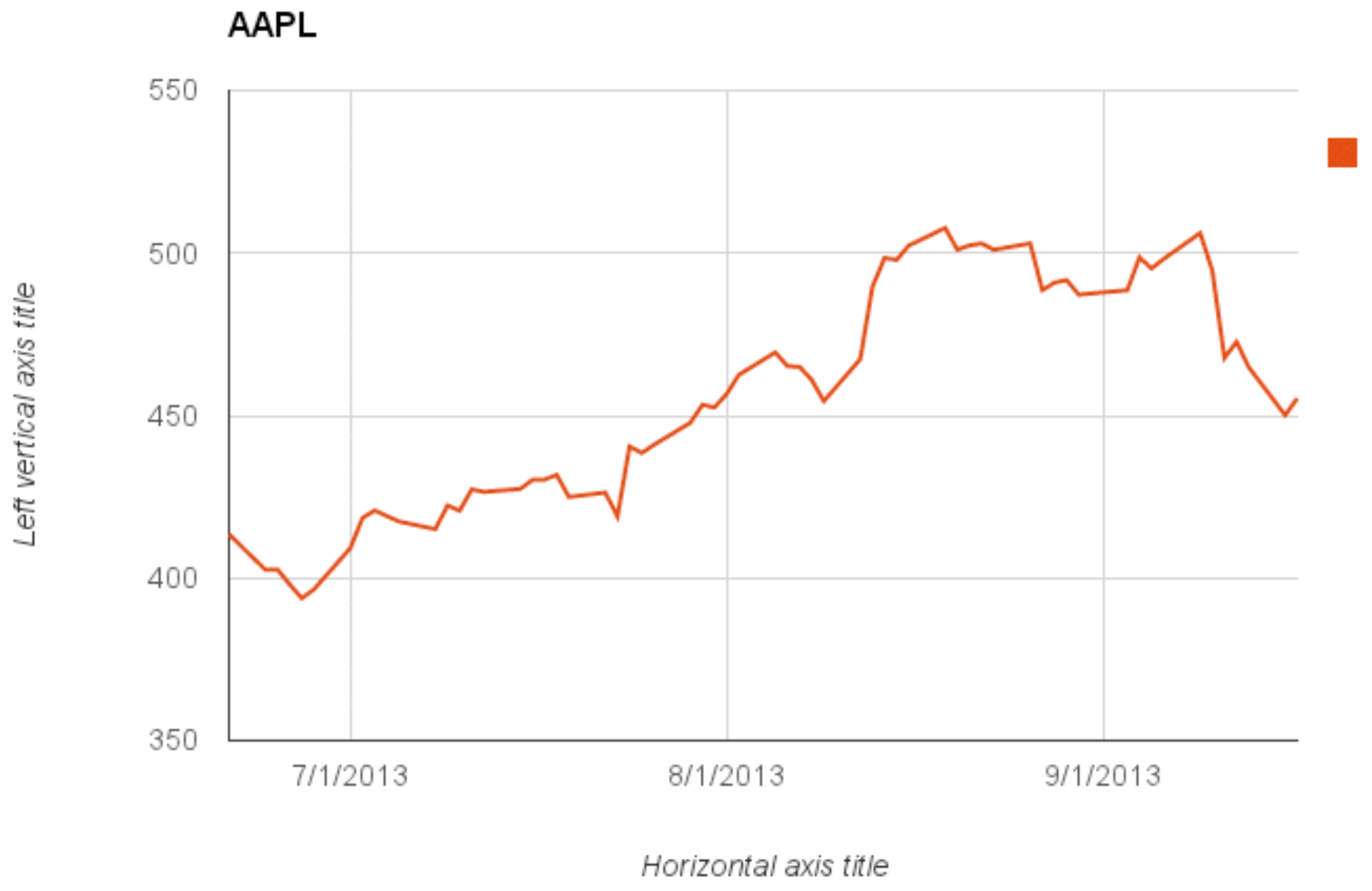
GMP

INTRODUCTION

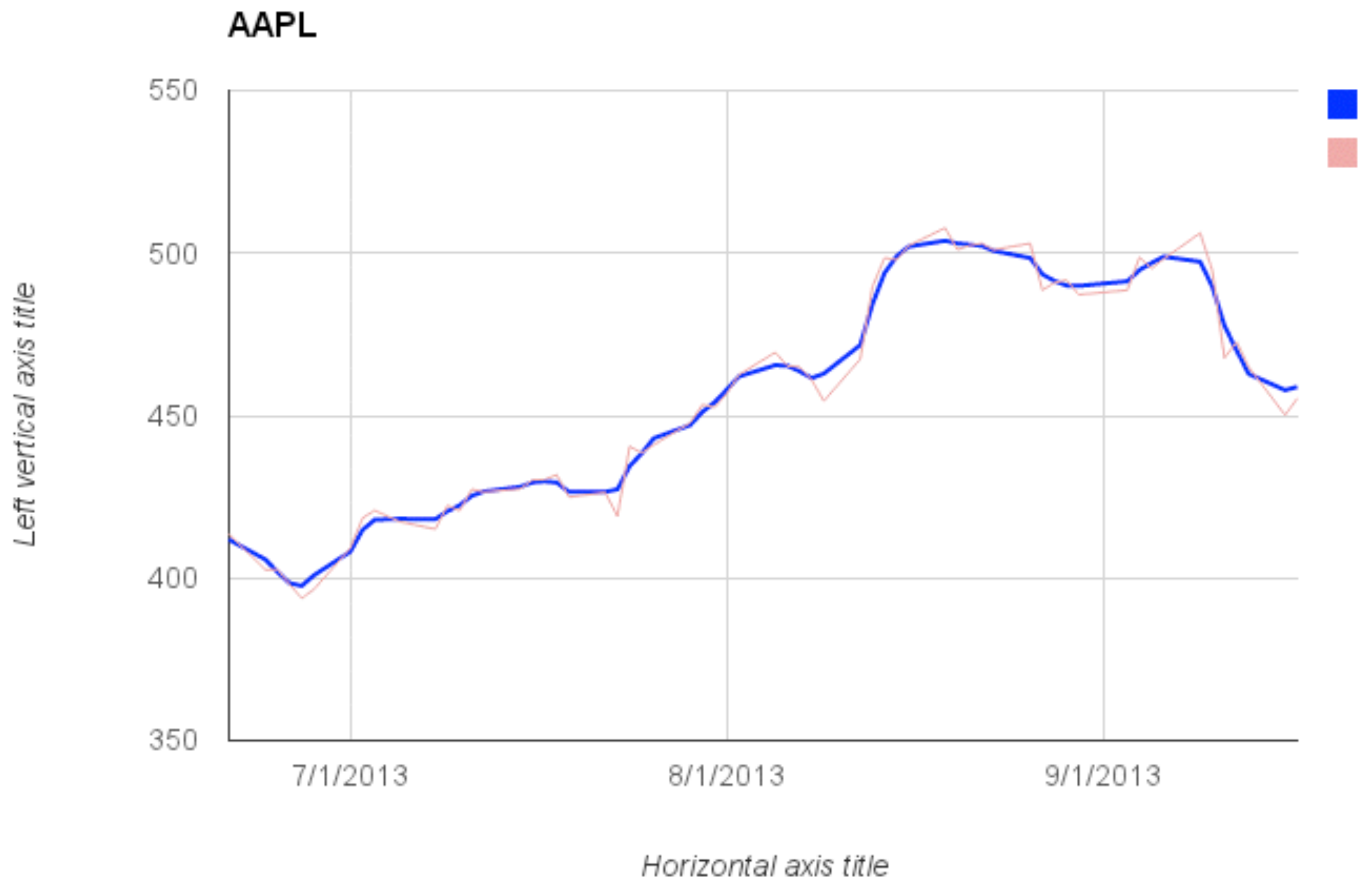
Since Schönhage and Strassen have shown in 1971 how to multiply two N-bit integers in $O(N \log N \log\log N)$ time [21], several authors showed how to reduce other operations — inverse, division, square root, gcd, base conversion, elementary functions — to multiplication, possibly with $\log N$ multiplicative factors [5, 8, 17, 18, 20, 23]. It has now become common practice to express complexities in terms of the cost $M(N)$ to multiply two N-bit numbers, and many researchers tried hard to get the best possible constants in front of $M(N)$ for the above-mentioned operations (see for example [6, 16]).

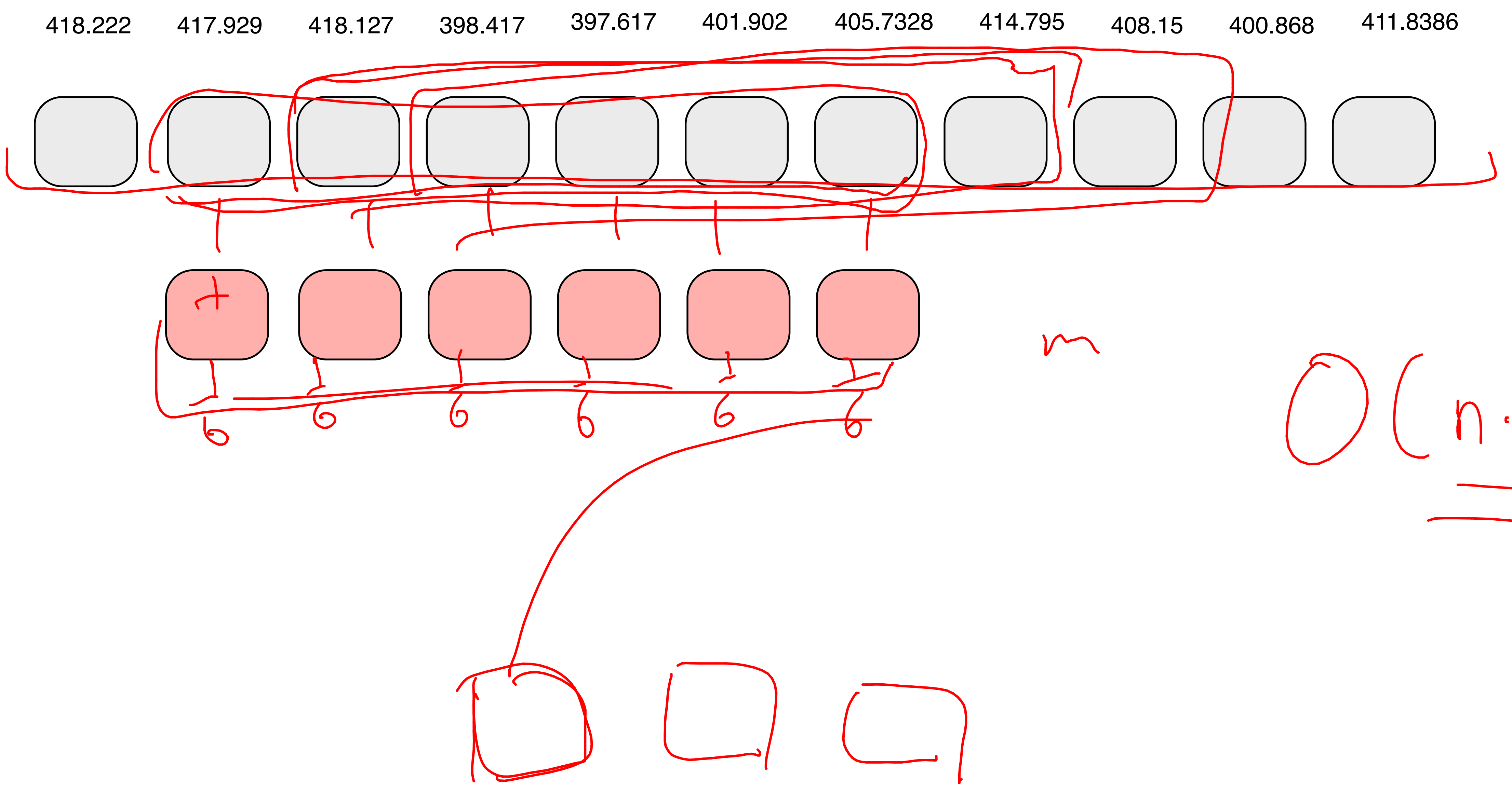
Strangely, much less effort was made for decreasing the implicit constant in $M(N)$ itself, although any gain on that constant will give a similar gain on all multiplication-based operations. Some authors reported on implementations of large integer arithmetic for specific hardware or as part of a number-theoretic project [2, 10]. In this article we concentrate on the question of an optimized implementation of Schönhage-Strassen's algorithm on a classical workstation.

Applications of FFT



Applications of FFT





n data items

m

$O(n \cdot m)$

Convolution $\Theta(n \cdot \log m)$ time

String matching with *

ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA

DNA sequence

4B

Looking for all occurrences of

GGC*GAG*C*GC ~ millions.

where I don't care what the * symbol is.

