CSE 421

Divide and Conquer / Multiplication & Median

Yin Tat Lee

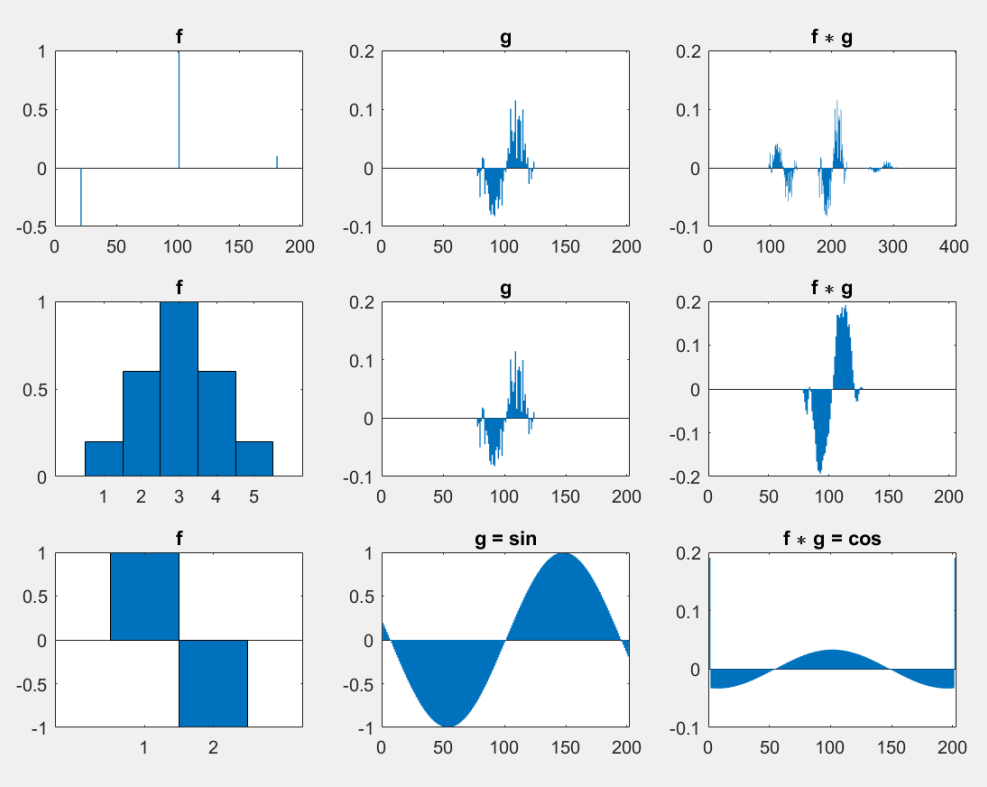
Integer Multiplication

Matlab code is included in the pptx.
For this class, you just need to remember
FFT and convolution can be done in O(n log n) time

Convolution

Recall that

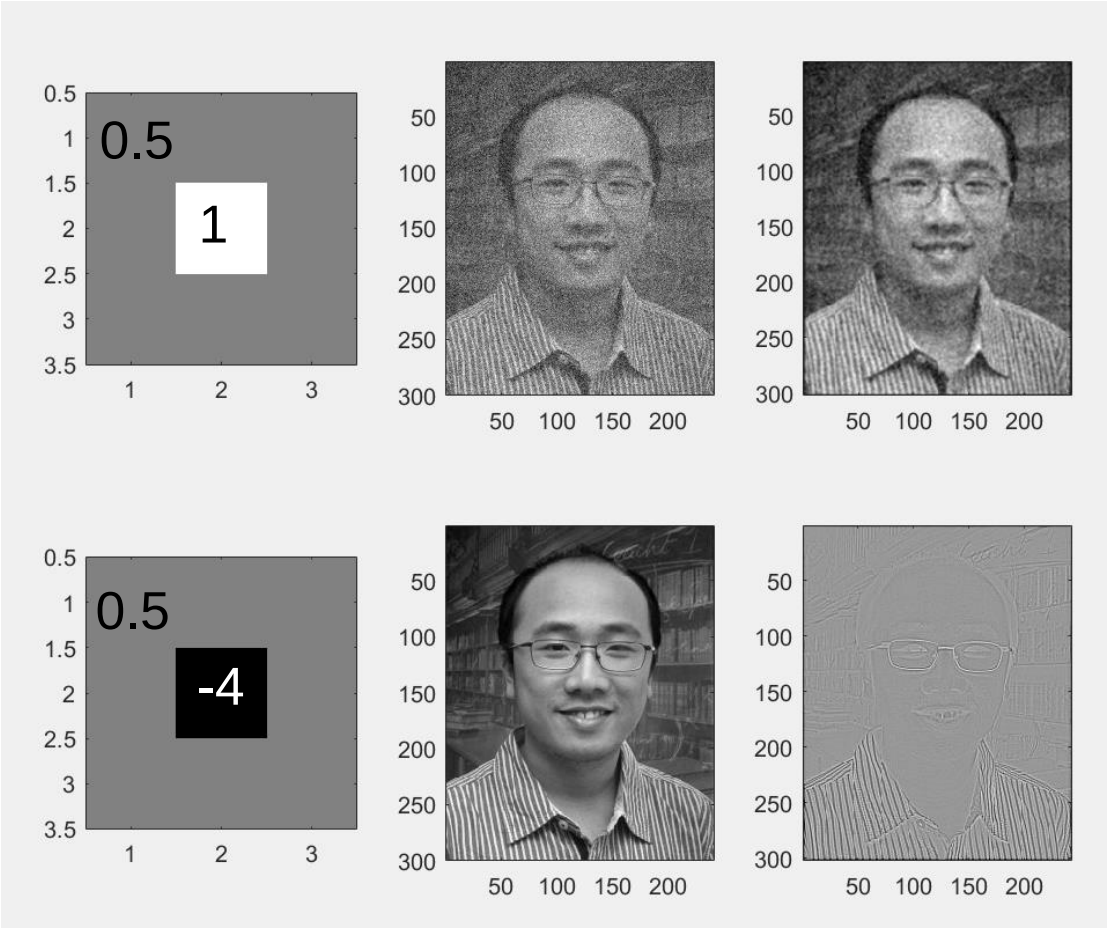
$$(f * g)_t = \sum_{k=-\infty}^{\infty} f_k g_{t-k} .$$



