CS170

- Sum of two n-bit integers is at most $n+1$ bit.
  Optimal addition: $O(n)$.
  Multiplication: $O(n^2)$ for both regular and modular multiplication.
  Modular division: $O(n^3)$.
  Modular exponential: $O(n^3)$.

- Euclid's rule: If x, y are positive integers with $x \geq y$, then $\gcd(x, y) = \gcd(x \mod y, y)$. Euclid's algorithm: input size shrinks by 1 in each iteration.
  $O(n^3)$. Within $2n$ recursive calls, each involves $O(n^2)$ modular division.

- a has a unique multiplicative inverse in modulo N if and only if a, N are coprime.The inverse can be found in $O(n^3)$ by extended Euclid algorithm.

- Fermat's little theorem: If p is prime, then for every $1 \leq a \leq p$, $a^{p-1} \equiv 1(\mod p)$.
  Randomly pick k positive integers $a_i < N$. If all a_i pass Fermat's tests, returns Yes. If any of a_i fails, returns No.
  Ignoring Carmichael numbers, $P[\text{Yes}|N \text{ is not prime}] \leq \frac{1}{2^k}$.

- Multiplication by divide and conquer: $O(n^{\log_2 3})$.
  Matrix multiplication by divide and conquer: $O(n^{\log_2 7})$.

- Polynomial multiplication: evaluation $\to$ multiplication $\to$ interpolation.
  n-th root of unity: $\omega = e^{\frac{2\pi i}{n}}$

```
function FFT(A, w)
    A: coefficient representation of a polynomial with degree <= n-1
    w: n-th root of unity

    if w = 1:
        return A(1)
    call FFT(A_e, w ** 2) and FFT(A_o, w ** 2)
    for j from 0 to n-1:
        A(w ** j) = A_e(w ** (2j)) + w ** j A_o(w ** (2j))

    return A(w ** 0), ..., A(w ** (n-1))
```

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

$$\langle \text{values} \rangle = FFT(\langle \text{coefficients} \rangle, \omega)$$

$$\langle \text{coefficients} \rangle = FFT(\langle \text{values} \rangle, \omega^{-1})$$

  Define $M_n(\omega)$ as $[M_n(\omega)]_{ij} = \omega^{ij}, \omega = \omega_n = e^{\frac{2\pi i}{n}}$. Then $M_n\omega^{-1} = \frac{1}{n}M_n(\omega^{-1})$.

- DFS, BFS: $O(V + E)$.

- A directed graph has a cycle if and only if its depth-first search reveals a back edge.

- DAG can be linearized. Proof: DAG has no cycle, thus no back edge. Since the only edge (u, v) in a graph for which post(u) <post(v) are back edges, then for any edge (u, v), post(u) $\geq$post(v). Thus the reverse of post order give topological sort.
  Acyclicity, linearizability, and the absence of back edges during a depth-first search are equivalent.

- Every directed graph is a dag of its strongly connected components.
  If the explore subroutine is started at node u, then it will terminate precisely when all nodes reachable from u have been visited.
  The node that receives the highest post number in a depth-first search must lie in a source strongly connected component.

- For each d, there is a moment at which (1) all nodes at distance $\leq d$ from s have their distances correctly set; (2) all other nodes have their distances set to ∞; and (3) the queue contains exactly the nodes at distance d.

- Dijkstra: At the end of each iteration of the while loop, the following conditions hold: (1) there is a value d such that all nodes in R are at distance $\leq d$ from s and all nodes outside R are at distance $\geq d$ from s, and (2) for every node u, the value $\text{dist}(u)$ is the length of the shortest path from s to u whose intermediate nodes are constrained to be in R (if no such path exists, the value is ∞).
  Running time with binary heap: $O((|V| + |E|) \log |V|)$.

- Bellman-Ford: $O(|V| \times |E|)$.

```
procedure update((u, v) in E):
    dist(v) = min{dist(v), dist(u) + l(u, v)}
```

- Kruskal: $|V|$ makeset, $2|E|$ find, and $|V| - 1$ union operations.
  With union rank, $O(|E| \log |V|)$.