$O(4B \cdot 1m)$

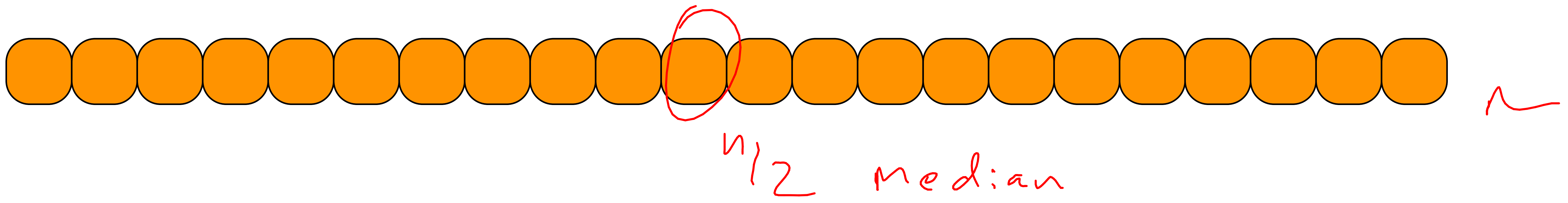
$10^9 \cdot 10^6 \sim 10^{15}$

$10^9 \cdot \log(10^8) = 10^{10}$

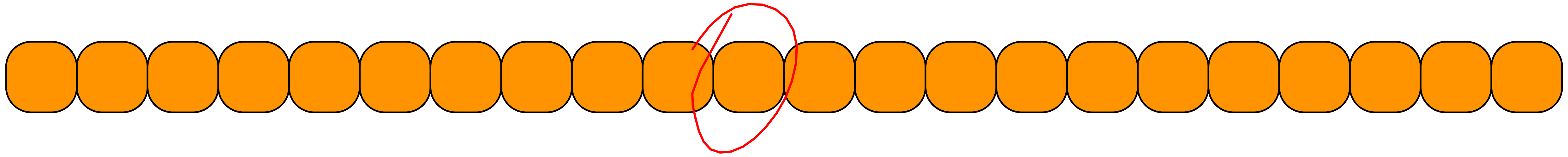
MEDIAN

ith order

statistic



problem: given a list of **n** elements, find the element of rank $\lfloor \mathbf{n}/2 \rfloor$. (half are larger, half are smaller)

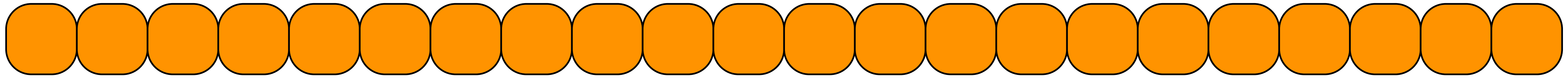


problem: given a list of **n** elements, find the element of rank n/2. (half are larger, half are smaller)

can generalize to i

first solution: sort and pluck.

$$O(n \log n)$$

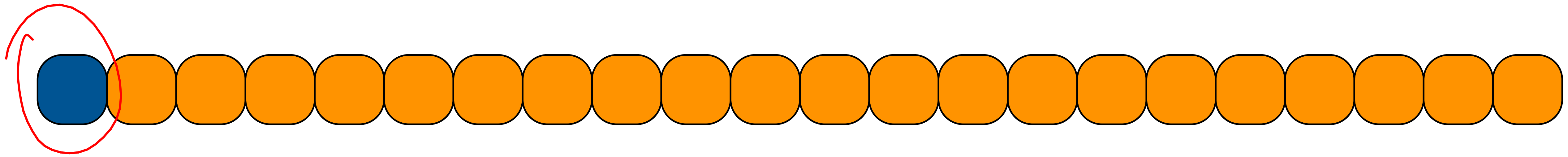


problem: given a list of **n** elements, find the element of rank **i**.

key insight:

we do not have to "fully" sort.

semi sort can suffice.

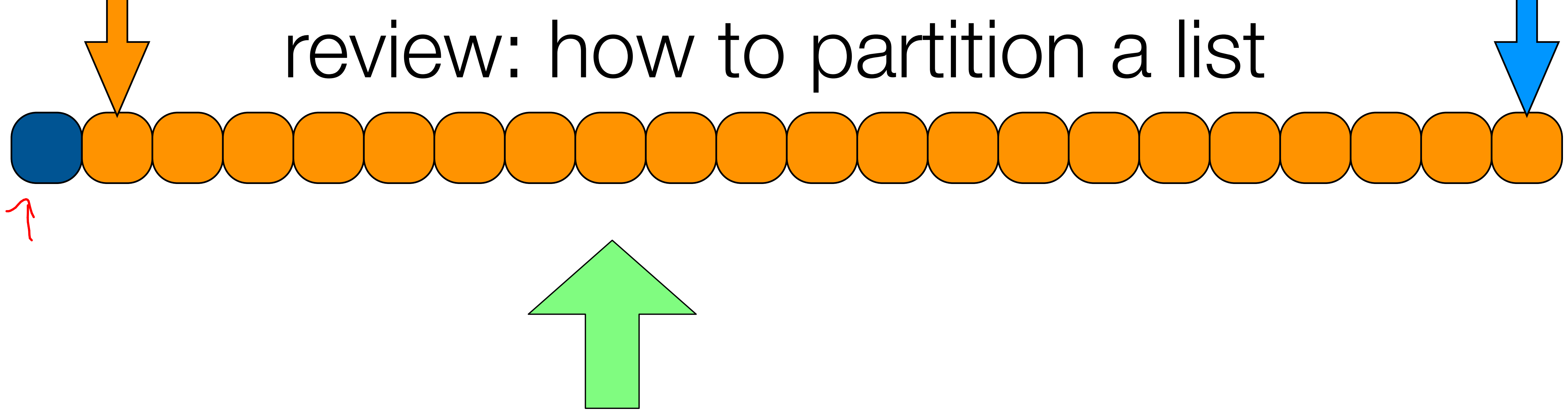


Partition element

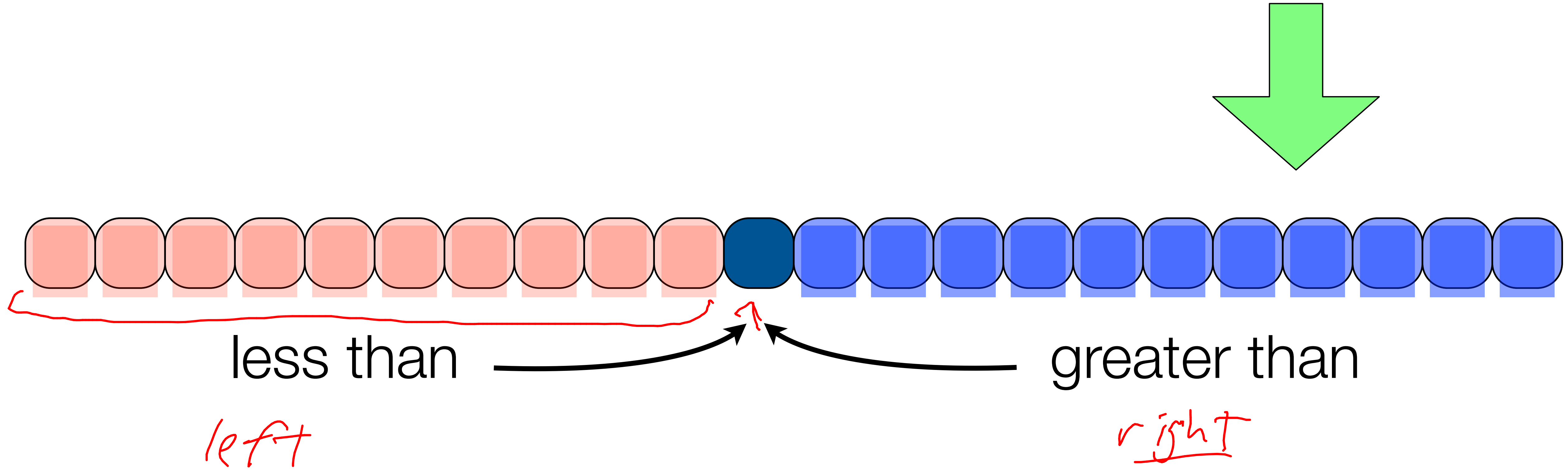
pick first element
partition list about this one
see where we stand