Question: Why we can solve integer multiplication using convolution?

2-D Convolution (just for fun)

$$(f * g)_{t_1,t_2} = \sum_{k_1,k_2=-\infty}^{\infty} f_{k_1,k_2} g_{t_1-k_1,t_2-k_2} \cdot$$

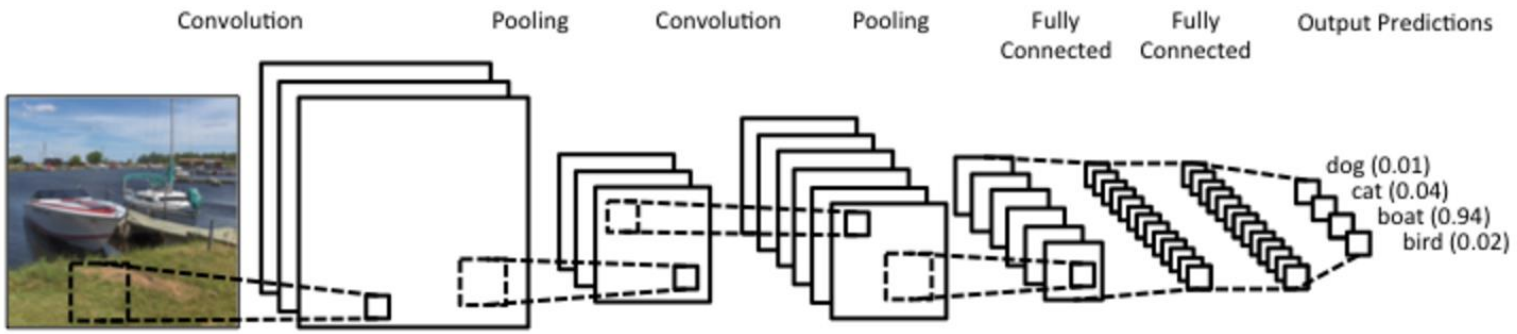


Convolution neural network (just for fun)

Convolution is a way to extract information from data.

But how can we design what kernel to use?

