Inverse FFT:

Linear-algebra view of the FFT algorithm:

FFT: Coefficients vector a (input) $\longrightarrow$ Values vector A (output)

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \vdots & & \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} A(1) \\ A(\omega_n) \\ A(\omega_n^2) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix}$$

$$\underbrace{}_{M_n(\omega_n)} \quad \underbrace{}_{a} \quad \underbrace{}_{A}$$

$$A = M_n(\omega_n)\, a = FFT(a, \omega_n)$$

Now we want to do the inverse: A (input) $\longrightarrow$ a (output)

if $M_n(\omega_n)^{-1}$ exists then $a = M_n(\omega_n)^{-1} A$

<u>Lemma:</u> $M_n(\omega_n)^{-1} = \frac{1}{n} M_n(\omega_n^{-1})$

As we saw before, $\omega_n^{-1} = \omega_n^{n-1}$ (since $\omega_n \times \omega_n^{n-1} = 1$)

Therefore, $a = \frac{1}{n} M_n(\omega_n^{-1}) A = \frac{1}{n} FFT(A, \omega_n^{n-1})$

To prove the lemma we need the following basic fact:

Claim: For any ω which is a n^{th} root of unity & $\omega \neq 1$, then:

$$1+\omega+\omega^2+\cdots+\omega^{n-1}=0$$

Proof: For any number z notice that

$$(z-1)(1+z+z^2+\cdots+z^{n-1})=z^n-1$$

Plug in $z=\omega$ and use that $\omega^n=1$ then:

$$(\omega-1)\underbrace{(1+\omega+\omega^2+\cdots+\omega^{n-1})}=0$$

So either this $=0$ or $=0$

We know $\omega\neq 1$ so $1+\omega+\omega^2+\cdots+\omega^{n-1}=0$. ∎

Now let's prove the lemma.

We need to show that:

$$\frac{1}{n}M_n(\omega_n)M_n(\omega_n^{n-1})=I \text{ where } I=\begin{bmatrix}1 & & 0\\ & \ddots & \\ 0 & & 1\end{bmatrix}$$

Look at entry (k,k) for $M_n(\omega_n)\times M_n(\omega_n^{n-1})$ for $k=0,\dots,n-1$:

$$=(1,\omega_n^k,\omega_n^{2k},\dots,\omega_n^{(n-1)k})\cdot(1,\omega_n^{(n-1)k},\omega_n^{2(n-1)k},\dots,\omega_n^{(n-1)(n-1)k})$$

$$=1+1+1+\cdots+1$$

$$=n \quad ✓$$

For $k \neq j$, look at entry (k,j) of $M_n(\omega_n) \times M_n(\omega_n^{n-1})$:

$$= \left(1, \omega_n^{k}, \omega_n^{2k}, \ldots, \omega_n^{(n-1)k}\right) \cdot \left(1, \omega_n^{(n-1)j}, \omega_n^{2(n-1)j}, \ldots, \omega_n^{(n-1)(n-1)j}\right)$$

$$= 1 + \omega_n^{k-j} + \omega_n^{2(k-j)} + \cdots + \omega_n^{(n-1)(k-j)}$$

let $\omega = \omega_n^{k-j}$ & note that $\omega \neq 1$ since $k \neq j$ & $0 \leq k, j \leq n-1$

$$= 1 + \omega + \omega^2 + \cdots + \omega^{n-1}$$

$= 0$ from our earlier claim. ∎

This finishes off FFT.

Dynamic Programming:

Example 1: LIS = longest increasing subsequence.

Given n numbers $a_1, \ldots, a_n$,
find the length of the LIS.

Example: 5, 7, 4, -3, 9, 1, 4, 8, 6, 7, 5

LIS = 5 from -3, 1, 4, 6, 7

First step, define subproblem in words,
then define recurrence. If can't find a
recurrence then probably get an idea how to revise

Attempt 1:

Let $T(i)$ = length of LIS in $a_1, \ldots, a_i$

What's the recurrence?

For the above example, $T(6) = 3$ from 5, 7, 9,
but we want -3, 1 so that $T(7) = 3$ from -3, 1, 4.

Then for $T(8)$ can we add 8 on?
Yes if it's -3, 1, 4, no if its 5, 7, 9,
how do we know which?
Keep track of all possible endings, so
try the following.

Attempt 2:

Let $T(i)$ = length of LIS in $a_1, \dots, a_i$ which includes a_i

Now we know the end so we know if we can add to it.

Hence, $T(i) = 1 + \max\limits_{1 \leq j < i} \{T(j) : a_j < a_i\}$

Algorithm:

LIS($a_1, \dots, a_n$)

for $i = 1 \to n$
  $T(i) = 1$
  for $j = 1 \to i-1$
   if $a_j < a_i$ then $T(i) = \max\{T(i), 1 + T(j)\}$

max = 1
for $i = 2 \to n$
  if $T(i) > T(\text{max})$ then max = i
Return($T(\text{max})$)

Running time: $O(n^2)$

Knapsack:

n objects with integer weights $w_1, \dots, w_n$
& integer values $v_1, \dots, v_n$

total capacity B

What's subset S of objects where

$$\sum_{i \in S} w_i \leq B$$

& which maximizes $\sum_{i \in} v_i$

Version 1: one copy of each object.

Attempt 1: Let $T(i)$ = max value attainable using subset of objects $1, \dots, i$

But then for $T(i)$ can we add object i to optimal solution for $T(i-1)$? May want suboptimal solution for $T(i-1)$ which has enough capacity so that can add object i.
So want to see optimal solution for given capacity.

Attempt 2: Let $T(i,b)$ = max value attainable using subset of $1, \dots, i$ & total capacity $\leq b$.

Recurrence: if $w_i \leq b$

Don't use i — best of $1,\dots,i-1$ with remaining weight — Use i

then $T(i,b) = \max\{T(i-1,b), T(i-1,b-w_i)+v_i\}$

else $T(i,b) = T(i-1,b)$

Final answer: $T(n,B)$

Running time: $O(nB)$

Version 2: unlimited supply of each object.

Now we don't need to keep track of what objects are used so far.

Let $T(b)$ = max value attainable using total capacity $\leq b$.

For the recurrence, try all possibilities for the last object to add.

$$T(b) = \max_{1 \leq i \leq n} \{T(b-w_i)+v_i : w_i \leq b\}$$

Final result: $T(B)$

Running time: $O(nB)$

Knapsack has running time $O(nB)$.

Is this a polynomial-time algorithm?

No, the number B is part of the input and it's input size is $O(\log B)$.

We'll see later that knapsack is NP-complete.

We'll reduce from a 3-SAT instance with n variables & m constraints, and the knapsack instance will have B = exponentially large in n & m.