review: how to partition a list

review: how to partition a list

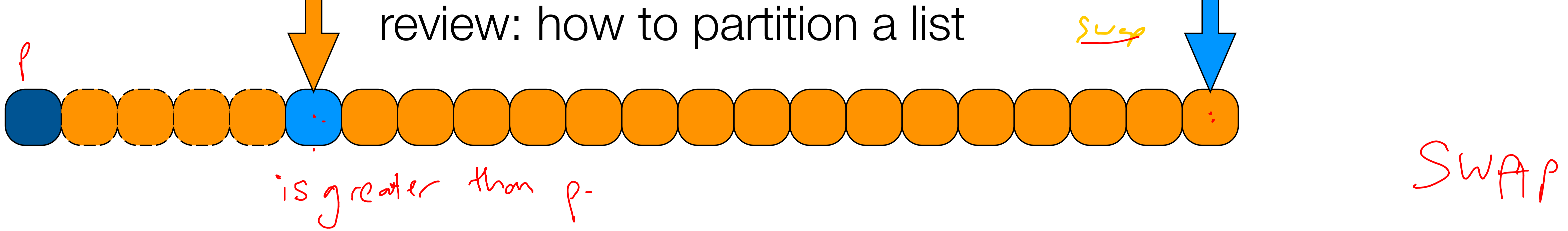


GOAL: start with THIS LIST and END with THAT LIST

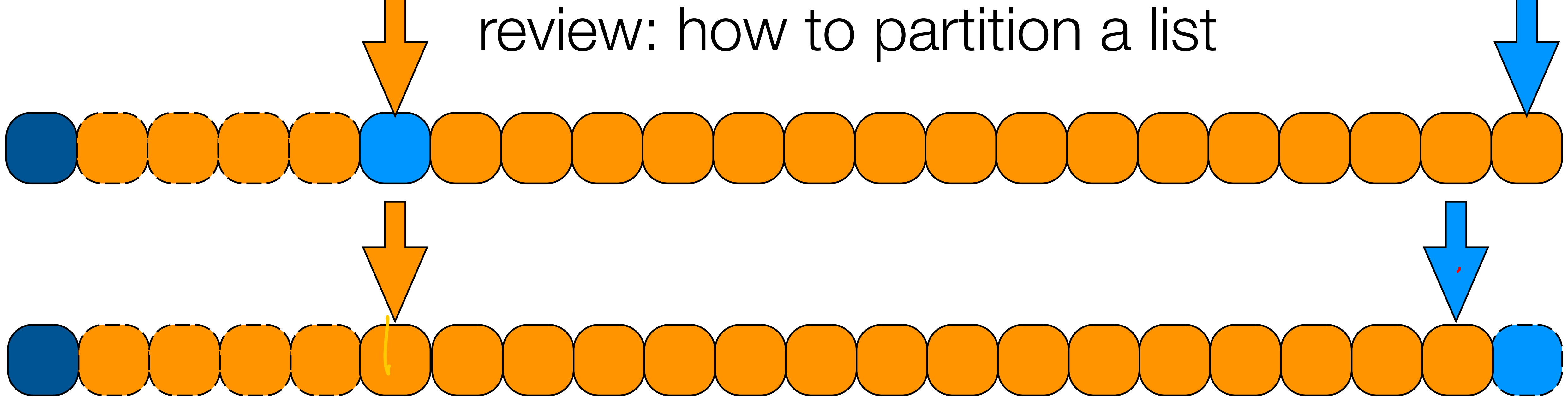


review: how to partition a list

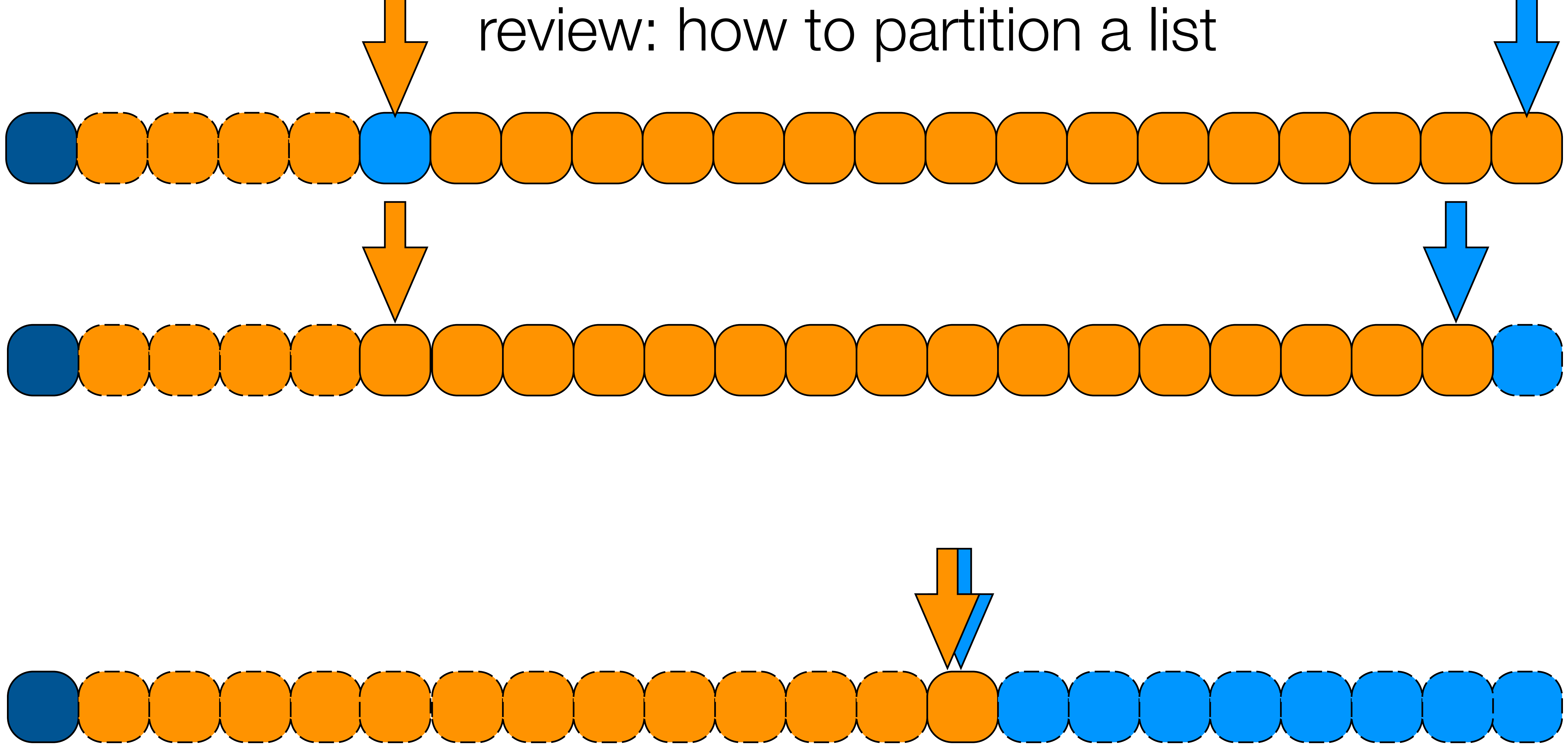
review: how to partition a list



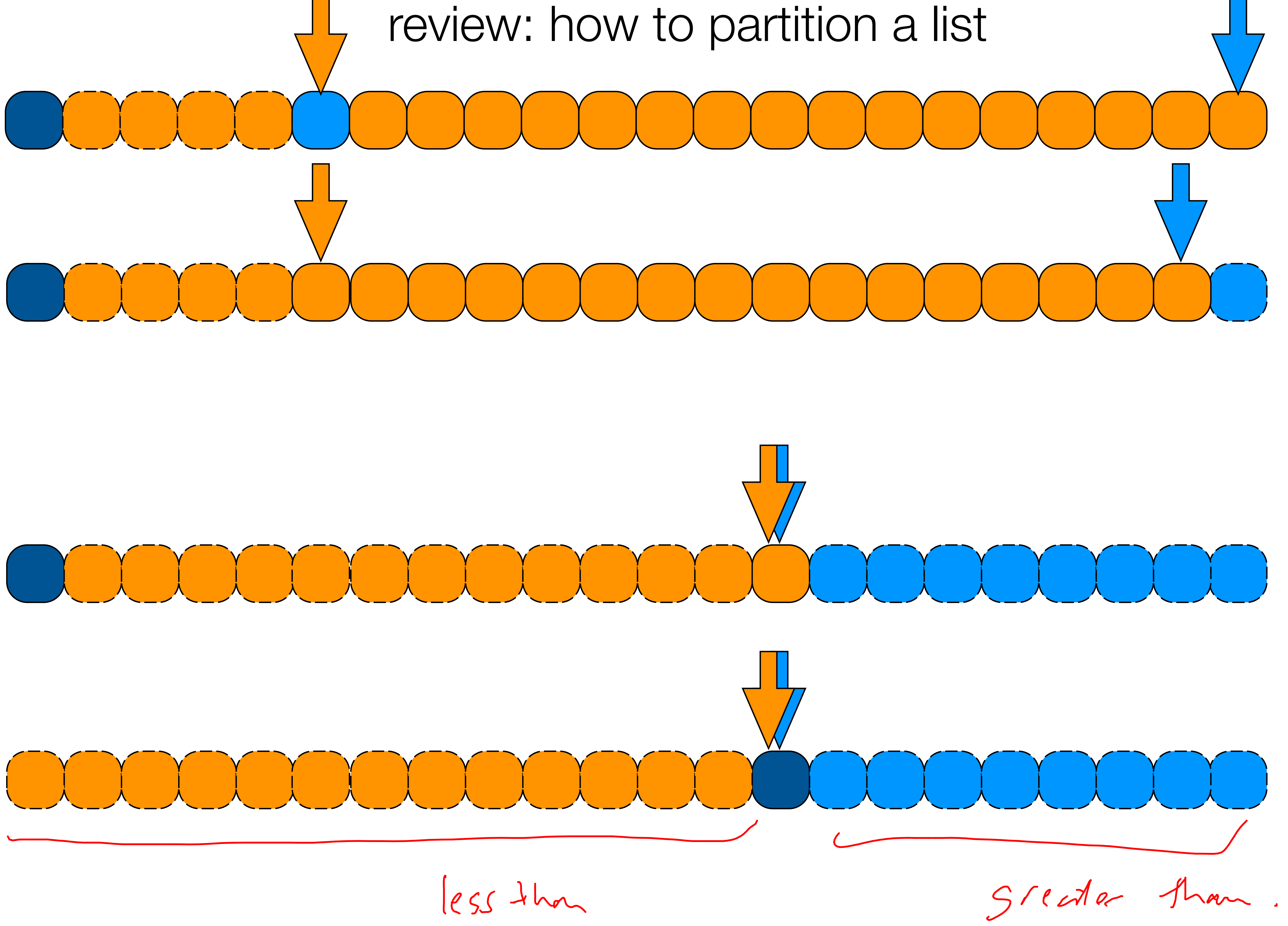
review: how to partition a list



review: how to partition a list

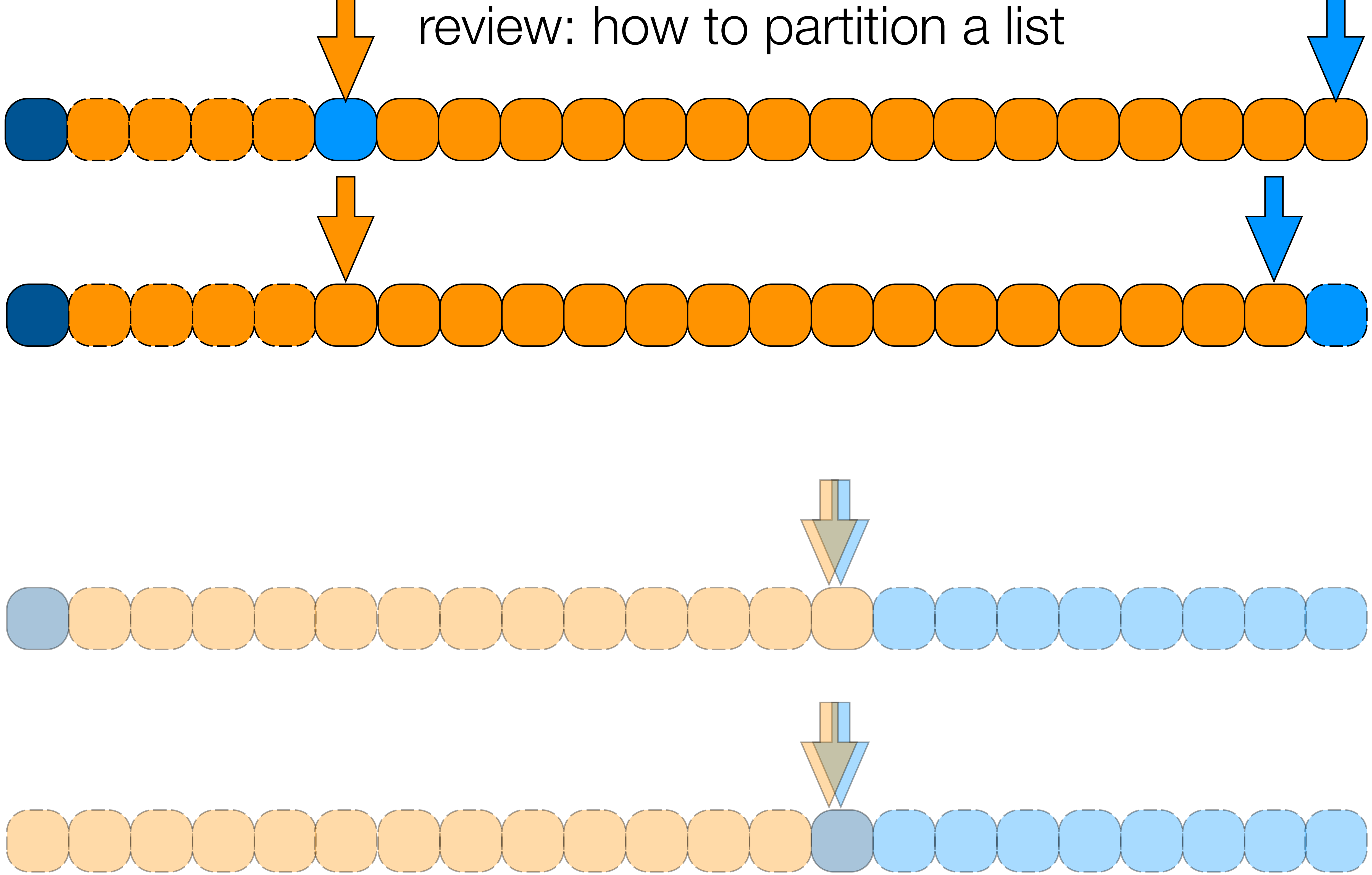


review: how to partition a list

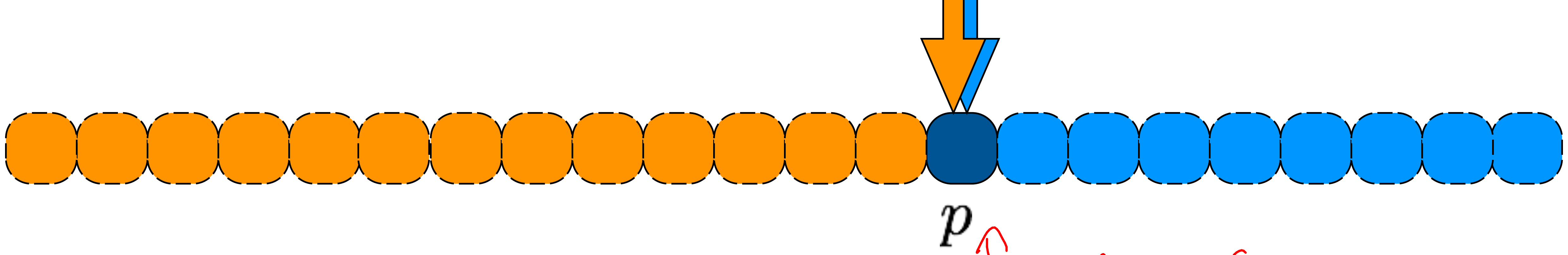


$\underline{\Theta(n)}$

review: how to partition a list



partitioning a list about an element takes linear time.



index of the partition

select $(\underline{i}, A[1, \ldots, n])$

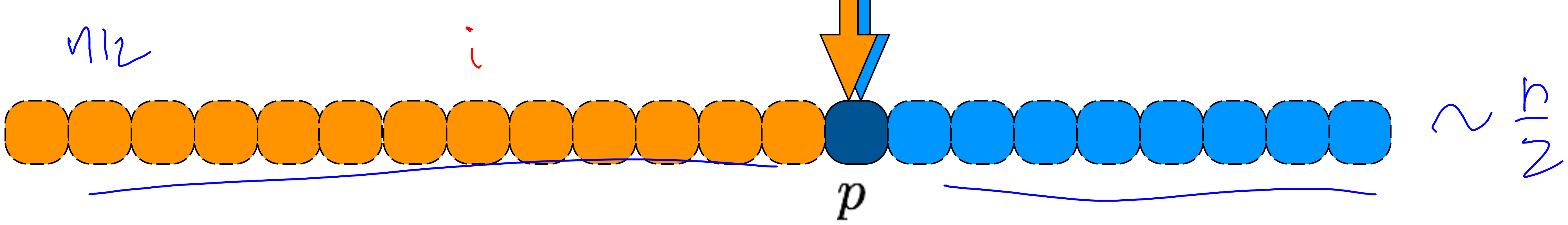
Base case if $|A| \leq 2$

$p \leftarrow$ partition(A) so that all elements are either $<$ or $>$ p.