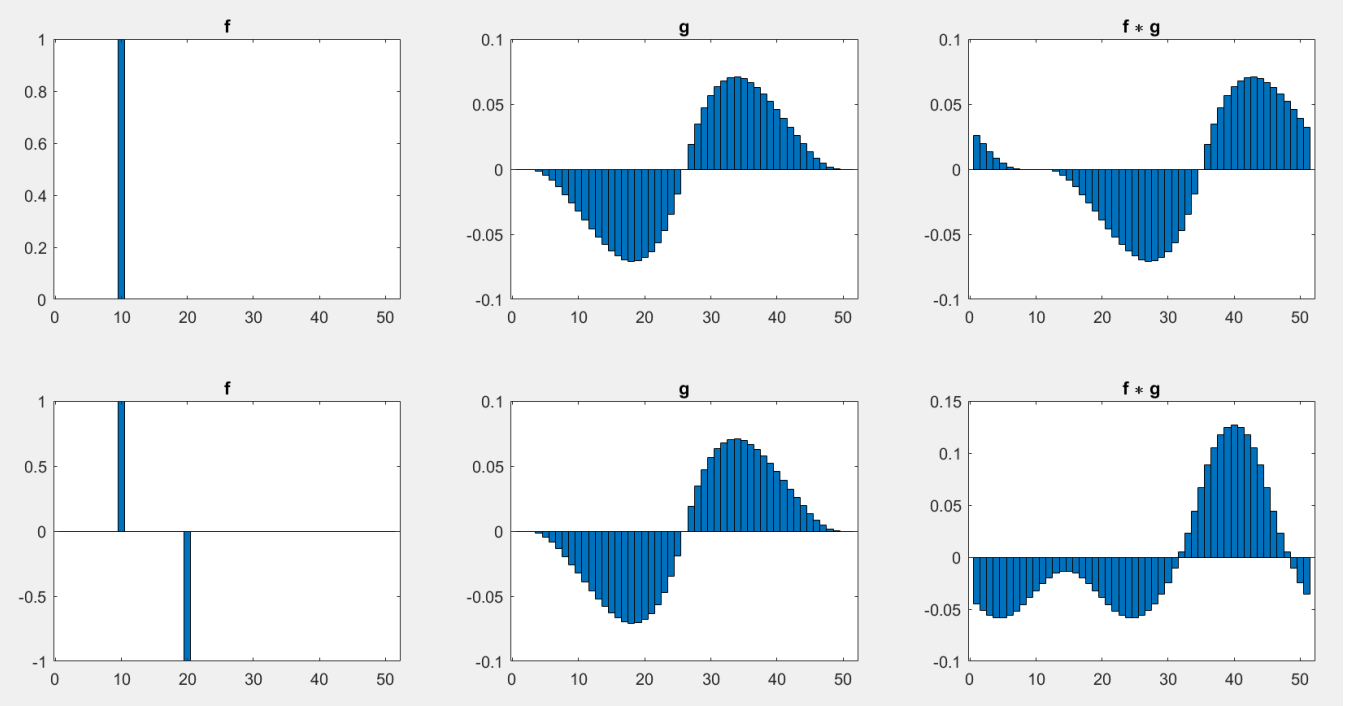
Optimization…



Circular Convolution

Given $f_0, f_1, \cdots, f_{N-1}; g_0, g_1, \cdots, g_{N-1}$

$$(f * g)_t = \sum_{k=0}^{N-1} f_k g_{t-k \bmod N}.$$



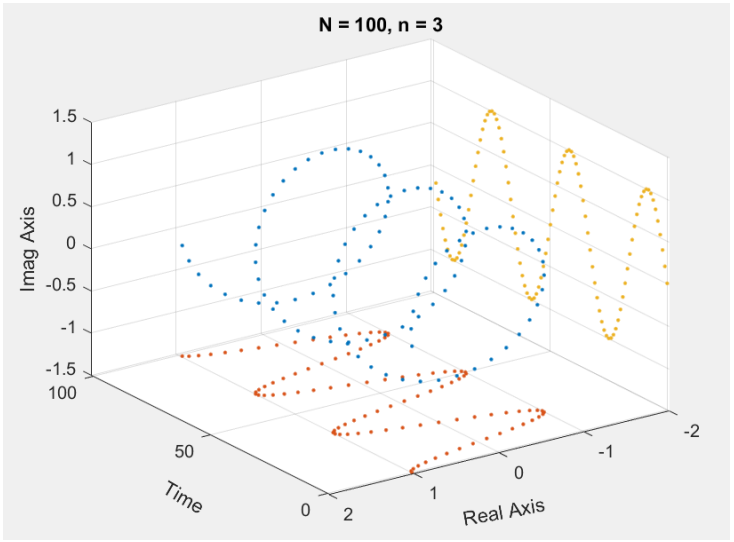
Question: If we can compute circular convolution in time $O(n \log n)$, then we can calculate convolution in time $O(n \log n)$.

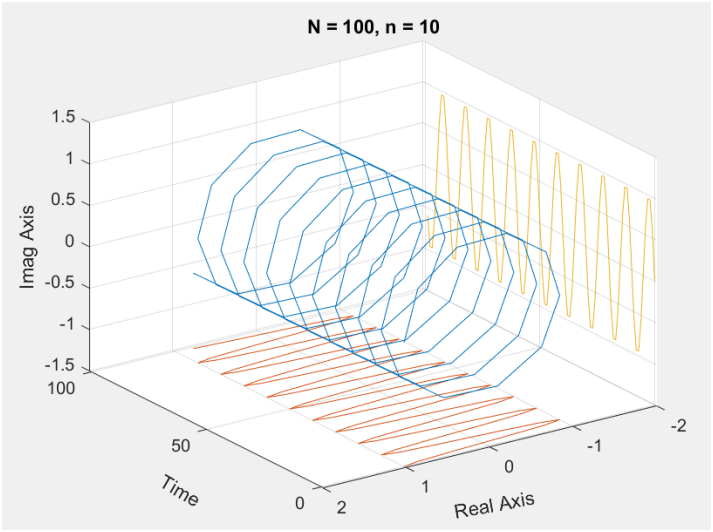
Discrete Fourier Transform

Given $x_0, x_1, \cdots, x_{N-1} \in \mathbb{C}$;

$$\mathcal{F}(x)_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N}kn} = \sum_{n=0}^{N-1} x_n \cdot \left[\cos(\frac{2\pi kN}{n}) - i\sin(\frac{2\pi kN}{n})\right]$$

What is $e^{-\frac{2\pi i}{N}kn}$ looks like for fixed n and N?





Fourier Transform is just the sum of these spirals.

DFT is invertible!

