Exam 1

- DP
  - Fib(n) $\Rightarrow O(n)$

```
F[0] = 0
F[1] = 1
for i = 2 -> n:
    F[i] = F[i-1] + F[i-2]
return F[n]
```

  - LIS$(a_1, \ldots, a_n)$ $\Rightarrow O(n^2)$

```
for i = 1 -> n:
    L[i] = 1
    for j = 1 -> i-1:
        if a_j < a_i & L[i] < 1 + L[j]:
            L[i] = 1 + L[j]
max = 1
for i = 2 -> n:
    if L[i] > L[max]: max = i
return L[max]
```

  - LCS(X, Y) $\Rightarrow O(n^2)$

```
for i = 0 -> n:
    L[i, 0] = 0
    L[0, i] = 0
for i = 1 -> n:
    for j = 1 -> n:
        if X_i = Y_j: L[i,j] = 1 + L[i-1, j-1]
        else: L[i,j] = max{L[i-1,j], L[i,j-1]}
return L[n,n]
```

- Knapsack No Repeat$(w, v, B) \Rightarrow O(nB)$

```
for b = 0 → B : K[0,b] = 0
for i = 1 → n : K[i,0] = 0
for i = 1 → n :
    for b = 1 → B :
        if wi ≤ b : K[i,b] = max{vi + K[i-1, b-wi], K[i-1,b]}
        else : K[i,b] = K[i-1,b]
return K[n,B]
```

- Knapsack Repeat$(w, v, B) \Rightarrow O(nB)$

```
for b = 0 → B :
    K[b] = 0
    for i = 1 → n :
        if wi ≤ b & K[b] < vi + K[b-wi] :
            K[b] = vi + K[b-wi]
return K[B]
```

- RA
  - Mod Arithmetic
    - $x \equiv y \bmod n \Rightarrow \frac{x}{n}$ & $\frac{y}{n}$ have same rem.
    - $x \bmod n = r$ if $x = qN + r$ for ints q, r
    - if $x \equiv y \bmod n$ and $a \equiv b \bmod n$
      - $x + a \equiv y + b \bmod N$
      - $xa \equiv yb \bmod N$
  - Modular Exponentiation $\Rightarrow O(n^2 2^n)$

    $x \bmod N = a_1$

    $x^2 \equiv a_1 x \bmod N = a_2$

    $x^2 \equiv a_2 x \bmod N = a_3$ ... etc
  - Repeated Squaring

    $x \bmod N = a_1$

    $x^2 \equiv (a_1)^2 \bmod N = a_2$

    $x^4 \equiv (a_2)^2 \bmod N = a_4$

    $x^8 \equiv (a_4)^2 \bmod N = a_8$ ... etc

- Mod Exp(x, y, N)
  - input: n-bit ints $x, y, N \geq 0$
  - output: $x^y \bmod N$
  - if $y=0$: return 1
  - $z = $ ModExp$(x, \lfloor \frac{y}{2} \rfloor, N)$
  - if y is even: return $z^2 \bmod N$
  - else: return $x z^2 \bmod N$
- Mutiplicative Inverse
  - x is mul. inv. of $z \bmod n$ if $xz \equiv 1 \bmod n$
  - $x \equiv z^{-1} \bmod N$ and $z \equiv x^{-1} \bmod N$
  - $x^{-1} \bmod n$ exists iff $\gcd(x, n) = 1$
- Euclid's Rule
  - $\gcd(x, y) = \gcd(x \bmod y, y)$
- Euclid(x, y) $\Rightarrow O(n^3)$
  - input: ints x, y where $x \geq y \geq 0$
  - output: $\gcd(x, y)$
  - if $y = 0$: return x
  - else: return Euclid($y, x \bmod y$)
- Ext Euclid(x, y)
  - input: ints x, y where $x \geq y \geq 0$
  - output: ints d, α, β where $d = \gcd(x, y)$
    - $\alpha \equiv x^{-1} \bmod y$
    - $\beta \equiv y^{-1} \bmod x$
  - if $y=0$: return $(x, 1, 0)$
  - $(d, \alpha', \beta') = $ ExtEuclid($y, x \bmod y$)
  - return $(d, \beta', \alpha' - \lfloor \frac{x}{y} \rfloor \beta')$

- RSA
  - Fermat Little Theorem
    - if p is prime, for every $\underbrace{1 \le z \le p-1}_{gcd(z,p)=1}$
      $z^{p-1} \equiv 1 \mod p$
  - Euler's Theorem
    - for any N, z where $gcd(z, N) = 1$, $z^{\phi(N)} \equiv 1 \mod N$
      - $\phi(N)$ = num ints between 1 and N relatively prime to N
      - if $N = pq$, $\phi(N) = (p-1)(q-1)$
  - RSA
    1) Pick 2 n-bit random primes p and q, let $N = pq$
       - choose random n-bit number r
         if r is prime: return r
         else: repeat until found
    2) Pick e relatively Prime to $\phi(N)$
       - e should be small (3, 5, 7, 11, etc..)
    3) Publish public key (N, e)
    4) compute private key $d = e^{-1} \mod \phi(N)$
       - use Ext Euclid
    - encrypt message m: $y \equiv m^e \mod N$
    - decrypt message y: $m \equiv y^d \mod N$
      - use fast mod. exp.
- Primality
  - Fermat Witness
    - if r is prime, for all $z \in \{1, \ldots, r-1\}$ $z^{r-1} \equiv 1 \mod r$
    - $z^{r-1} \not\equiv 1 \mod r \Rightarrow$ r is composite
      z is fermat witness
    - every composite has fermat witness
  - Simple Test
    1) choose $z_1, \ldots, z_k$ randomly from $\{1, \ldots, r-1\}$
    2) for $i = 1 \to k$: compute $z_i^{r-1} \equiv 1 \mod r$
    3) if for ALL i, $z_i^{r-1} \equiv 1 \mod r$: output TRUE
       else: output FALSE