if $(i == p)$ return $A[p]$.

else $(i < p)$ select$(i, A[0 \ldots p-1])$

else select$(i - p - 1, A[p, \ldots n])$



select $(i, A[1,\ldots,n])$

handle base case.

partition list about first element $\rightarrow \Theta(n)$

if pivot p is position **i**, return pivot $\rightarrow \Theta(1)$

else if pivot p is in position > i select $(i, A[1,\ldots,p-1])$

else select $((i-p-1), A[p+1,\ldots,n])$

Big idea I

$$T(n) = \cancel{2}\, T\left(\frac{n}{2}\right) + \Theta(n) \implies \Theta(n)$$

select $(i, A[1, \ldots, n])$

Assume our partition always splits list into two eql parts

handle base case.

partition list about first element

if pivot is position **i**, return pivot

else if pivot is in position > i select $(i, A[1, \ldots, p-1])$

else select $((i-p-1), A[p+1, \ldots, n])$

select $(i, A[1, \ldots, n])$

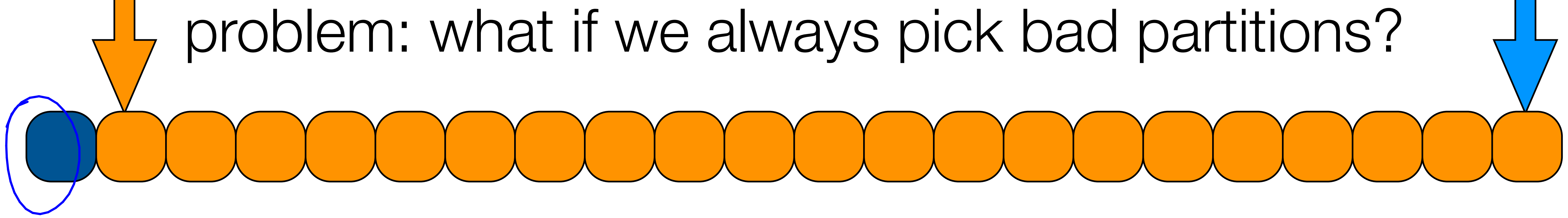
Assume our partition always splits list into two eql parts

handle base case.
partition list about first element
if pivot is position i, return pivot
else if pivot is in position > i select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

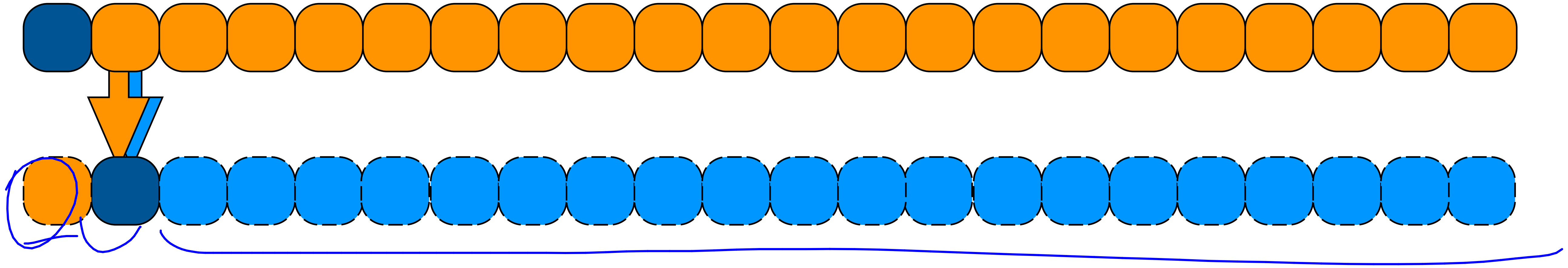
$$T(n) = T(n/2) + O(n)$$

$$\Theta(n)$$

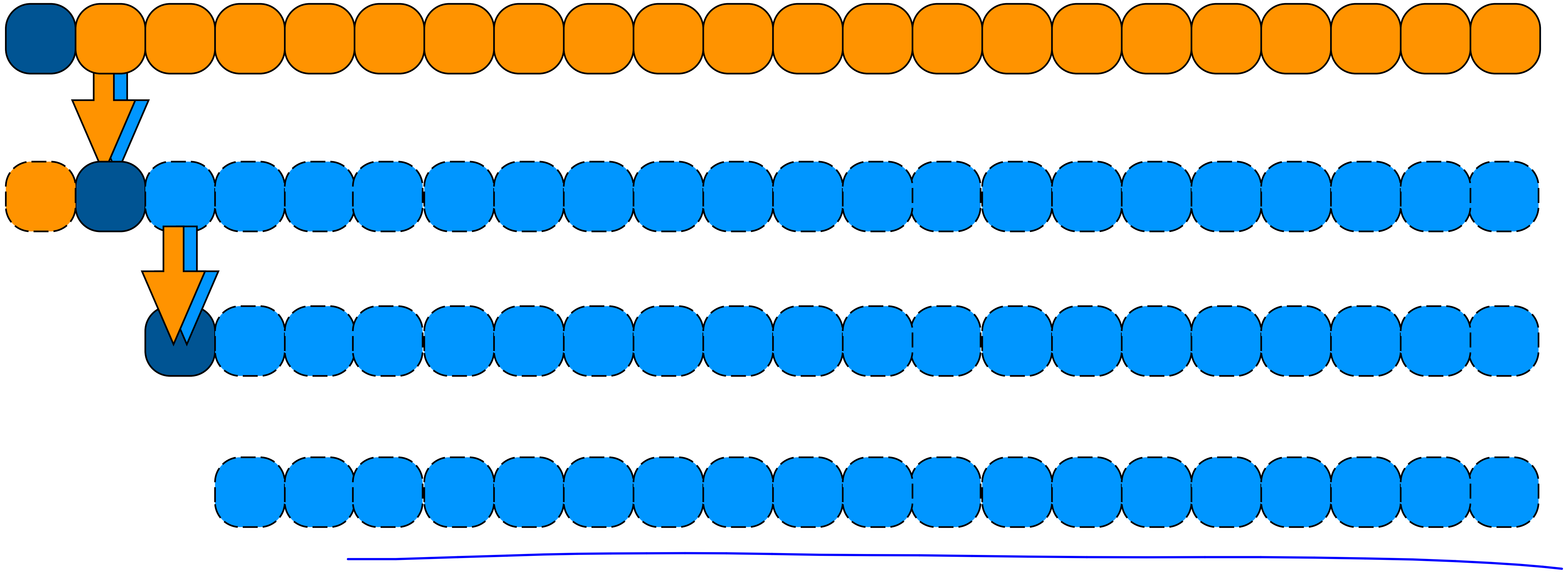
problem: what if we always pick bad partitions?

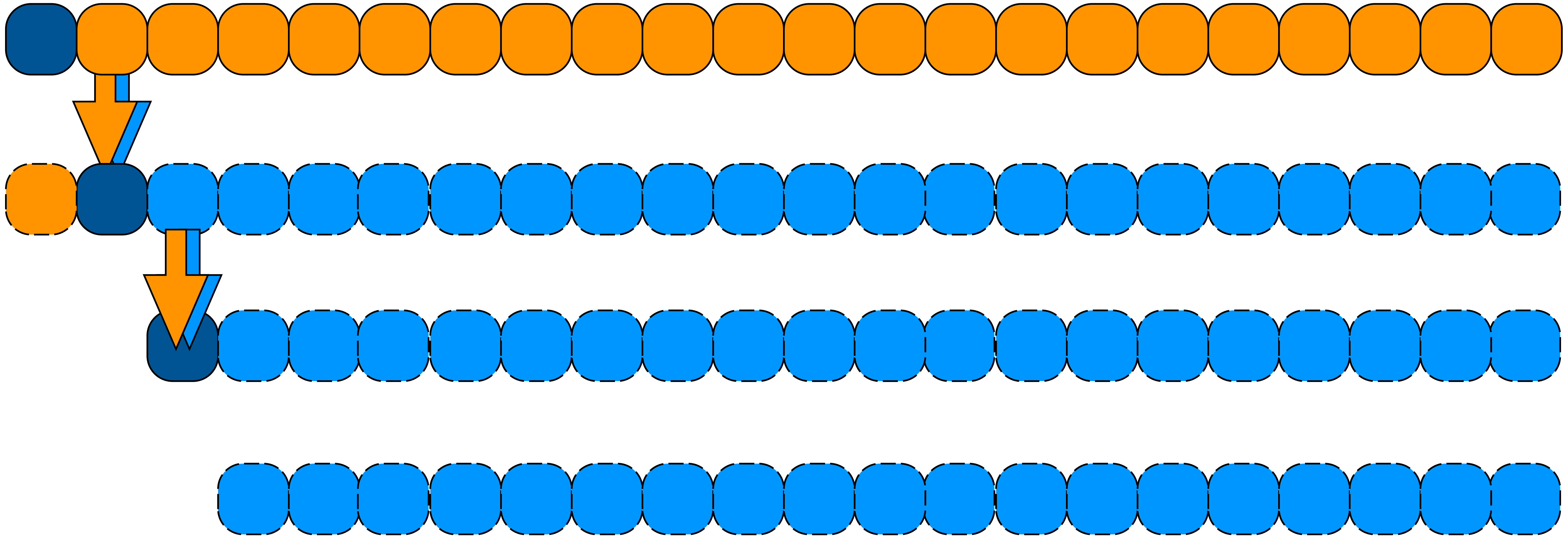


problem: what if we always pick bad partitions?



problem: what if we always pick bad partitions?





problem: what if we always pick bad partitions?

$$T(n) = T(n-5) + \Theta(n) = \Theta(n^2)$$

$$= T(n-5) + \Theta(n)$$

select $(i, A[1, \ldots, n])$

handle base case.
partition list about first element
if pivot is position **i**, return pivot
else if pivot is in position > **i** select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

select $(i, A[1, \ldots, n])$

handle base case.
partition list about first element
if pivot is position **i**, return pivot
else if pivot is in position > **i** select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

if we always pick bad partitions !!

$$T(n) = T(n-1) + O(n)$$

$$\Theta(n^2)$$

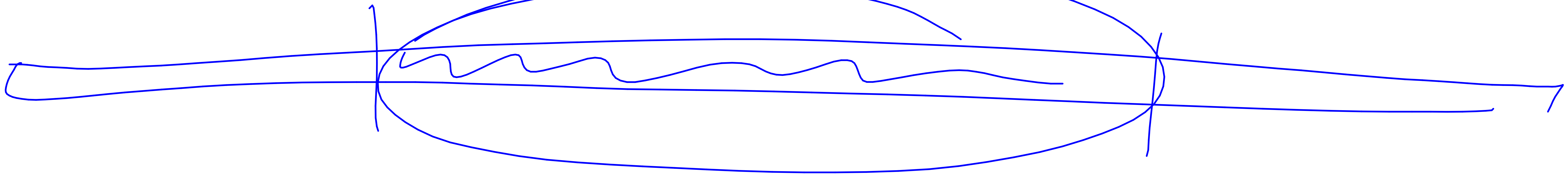
Needed:

a good partition element

partition $(A[1, \ldots, n])$

One in which 30% of the elements are smaller, 30% are larger –

good partition element.