Given $x_0, x_1, \cdots, x_{N-1} \in \mathbb{C}$;

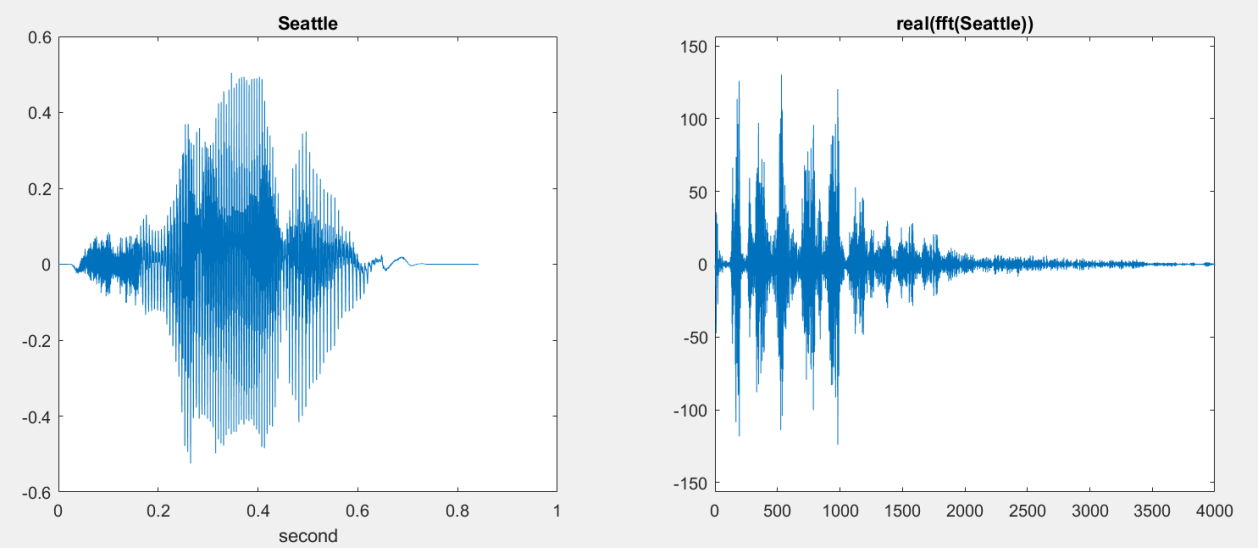
$$\mathcal{F}(x)_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} kn}.$$

This maps is invertible and hence

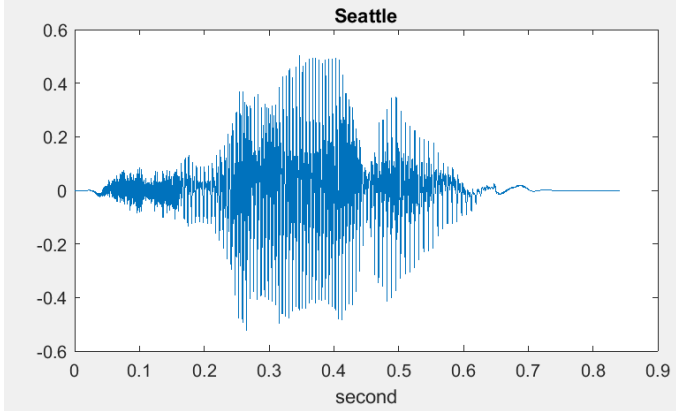
$$x_k = \frac{1}{N} \sum_{n=0}^{N-1} \mathcal{F}(x)_n \cdot e^{\frac{2\pi i}{N} kn}.$$

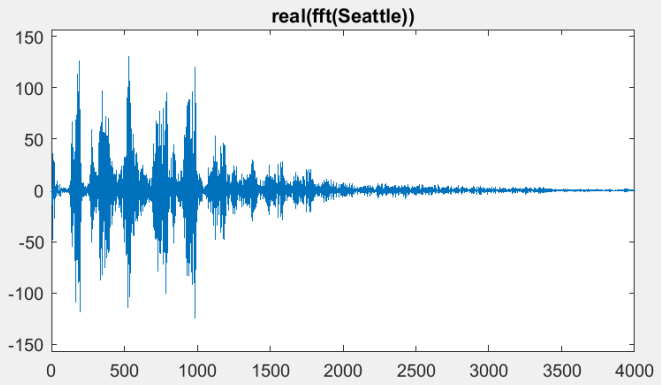
This is saying all complex signals can be represented by spirals!

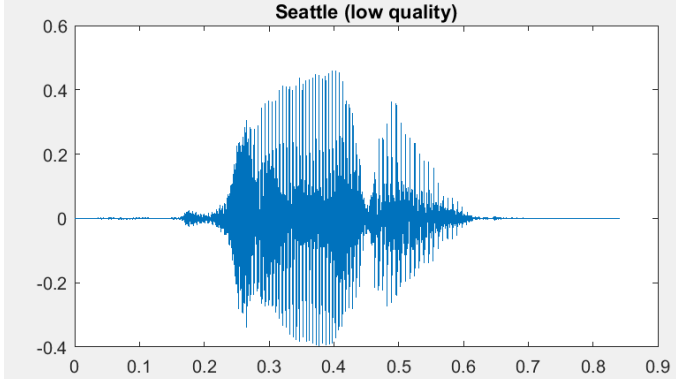
So, you can think x is signal, $\mathcal{F}(x)$ is the frequency (or opposite).

