SORTING ALGORITHMS

Find k^{th} smallest element:

```
fxn select(A[0..n-1],k)
 if n=1, return A[0]
 pick some v=A[i]
 A< : entries < v, length n<
 A= : entries = v, length n=
 A> : entries > v, length n>
 if k < n<, select(A<,k)
 if n< <= k < n>, return v
 else select(A>,k-n<-n=)
```

PARALLEL COMPUTING

Essentially, you have O(work done * num of sequential levels + work done * num of parallelized levels). If you have $p \geq n$, then you do $\frac{f(n)}{p}$ work at each part at $\log p$ levels. If $p < n$, then you have $\frac{f(n)}{p}$ work at $\log p$ parallelized levels + $\frac{f(n)}{p}$ work done at $\log \frac{n}{p}$ sequential levels.

RECURRENCES

Given $f(x), g(x)$: $f = O(g)$ if $\frac{f}{g} \leq$ constant (has upper bound). This means that $g = \Omega(f)$. $f = \Theta(g)$ means that $f = O(g)$ and $g = O(f)$ (f and g differ by constant factor). n^a dominates n^b if $a > b$. Any exponential dominates any polynomial $3^n > n^5$. Any polynomial dominates any log ($n > (\log n)^3, n^2 > n \log n$). $f_1 = O(g_1)$ and $f_2 = O(g_2) \rightarrow f_1 + f_2 = O(g_1 + g_2)$ (same with multiplication). $\log(n) = O(n)$. if $\frac{f(n)}{g(n)} \rightarrow$ finite nonzero constant as $n \rightarrow \infty$, then $f = \Theta(g)$. If $\frac{f(n)}{g(n)} \rightarrow 0$ as $n \rightarrow \infty$, then $f = O(g)$ but not $\Omega(g)$. If you have confusing exponents, try taking the log of both sides.

$$T(n) = \begin{cases} \Theta(f(n)) & \text{if } f(n) = \Omega(n^d) \quad d > \log_b a \\ \Theta(f(n)\log n) & \text{if } f(n) = \Theta(n^d) \quad d = \log_b a \\ \Theta(n^{\log_b a}) & \text{if } f(n) = O(n^d) \quad d < \log_b a \end{cases}$$

Compare $f(n)$ to $n^{\log_b a}$. This lets you know which one of these cases you want to choose d to fit.

SUMMATIONS

$$\sum_{i=0}^{n} = \frac{n(n+1)}{2} \quad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \quad \sum_{k=0}^{n-1} ar^k = a\frac{1-r^n}{1-r} \quad \sum_{i=1}^{\log n} n = n \log n \quad \sum_{i=1}^{n} \frac{1}{2^i} = 1 - \frac{1}{2^n}$$

FFT

When doing FFT, you evaluate your polynomial at the n^{th} roots of unity up to the closest power of 2: $\{1, i, -1, -i\}$ for F_4. Do do the *inverse* FFT, evaluate it at the complex conjugate of that same vector: $\{1, -i, -1, i\}$ for F_4. We do $y = Fx$ to change polynomial from coefficients x to values y.

```
fxn FFT(A,w):
in: coefficient representation of polynomial
  A(x) of degree <= n-1, w=nth root of unity
out: value representation A(w^0)..A(w^n-1)
if w=1, return A(1)
express A(x) in form A_e(x^2) + xA_O(x^2)
FFT(A_e,w^2) to evaluate A_e at even powers of w
FFT(A_o,w^2) to evaluate A_o at even powers of w
for j=0..n-1:
 compute A(w^j) = A_e(w^2j)+w^j*A_o(w^2j)
return A(w^0)...A(w^n-1)
```

FFT example on $P(x) = 2x^0 + 3x^1 + 4x^2 + 5x^3$. Rewrite as even and odd terms: $P(x) = (2x^0 + 4x^2) + x(3x^0 + 5x^2)$. To make a recursive call here, we need to have $P(x)$ with degree< 2, because we're moving from F_4 to F_2. We divide the exponents of $P_{\text{even,odd}}$ by 2, then plug in x^2 to preserve the polynomial. $P(x) = P_{\text{even}}(x^2) + xP_{\text{odd}}(x^2)$. Evaluate at the 2^{th} roots of unity $1, -1$, then we can evaluate $P(1, i, -1, -i)$ by using the fact that each of those squared is either 1 or -1.

GRAPHS

To find source, run DFS and the vertex w/ largest post[v] is in a source SCC. To find sink, do the same process on same graph w/ edges reversed. The source in G_R will be a sink in G.