Needed:

a good partition element

partition $(A[1,\ldots,n])$

produce an element where
30% smaller, 30% larger

solution:
bootstrap

1975 Floyd →

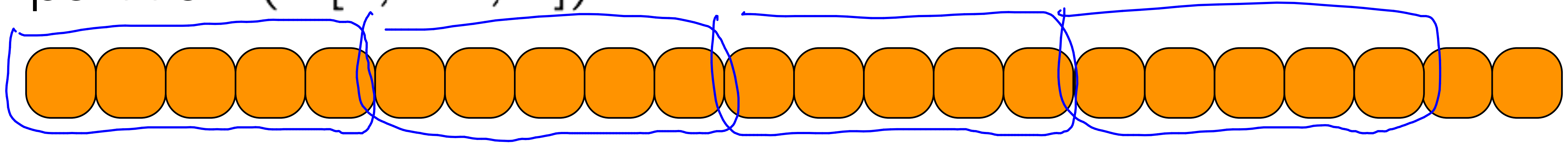
Rivest →

image: gucci

image: mark nason

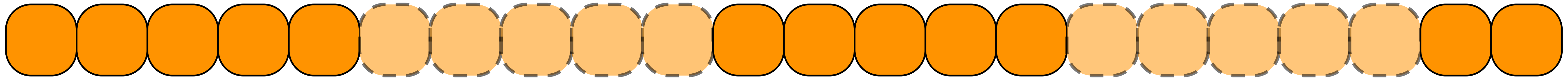
image: d&g

partition $(A[1, \ldots, n])$

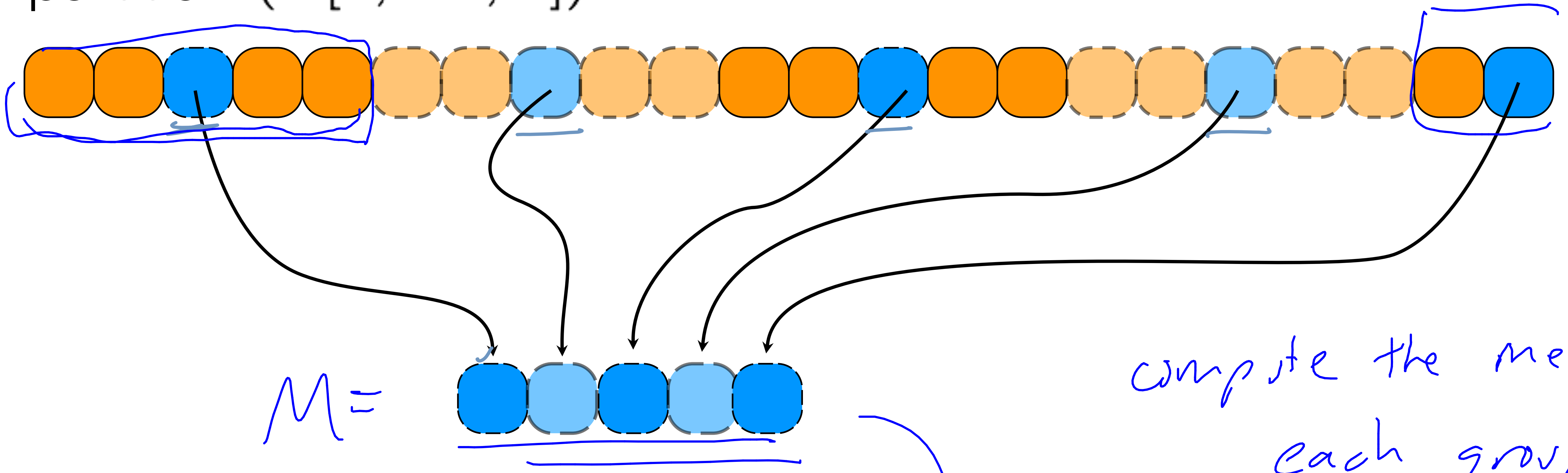


divide list into
chunks of 5

partition $(A[1, \ldots, n])$



partition $(A[1,\ldots,n])$



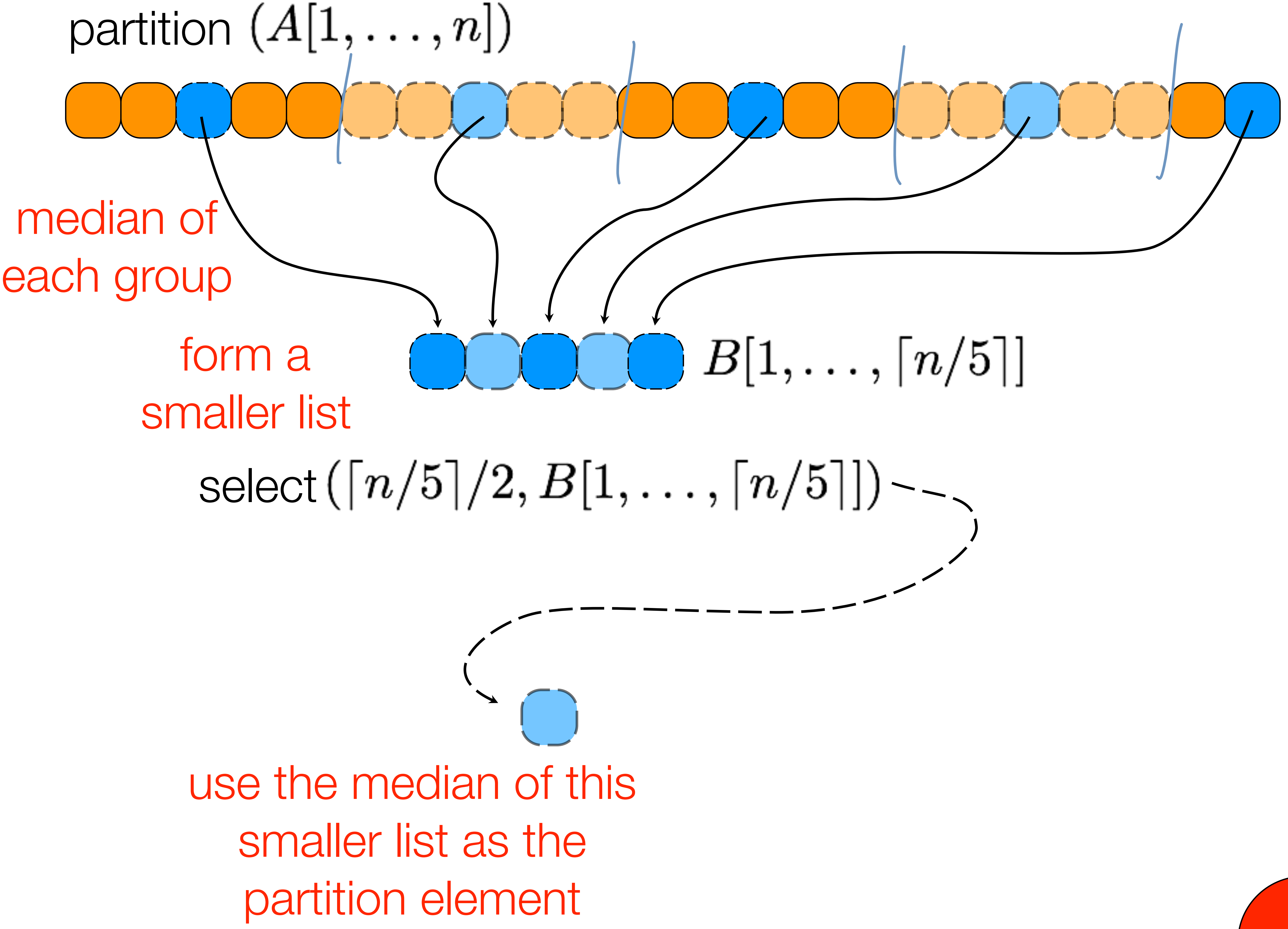
$M =$

compute the medians of each group

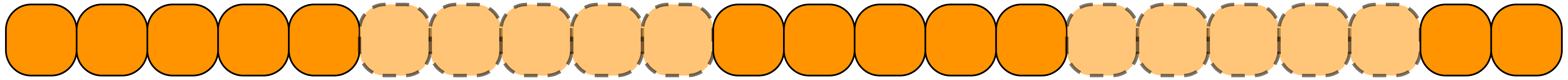
how big is this list $\lceil \frac{n}{5} \rceil$

Use select$\left(\frac{n}{10}, M\right)$ to pick p.

our partition element.

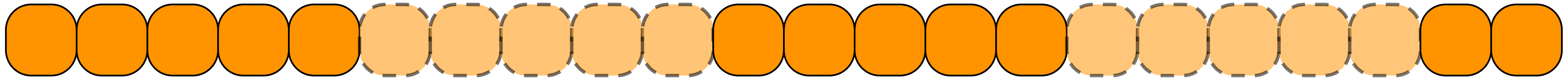

partition $(A[1,\ldots,n])$
median of each group
form a smaller list
$B[1,\ldots,\lceil n/5\rceil]$
select $(\lceil n/5\rceil/2, B[1,\ldots,\lceil n/5\rceil])$
use the median of this smaller list as the partition element

partition $(A[1, \ldots, n])$



1.
2.
3.
4.
5.

partition $(A[1,\ldots,n])$

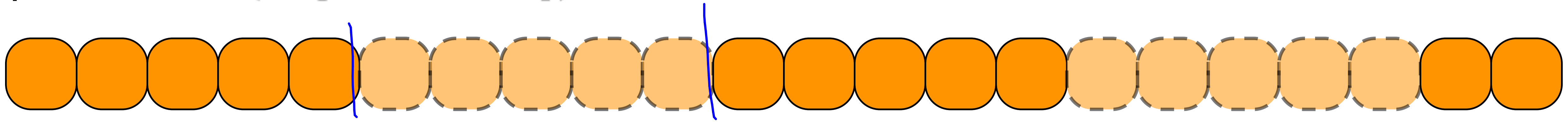


- divide list into groups of 5 elements
- find median of each small list
- gather all medians

call select(...) on this sublist to find median

return the result

partition $(A[1,...,n])$



divide list into groups of 5 elements $\rightarrow \Theta(n)$

find median of each small list $\rightarrow \Theta(n)$

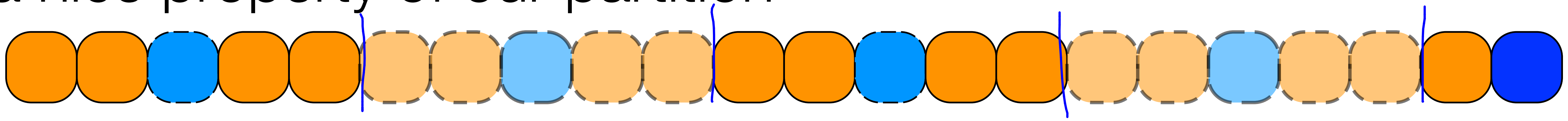
gather all medians $\rightarrow \Theta(n/5)$

call select(...) on this sublist to find median $\rightarrow S\left(\frac{n}{5}\right)$