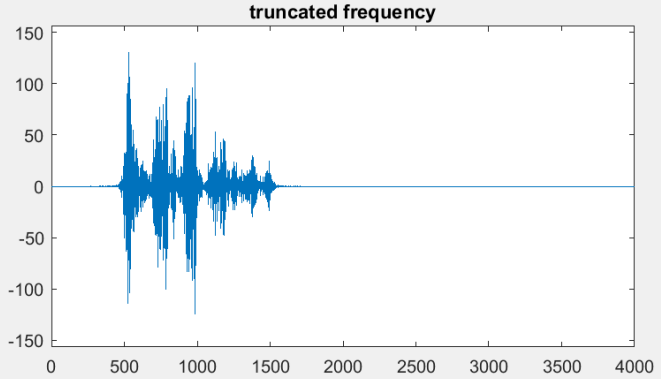


(just for fun)


Seattle
second


real(fft(Seattle))


Seattle (low quality)


truncated frequency

Convolution Theorem

Given $x_0, x_1, \cdots, x_{N-1} \in \mathbb{C}$;

$$\mathcal{F}(x)_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} kn}.$$

Theorem: $\mathcal{F}(x) \cdot \mathcal{F}(y) = \mathcal{F}(x * y)$.

Corollary: $x * y = \mathcal{F}^{-1}(\mathcal{F}(x) \cdot \mathcal{F}(y))$.

To compute convolution,
It suffices to compute DFT

Proof:

Let e_i be a vector that is 1 at index i and 0 otherwises.

Note that $x = \sum x_i e_i$ and $y = \sum y_j e_j$, then

Then, we have

$$\mathcal{F}(x) \cdot \mathcal{F}(y) = \sum x_i \mathcal{F}(e_i) \cdot \sum y_j \mathcal{F}\left(e_j\right) = \sum x_i y_j \mathcal{F}(e_i) \cdot \mathcal{F}\left(e_j\right)$$

Exercise: $\mathcal{F}(e_i) \cdot \mathcal{F}\left(e_j\right) = \mathcal{F}\left(e_i * e_j\right)$ for all i, j,

Hence, $\mathcal{F}(x) \cdot \mathcal{F}(y) = \sum x_i y_j \mathcal{F}\left(e_i * e_j\right) = \mathcal{F}\left(\sum x_i y_j e_i * e_j\right) = \mathcal{F}(x * y)$.

Fast Fourier Transform

Given $x_0, x_1, \cdots, x_{N-1} \in \mathbb{C}$;

$$\mathcal{F}(x)_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} kn}.$$

For simplicity, we assume N is a power of 2. Note that

$$\begin{aligned}
\mathcal{F}_N(x)_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} \cdot kn} \\
&= \sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-\frac{2\pi i}{N} \cdot k(2n)} + e^{-\frac{2\pi i}{N} \cdot k} \cdot \sum_{n=0}^{N/2-1} x_{2n+1} \cdot e^{-\frac{2\pi i}{N} \cdot k(2n)} \\
&= \mathcal{F}_{N/2}(x_{\text{even}})_k + e^{-\frac{2\pi i}{N} \cdot k} \cdot \mathcal{F}_{N/2}(x_{\text{odd}})_k.
\end{aligned}$$

Note that the vector $\mathcal{F}_{N/2}(x)$ has period $\frac{N}{2}$, namely,

$$\begin{aligned}
\mathcal{F}_{N/2}(x)_{k+\frac{N}{2}} &= \sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-\frac{2\pi i}{N/2} \cdot (k+\frac{N}{2})n} \\
&= e^{-\frac{2\pi i}{N/2} \cdot \frac{N}{2} n} \cdot \sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-\frac{2\pi i}{N/2} \cdot (kn)} \\
&= e^{-2\pi i n} \cdot \sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-\frac{2\pi i}{N/2} \cdot (kn)} \\
&= \sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-\frac{2\pi i}{N/2} \cdot (kn)}
\end{aligned}$$

where we used that $e^{-2\pi i} = 1$. Hence, we have that

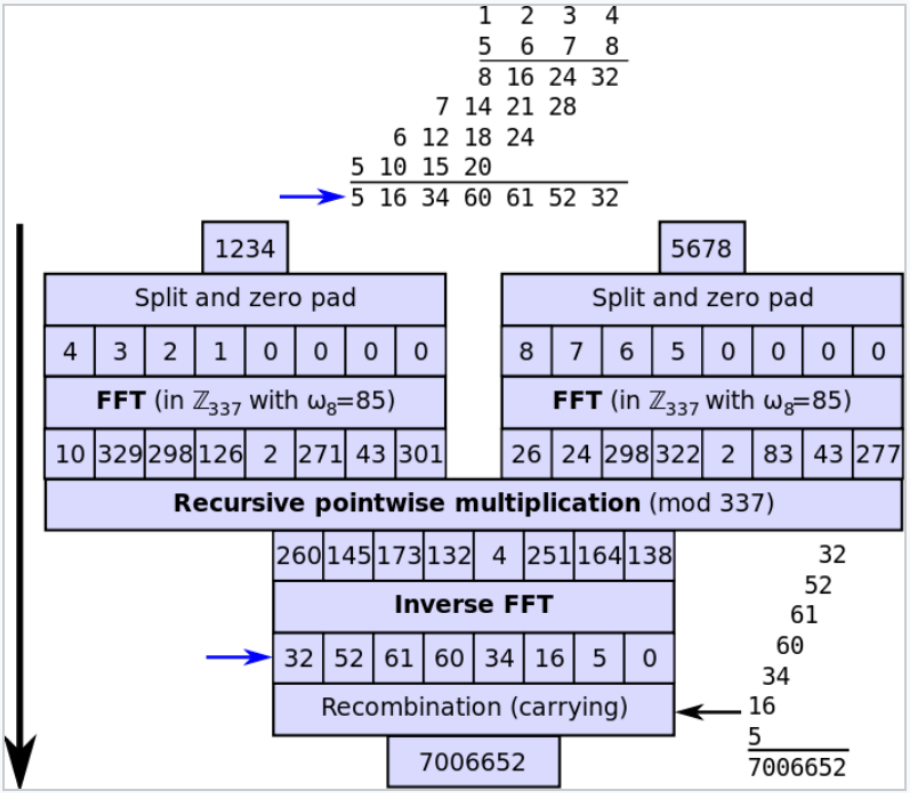
$$\mathcal{F}_N(x)_k = \begin{cases} \mathcal{F}_{N/2}(x_{\text{even}})_k + e^{-\frac{2\pi i}{N} \cdot k} \cdot \mathcal{F}_{N/2}(x_{\text{odd}})_k & \text{for } 0 \le k < \frac{N}{2} \\ \mathcal{F}_{N/2}(x_{\text{even}})_{k-\frac{N}{2}} + e^{-\frac{2\pi i}{N} \cdot k} \cdot \mathcal{F}_{N/2}(x_{\text{odd}})_{k-\frac{N}{2}} & \text{for } \frac{N}{2} \le k < N \end{cases}.$$