- Prime $r \Rightarrow Pr$(alg. outputs TRUE) $= 1$
- Composite $r \Rightarrow Pr$(alg. outputs TRUE) $\leq (\frac{1}{2})^k$

- Divide and Conquer
  - fast modular exponentiation
  - Euclid Alg.
  - Multiply n-bit integers better than $O(n^2)$

    FastMultiply$(x, y) \Longrightarrow O(n^{\log_2 3})$

    input: n-bit ints x, y where $n = 2^k$

    output: xy

    $X_L = 1^{st}$ $\frac{n}{2}$ bits of x ; X_R = last $\frac{n}{2}$ bits of X

    $Y_L = 1^{st}$ $\frac{n}{2}$ bits of y ; Y_R = last $\frac{n}{2}$ bits of Y

    $A =$ FastMultiply(X_L, Y_L)

    $B =$ FastMultiply(X_R, Y_R)

    $C =$ FastMultiply$(X_L + X_R, Y_L + Y_R)$

    return $2^n A + 2^{\frac{n}{2}}(C - A - B) + B$

  - FastSelect$(A, k) \Longrightarrow O(n)$
    1) Break A into $\frac{n}{5}$ groups of 5 elems each
    2) For $i = 1 \rightarrow \frac{n}{5}$: Sort G_i and let $m_i = \text{Median}(G_i)$
    3) Let $S = \{m_1, \ldots, m_{\frac{n}{5}}\}$
    4) $p =$ FastSelect$(S, \frac{n}{10})$
    5) partition A into $A < p$, $A = p$, $A > p$
    6) recurse on $A < p$ or $A > p$ or output p

  - Recurrences
    - $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$
    - $T(n) = 4T(\frac{n}{2}) + O(n) = O(n^2)$
    - $T(n) = 3T(\frac{n}{2}) + O(n) = O(n^{\log_2 3})$
    - $T(n) = T(\frac{3}{4}n) + O(n) = O(n)$

- FFT
  - Multiply polynomials
  - given $a=(a_0,\dots,a_{n-1})$ and $b=(b_0,\dots,b_{n-1})$
    compute $c=a*b=(c_0,\dots,c_{2n-2})$
  - FFT converts coefficients to values
  - roots of unity
    - $(n^{th}\text{ roots})^2 = (\frac{n}{2})$ roots (ie $\omega_{16}^2=\omega_8$)
  - $FFT(a,\omega) \Rightarrow O(n\log n)$
    if $n=1$: return a_0
    Let $a_{even}=(a_0,a_2,\dots,a_{n-2})$; $a_{odd}=(a_1,a_3,\dots,a_{n-1})$
    $(s_0,s_1,\dots,s_{\frac{n}{2}-1}) = FFT(a_{even},\omega^2)$
    $(t_0,t_1,\dots,t_{\frac{n}{2}-1}) = FFT(a_{odd},\omega^2)$
    for $j=0\to\frac{n}{2}-1$:
    $r_j = s_j + \omega^j t_j$
    $r_{\frac{n}{2}+j} = s_j - \omega^j t_j$
    return $(r_0,r_1,\dots,r_{n-1})$
- Inverse FFT goes values to coefficients
  - $a=\frac{1}{n}FFT(A,\omega_n^{n-1})$
- Multiplying Polynomials (a,b)
  $(r_0,r_1,\dots,r_{2n-1}) = FFT(a,\omega_{2n})$
  $(s_0,s_1,\dots,s_{2n-1}) = FFT(b,\omega_{2n})$
  for $j=0\to 2n-1$: $t_j=r_js_j$
  run IFFT on t to get c

• ex: $A(x) = 1 + x + 2x^2$; $B(x) = 2 + 3x$

$a = [1, 1, 2, 0]$

$$\begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 4 \\ i-1 \\ 2 \\ -i-1 \end{bmatrix} \quad \text{FFT}(a)$$

$$\begin{bmatrix} 2 \\ 3 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 5 \\ 2+3i \\ -1 \\ 2-3i \end{bmatrix} \quad \text{FFT}(b)$$

$$\begin{bmatrix} 4 \\ i-1 \\ 2 \\ -i-1 \end{bmatrix} \begin{bmatrix} 5 \\ 2+3i \\ -1 \\ 2-3i \end{bmatrix} = \begin{bmatrix} 20 \\ 2i-3-2-3i \\ -2 \\ -2i-3-2+3i \end{bmatrix} = \begin{bmatrix} 20 \\ -i-5 \\ -2 \\ i-5 \end{bmatrix}$$

$$\frac{1}{4} \begin{bmatrix} 20 \\ -i-5 \\ -2 \\ i-5 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 8 \\ 20 \\ 28 \\ 24 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 7 \\ 6 \end{bmatrix}$$

$C(x) = 2 + 5x + 7x^2 + 6x^3$