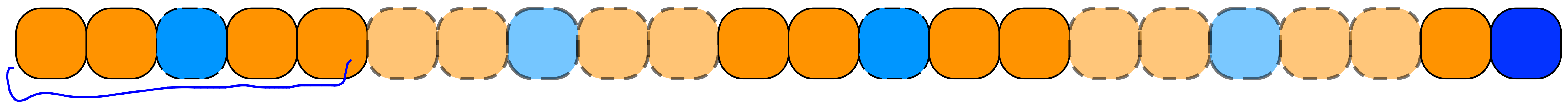
return the result

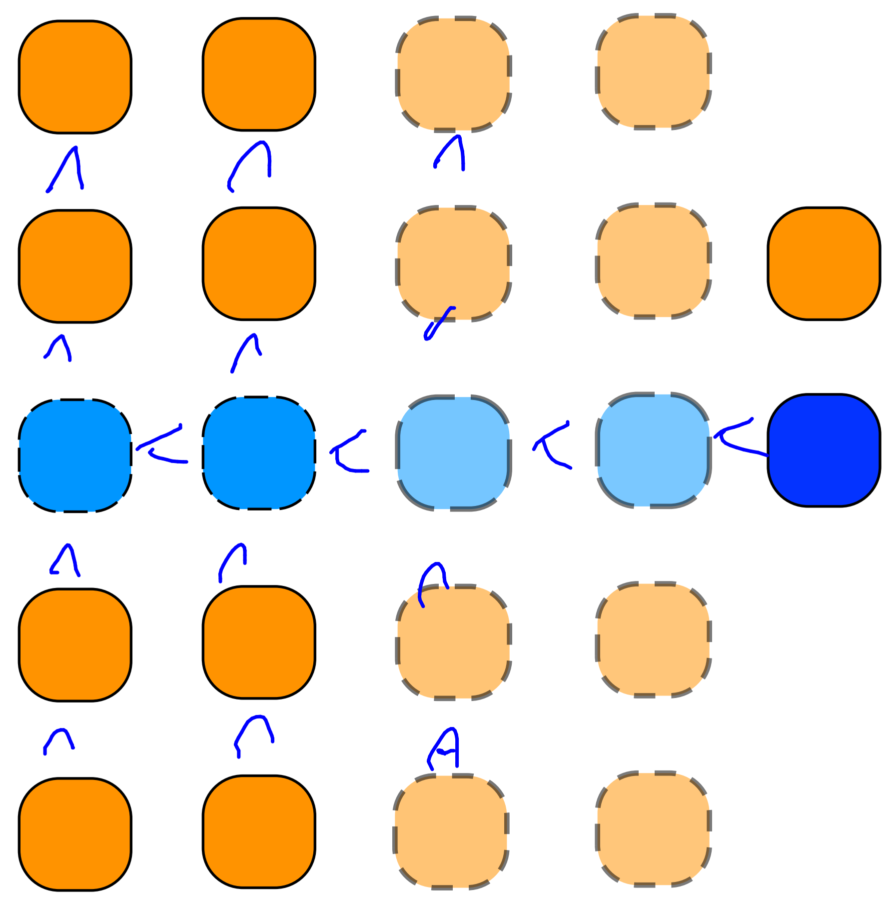
$$P(n) = S(\lceil n/5 \rceil) + O(n)$$

a nice property of our partition

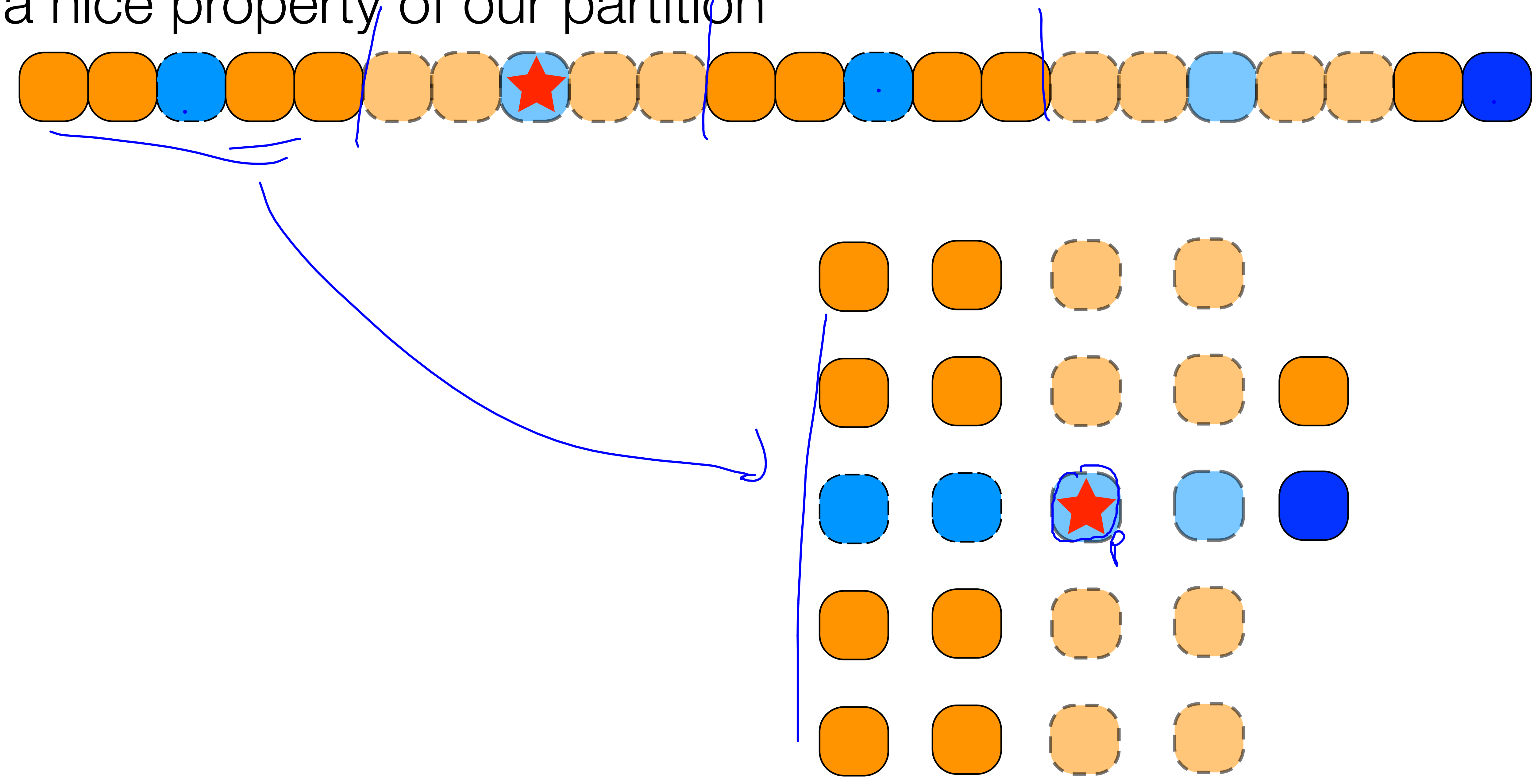


a nice property of our partition



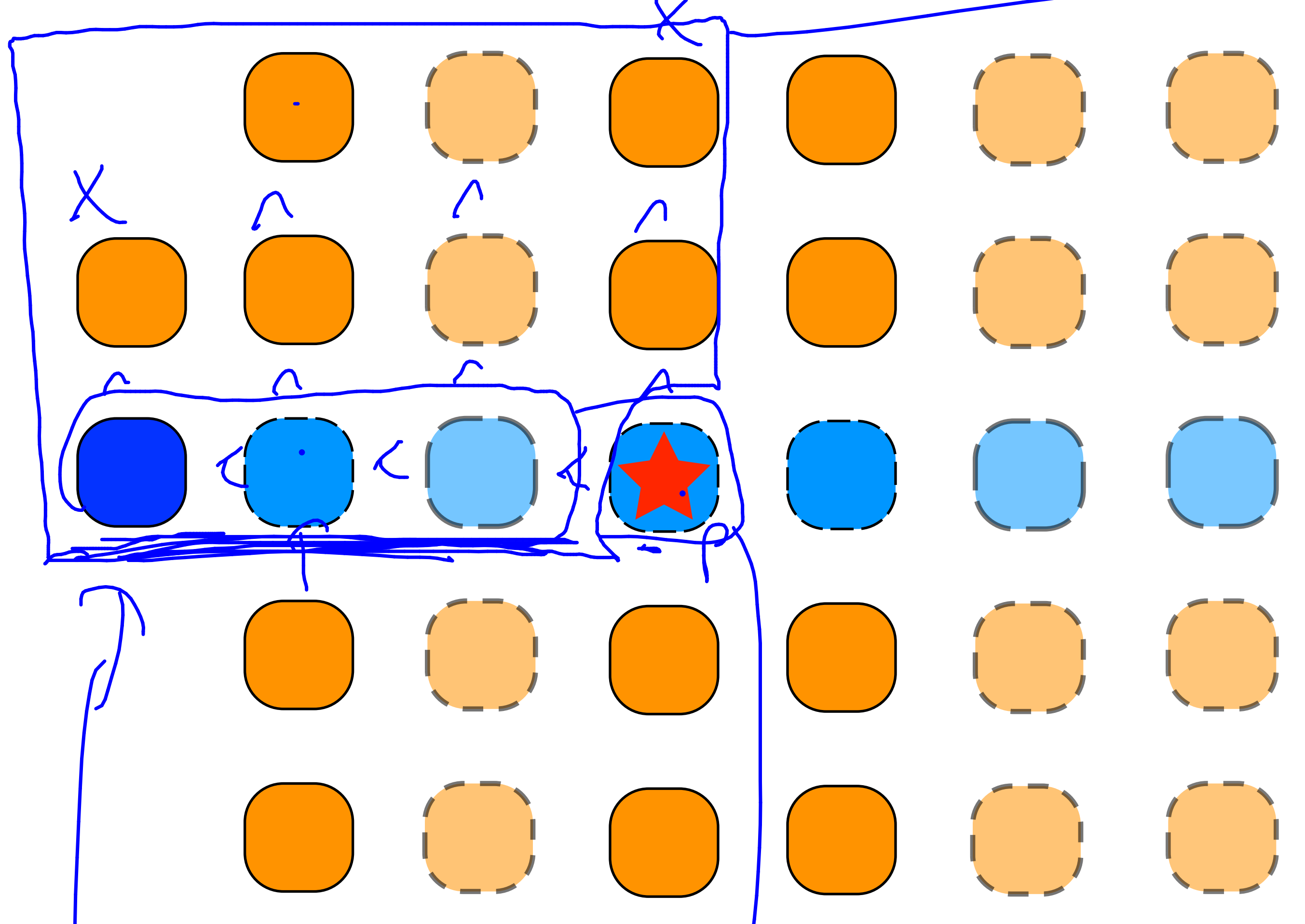


a nice property of our partition



SWITCH TO A BIGGER EXAMPLE

$$3\left(\left\lfloor \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rfloor - 2\right)$$

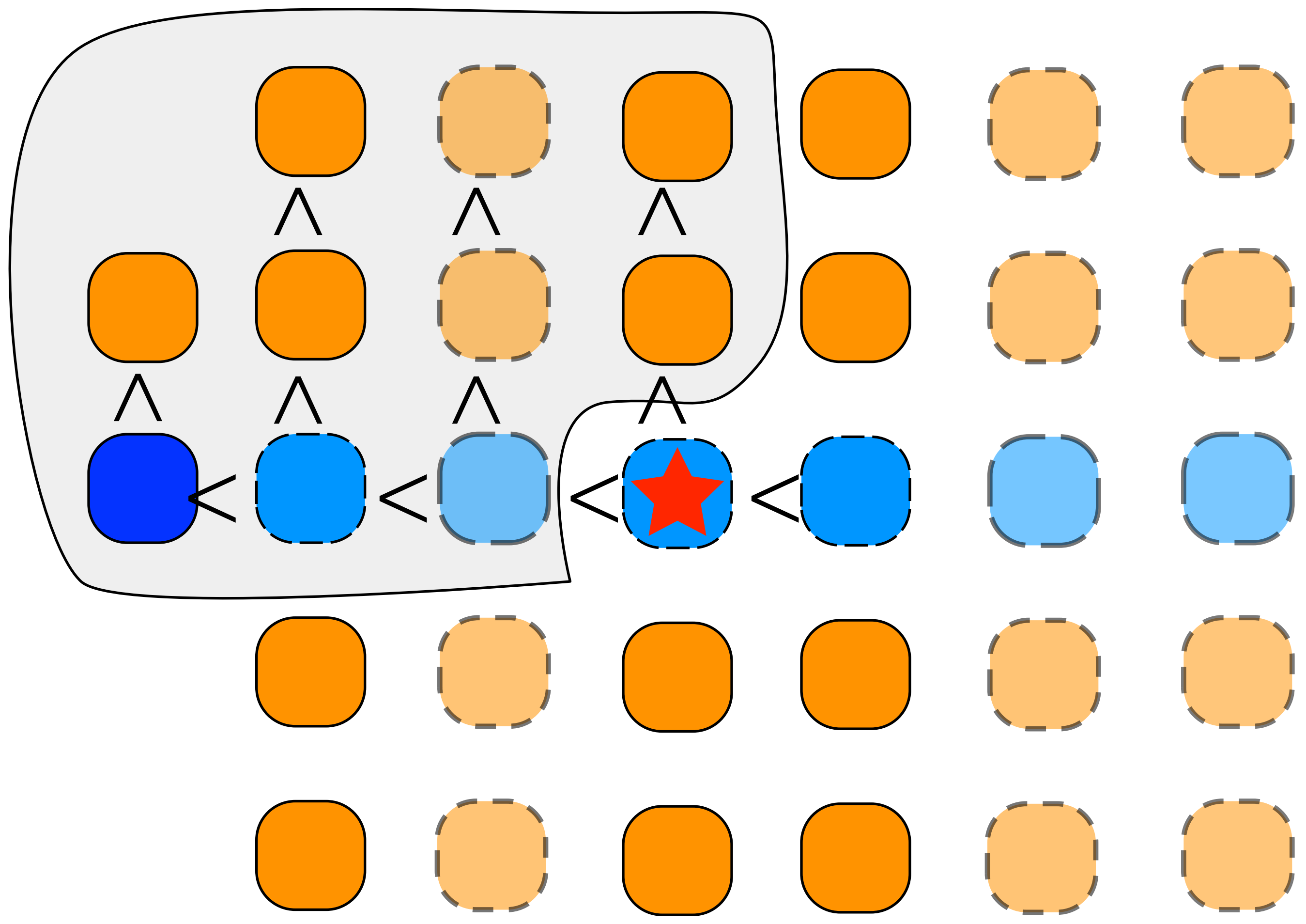


our partitioner p is larger than all of these elements

every other column has 3 elements that are smaller than p.