Using this formula, the cost of compute $\mathcal{F}_N$ satisfies the relation

$$\mathcal{T}(N) = 2\mathcal{T}(\frac{N}{2}) + O(N).$$

Therefore, the cost is $O(N \log N)$.

Integer Multiplication



Demonstration of multiplying 1234 × 5678 = 7006652 using fast Fourier transforms (FFTs). Number-theoretic transforms in the integers modulo 337 are used, selecting 85 as an 8th root of unity. Base 10 is used in place of base 2^w for illustrative purposes.

Median

Selecting k-th smallest

Problem: Given numbers $x_1, \ldots, x_n$ and an integer $1 \leq k \leq n$ output the k-th smallest number

$$\mathrm{Sel}(\{x_1, \ldots, x_n\}, k)$$

A simple algorithm: Sort the numbers in time $O(n \log n)$ then return the k-th smallest in the array.

Can we do better?

Yes, in time $O(n)$ if $k = 1$ or $k = 2$.

Can we do $O(n)$ for all possible values of k?

An Idea

Choose a number w from $x_1, \dots, x_n$

Define

- $S_<(w) = \{x_i : x_i < w\}$
- $S_=(w) = \{x_i : x_i = w\}$
- $S_>(w) = \{x_i : x_i > w\}$

Can be computed in linear time

Solve the problem recursively as follows:

- If $k \le |S_<(w)|$, output $Sel(S_<(w), k)$
- Else if $k \le |S_<(w)| + |S_=(w)|$, output w
- Else output $Sel(S_>(w), k - |S_<(w)| - |S_=(w)|)$

Ideally want $|S_<(w)|, |S_>(w)| \le n/2$. In this case ALG runs in $O(n) + O\left(\frac{n}{2}\right) + O\left(\frac{n}{4}\right) + \cdots + O(1) = O(n)$.

How to choose w?

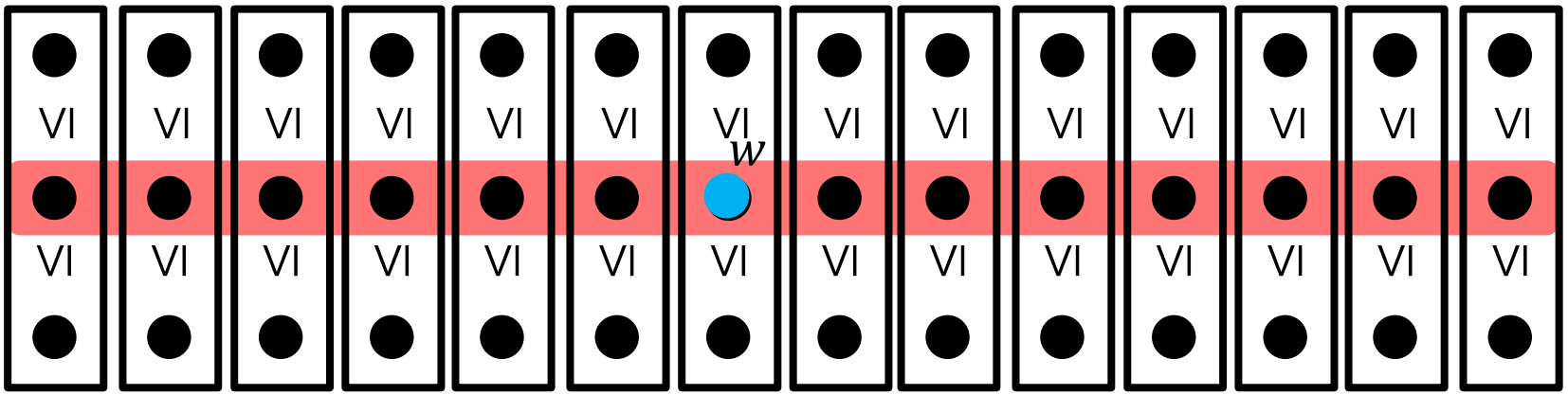
Suppose we choose w uniformly at random

similar to the pivot in quicksort.

Then, $\mathbb{E}[|S_<(w)|] = \mathbb{E}[|S_>(w)|] = n/2$. Algorithm runs in $O(n)$ in expectation.

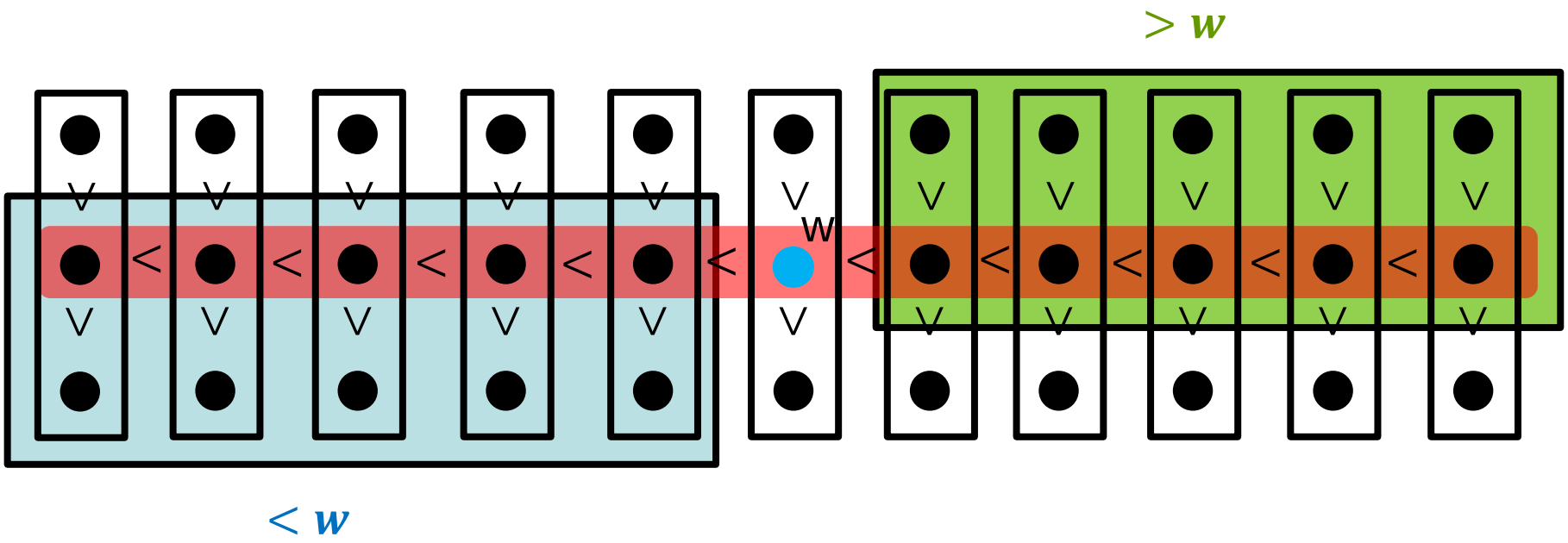
Can we get $O(n)$ running time deterministically?

- Partition numbers into sets of size 3.
- Sort each set (takes O(n))
- $w = Sel(midpoints, n/6)$



Assume all numbers are distinct for simplicity.

How to lower bound $|S_<(w)|, |S_>(w)|$?



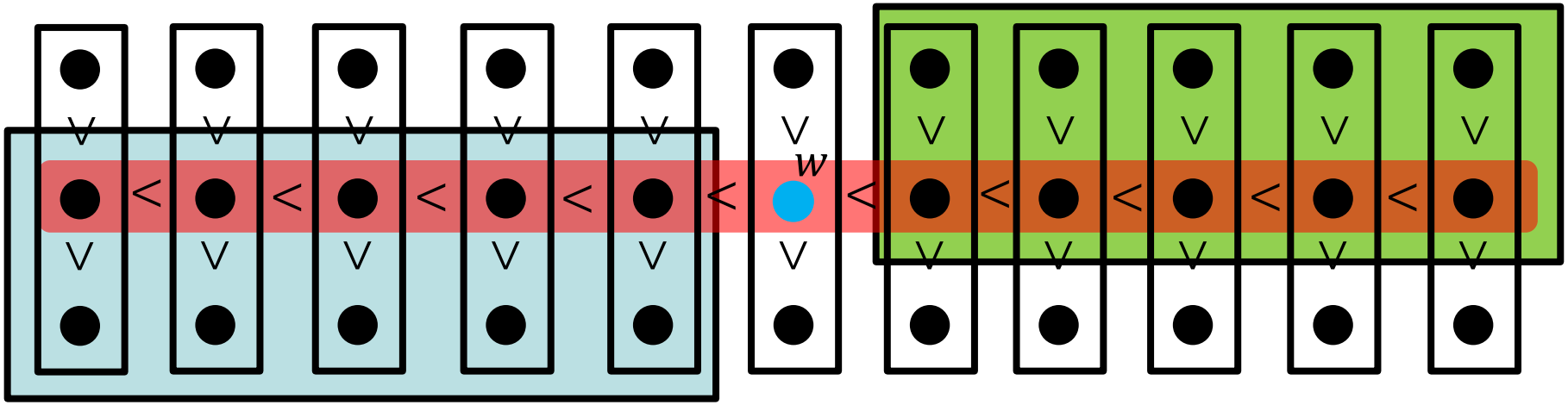
- $|S_<(w)| \geq 2\left(\frac{n}{6}\right) = \frac{n}{3}$
- $|S_>(w)| \geq 2\left(\frac{n}{6}\right) = \frac{n}{3}$.

$$\frac{n}{3} \leq |S_<(w)|, |S_>(w)| \leq \frac{2n}{3}$$

So, what is the running time?