might have fewer elements

a nice property of our partition

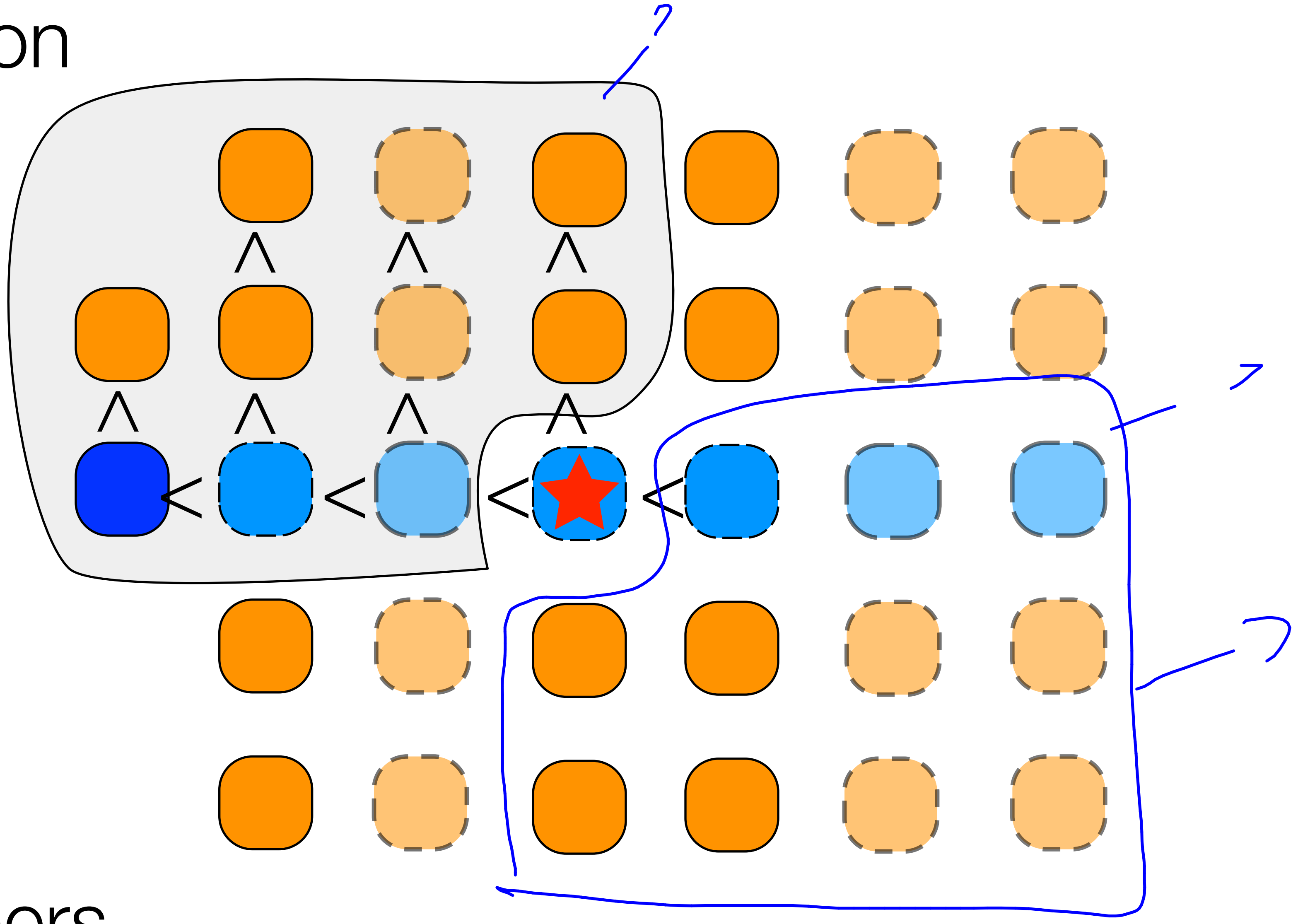


a nice property of our partition

$$3\left(\left\lceil \frac{1}{2}\lceil n/5\rceil \right\rceil - 2\right)$$

$$\geq \frac{3n}{10} - 6$$

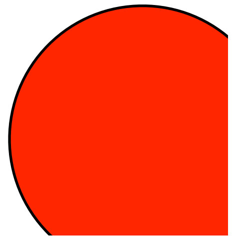
~30 %



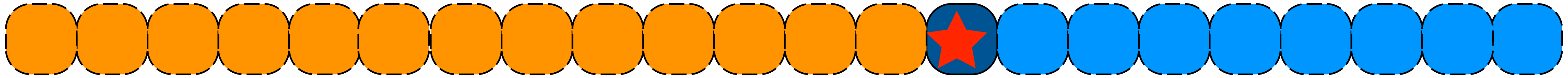
this implies there are

at most $\underline{\frac{7n}{10}} + 6$ numbers

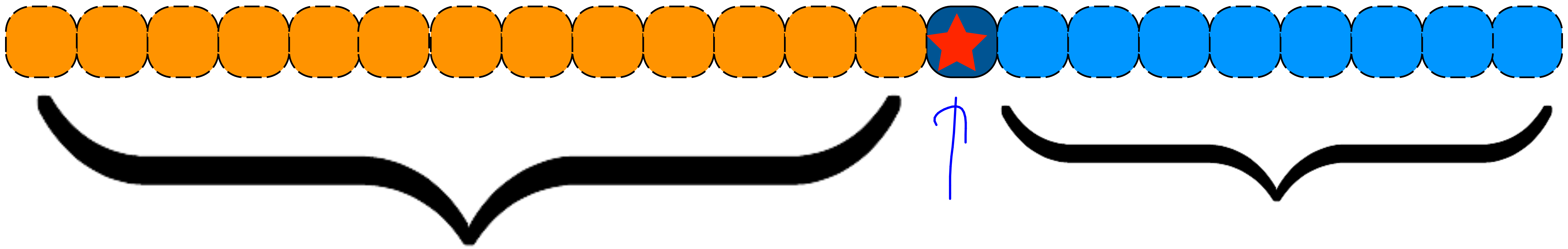
larger than ★
/smaller



a nice property of our partition

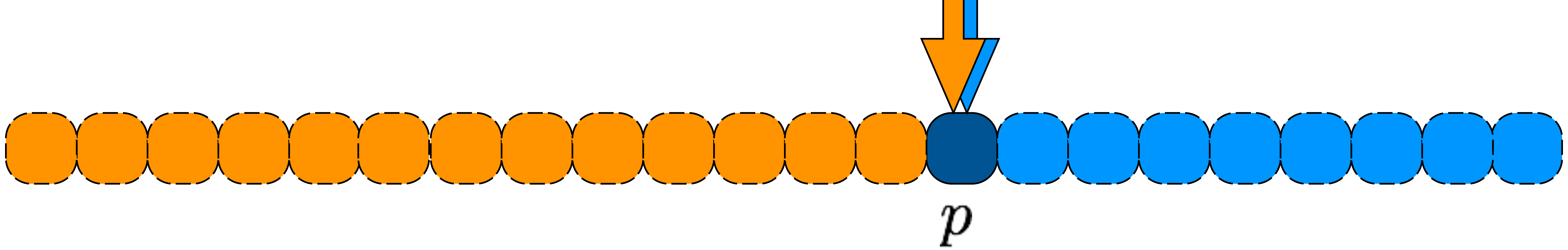


p_j

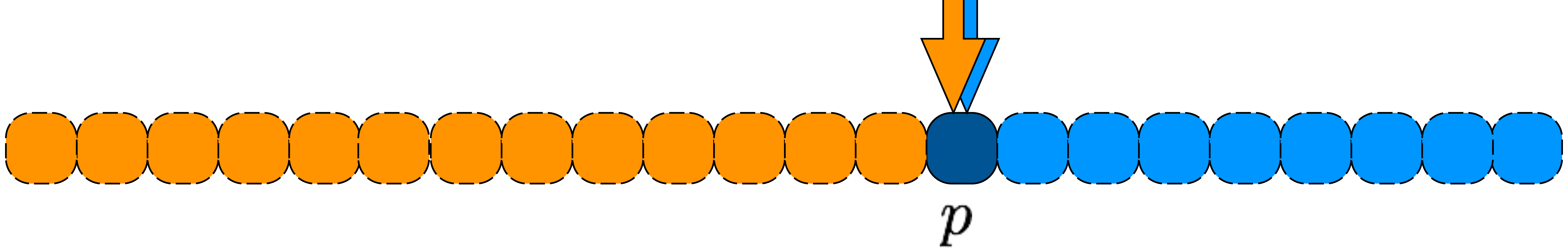


$$\le \frac{7n}{10} + 6 \qquad\qquad \le \frac{7n}{10} + 6$$

this is the result of partition.



select $(i, A[1, \ldots, n])$



select $(i, A[1, \ldots, n])$

handle base case for small list

else pivot = FindPartitionValue(A,n) $\longrightarrow$ $P(n) = S(\frac{n}{5}) + \Theta(n)$

partition list about pivot $\longrightarrow$ $\Theta(n)$

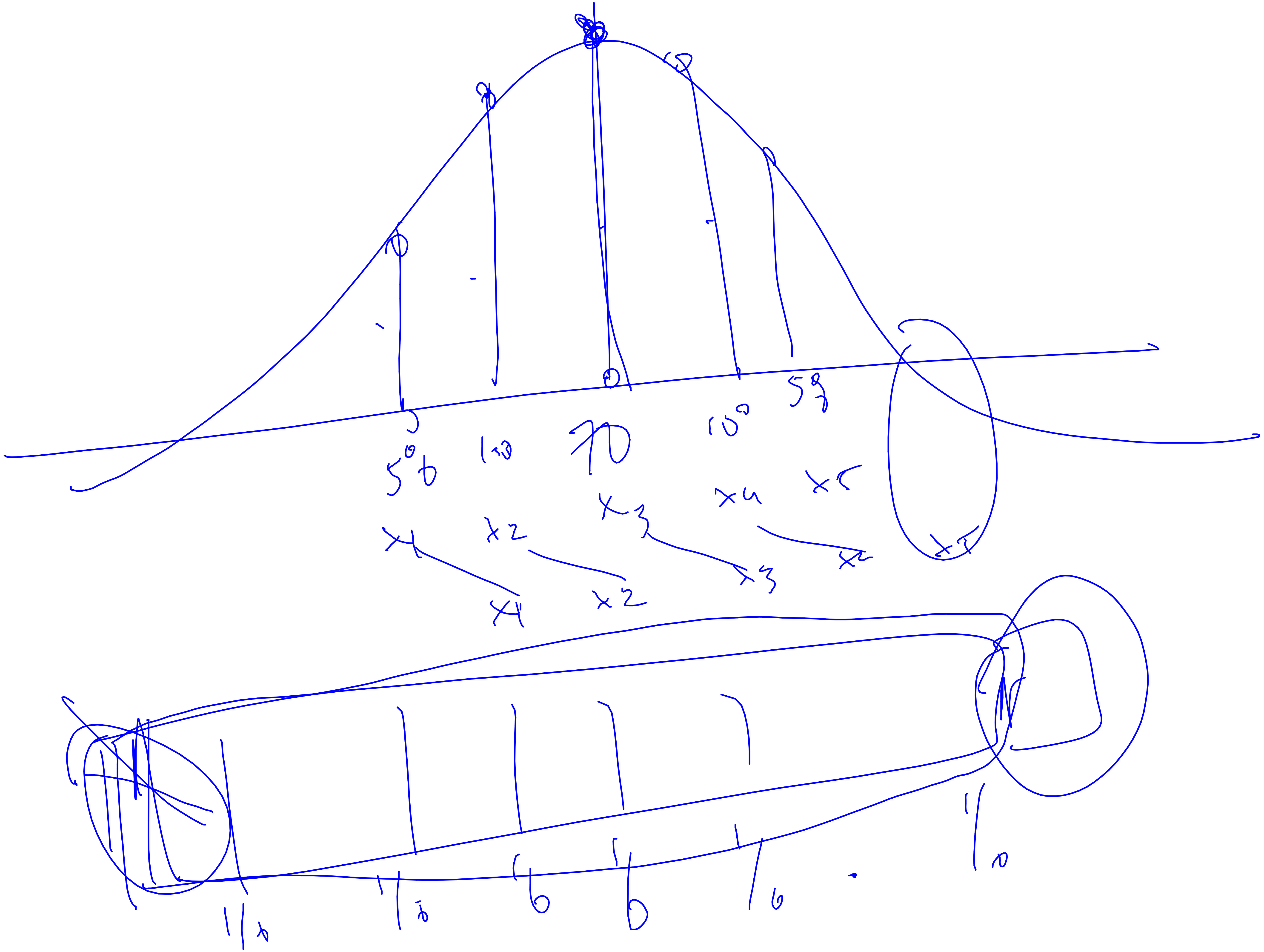
if pivot is position **i**, return pivot

else if pivot is in position > i select $(i, A[1, \ldots, p-1])$ $S(\frac{7n}{10}+6)$

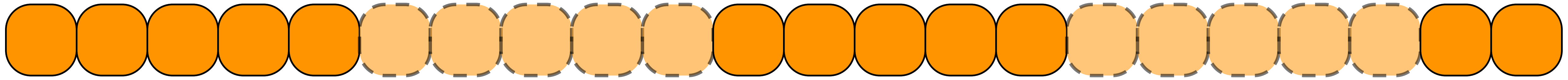
else select $((i-p-1), A[p+1, \ldots, n])$

$$S(n) = S(\lceil \tfrac{n}{5} \rceil) + S(\tfrac{7n}{10}+6) + \Theta(n) = \Theta(n)$$