Assume all numbers are distinct for simplicity.

Asymptotic Running Time?



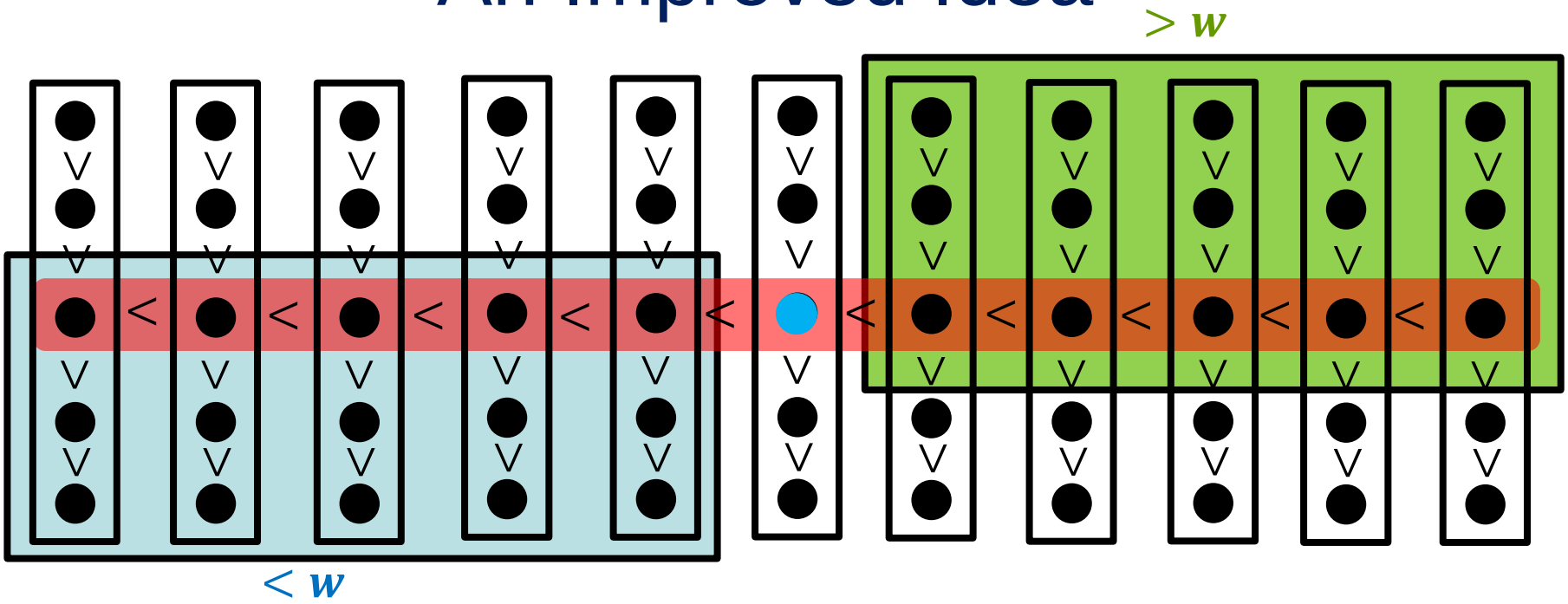
- If $k \leq |S_<(w)|$, output $Sel(S_<(w), k)$
- Else if $k \leq |S_<(w)| + |S_=(w)|$, output w
- Else output $Sel(S_>(w), k - |S_<(w)| - |S_=(w)|)$

$O(n \log n)$ again?
So, what is the point?

Where $\frac{n}{3} \leq |S_<(w)|, |S_>(w)| \leq \frac{2n}{3}$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n) \Rightarrow T(n) = O(n \log n)$$

An Improved Idea



Partition into n/5 sets. Sort each set and set $w = Sel(midpoints, n/10)$

- $|S_<(w)| \geq 3\left(\frac{n}{10}\right) = \frac{3n}{10}$
- $|S_>(w)| \geq 3\left(\frac{n}{10}\right) = \frac{3n}{10}$

$$\frac{3n}{10} \leq |S_<(w)|, |S_>(w)| \leq \frac{7n}{10}$$

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n) \Rightarrow T(n) = O(n)$$

An Improved Idea

```
Sel(S, k) {
   n ← |S|
   If (n < ??) return ??
   Partition S into n/5 sets of size 5
   Sort each set of size 5 and let M be the set of medians, so |M|=n/5
   Let w=Sel(M,n/10)
   For i=1 to n{
     If x_i < w add x to S_<(w)
     If x_i > w add x to S_>(w)
     If x_i = w add x to S_=(w)
   }
   If (k ≤ |S_<(w)|)
     return Sel(S_<(w), k)
   else if (k ≤ |S_<(w)| + |S_=(w)|)
     return w;
   else
     return Sel(S_>(w), k − |S_<(w)| − |S_=(w)|)
}
```

We can maintain each set in an array