$\frac{n}{5}+1$


x1
x2
x3
x4

FindPartition $(A[1,\ldots,n])$



divide list into groups of 5 elements

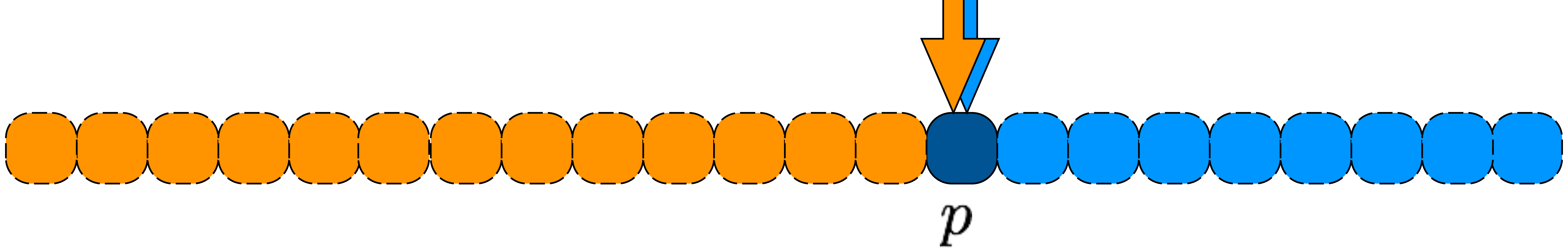
find median of each small list

gather all medians

call select(...) on this sublist to find median

return the result

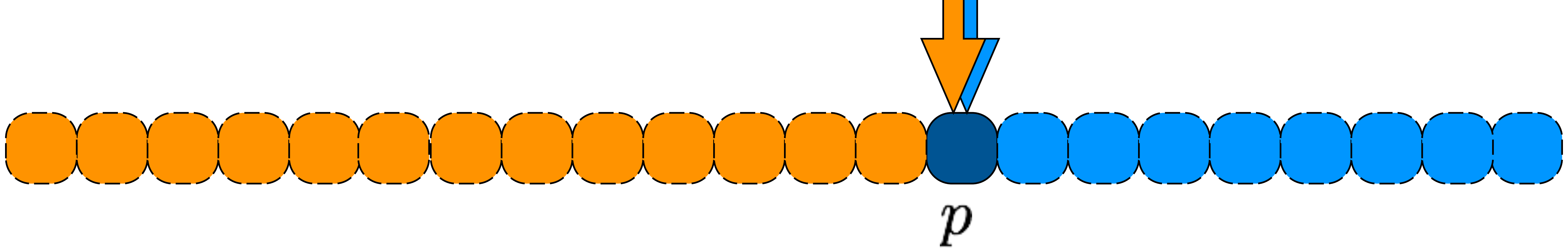
$$P(n) = S(\lceil n/5 \rceil) + O(n)$$



select $(i, A[1, \ldots, n])$

handle base case for small list
else pivot = FindPartitionValue(A,n)
partition list about pivot
if pivot is position **i**, return pivot
else if pivot is in position > **i** select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

$$S(n) = S(\lceil n/5 \rceil) + O(n) + S(7n/10 + 6)$$

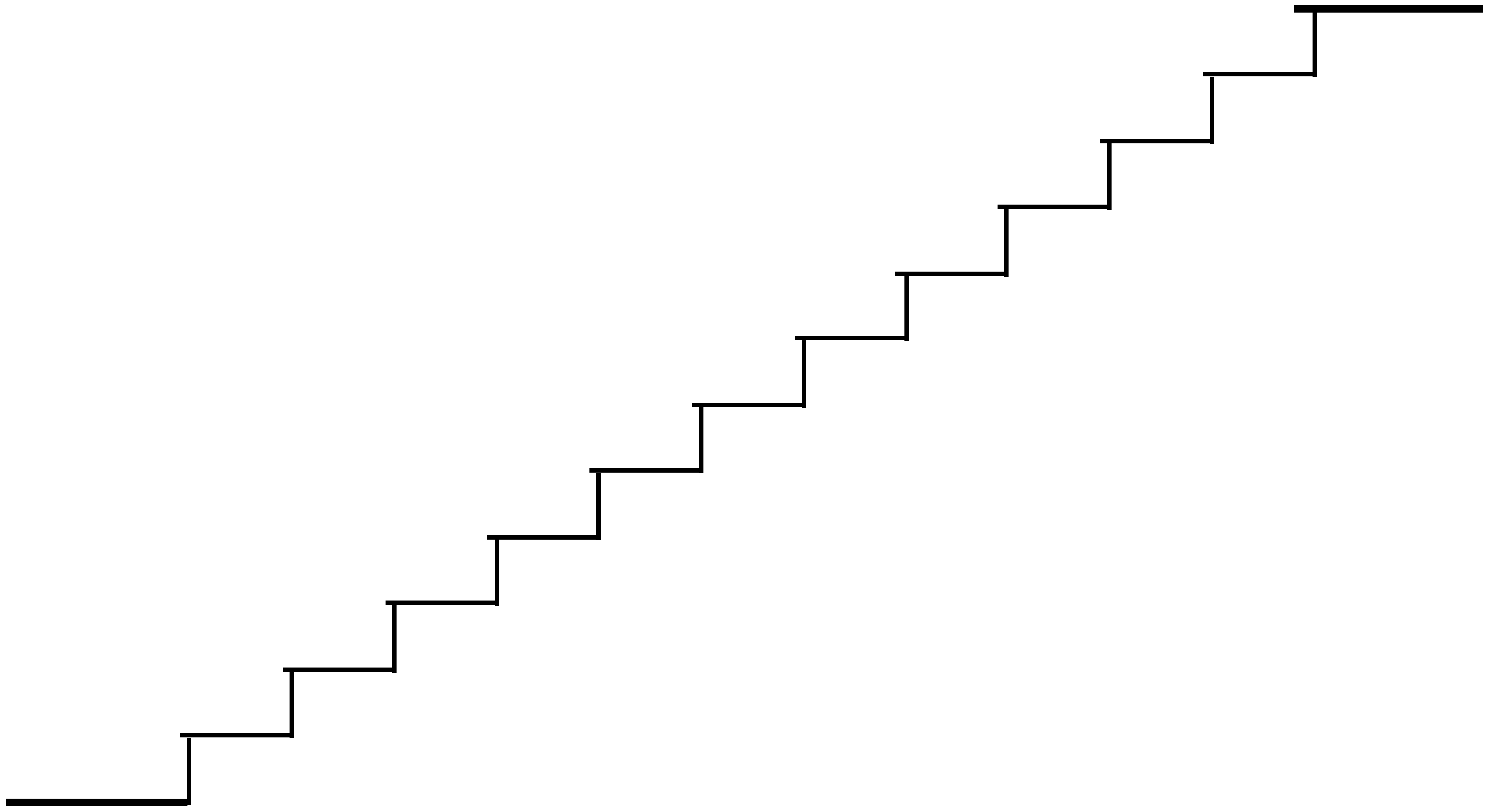


select $(i, A[1, \ldots, n])$

handle base case for small list
else pivot = FindPartitionValue(A,n)
partition list about pivot
if pivot is position **i**, return pivot
else if pivot is in position > **i** select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

$$S(n) = S(\lceil n/5 \rceil) + O(n) + S(7n/10 + 6)$$

$$\Theta(n)$$



```
Stairs(n)
    if n<=1 return 1
    return Stairs(n-1) + Stairs(n-2)
```

Stairs(n) if n<=1 return 1
ret Stairs(n-1) + Stairs(n-2)

Stairs(5)

Stairs(4) Stairs(3)

Stairs(3) Stairs(2) Stairs(2) Stairs(1)

Stairs(2) Stairs(1) Stairs(1) Stairs(0) Stairs(1) Stairs(0)

```
initialize memory M

Stairs(n)
  if n<=1 then return 1
  if n is in M, return M[n]
  answer = Stairs(i−1)+ Stairs(i−2)
  M[n] = answer
  return answer
```

```
Stairs(n)

    if n<=1 then return 1
    if n is in M, return M[n]
    answer = Stairs(i-1)+ Stairs(i-2)
    M[n] = answer
    return answer
```

Stairs(5)

Stairs(n)

```
stair[0]=1
stair[1]=1

for i=2 to n
  stair[i] = stair[i-1]+stair[i-2]

return stair[i]
```

```
    initialize memory M

Stairs(n)
```

```
Stairs(n)

    if n<=1 then return 1
    if n is in M, return M[n]
    answer = Stairs(i-1)+ Stairs(i-2)
    M[n] = answer
    return answer
```

Stairs(5)

Stairs(n)

```
stair[0]=1
stair[1]=1

for i=2 to n
  stair[i] = stair[i-1]+stair[i-2]

return stair[i]
```

Dynamic Programming

two big ideas

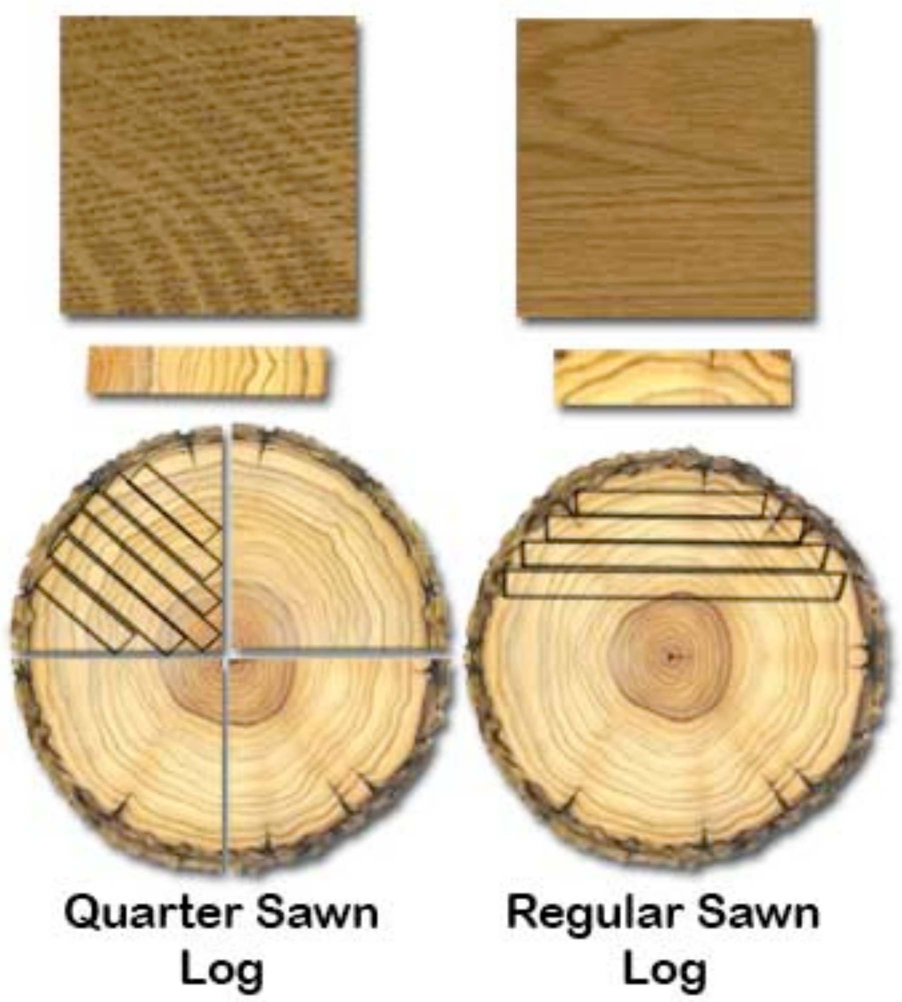
two big ideas

recursive structure

+

memoizing

wood cutting



http://www.amishhandcraftedheirlooms.com/quarter-sawn-oak.htm

http://snlm.files.wordpress.com/2008/08/bill-wakefield-and-carl-fie.gif

Spot price for lumber

Spot price for lumber

1” 2” 3” 4” 5” 6” 7” 8”

Log cutter dilemna

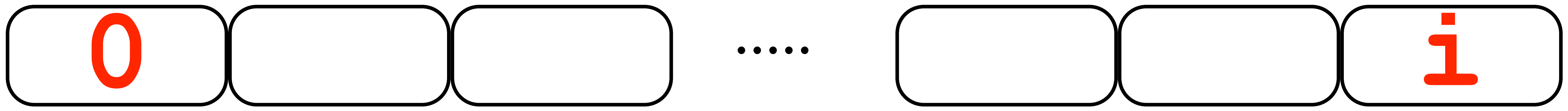
input to the problem: $n, (p_1, \ldots, p_n)$

goal:

Observation

Solution equation

Approach



BestLogs($n, (p_1, \ldots, p_n)$)

```
    if n<=0 return 0
```

BestLogs($n, (p_1, \ldots, p_n)$)

```
if n<=0 return 0
for i=1 to n
 Best[i] = max_{k=1...i} {p_k + Best[i-k]}
```

$$\text{Best}[i] = \max_{k=1\ldots i}\{p_k + \text{Best}[i-k]\}$$

The actual cuts?

BestLogs($n, (p_1, \ldots, p_n)$)

```
if n<=0 return 0
for i=1 to n
 Best[i] = 
```

$$\max_{k=1\ldots i}\{p_k + \mathrm{Best}[i-k]\}$$