- Huffman encoding: the cost of a tree is the sum of the frequencies of all leaves and internal nodes, except the root. $O(n \log n)$ with binary heap.

- Set cover:
  Suppose a set with n elements has an optimal cover consisting of k sets. Then the greedy algorithm will use at most $k \ln n$ sets.

- A list of predecessors in a graph is given by adjacency list of the reverse graph G^R.

- Knapsack: $O(nW)$.
  With repetition:
  $$K(w) = \max_{i:w_i \leq w} \{K(w - w_i) + v_i\}$$
  Without repetition:
  $$K(w, j) = \max\{K(w - w_j, j - 1) + v_j, K(w, j - 1)\}$$

- Matrix multiplication: $O(n^3)$.

- Floyd-Warshall algorithm: $O(|V|^3)$

```
for i = 1 to n:
    for j = 1 to n:
        dist(i, j, 0) = \infty
for (i, j) in E:
    dist(i, j, 0) = l(i, j)
# Outermost loop: expanding known region which has size k
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            # Relax all pairs with new intermediate node k
            dist(i, j, k) = min{dist(i, k, k-1) + dist(k, j, k-1), dist(i, j, k-1)}
```

- TSP: $O(n^2 2^n)$
  For a subset of cities $S \subset \{1, 2, \ldots, n\}$ such that $1, j \in S$, let $C(S, j)$ be the length of the shortest path visiting each node in S exactly once, starting at 1 and ending at j. The subproblems are ordered by $|S|$.

```
C({1}, 1) = 0
for s = 2 to n:
    for all subsets S of [n] with size s and containing 1:
        C(S, 1) = \infty
        for all j in S, j != 1:
            C(S, j) = min{C(S\{j}, i) + l(i, j) : i in S, i != j}
return min{C([n], j) + l(j, 1)}
```

- Independent sets in trees: $O(|V| + |E|)$

$$I(u) = \max\{1 + \sum_{\text{grandchildren}} I(w), \sum_{\text{children}} I(w)\}$$

- Maximum flow: $O(|E| \cdot \text{value of max flow})$ or $O(C|E|^2)$ where C is the maximum capacity of any single edge.
  With BFS: $O(|V| \times |E|)$ iterations, each iteration uses BFS in $O(|E|)$ to find s-t path. $O(|V| \times |E|^2)$ running time.
  The size of the maximum flow in a network equals the capacity of the smallest s-t cut.
  Let f be the final flow, L be the nodes that are reachable from s in residual graph G^f, $R = V - L$. Then (L, R) is a cut in the graph G and $\text{size}(f) = \text{capacity}(L, R)$.

- Search problem $\rightarrow$ optimization problem: use binary search to find optimal cost.

- Euler path: traverse edges exactly once.
  Solution exists iff (a) the graph is connected and (b) every vertex, with the possible exception of two vertices (start and final vertices) has even degree.

- Rudrata/Hamilton cycle: given a graph, find a cycle that visits each vertex exactly once—or report that no such cycle exists.

- Independent set: find g vertices, no two of which have an edge between them.

- Vertex cover: find b vertices that touch every edge.

- A search problem is specified by an algorithm $\mathcal{C}$. $\mathcal{C}(I, S)$ runs in polynomial time in $|I|$, the length of the instance, and outputs true iff S is a valid solution to instance I. Denote the class of all search problems by NP.
  The class of all search problems that can be solved in polynomial time is denoted P.
  A search problem is NP-complete if all other search problems reduce to it.
  NP-complete = NP-hard $\cap$ NP.

- Independent set $\rightarrow$ Vertex cover: a set of nodes S is a vertex cover of graph G if and only if the remaining nodes, V - S, are an independent set of G.

- Independent set $\rightarrow$ Clique: define $\bar{G} = (V, \bar{E})$, where $\bar{E}$ contains precisely those unordered pairs of vertices that are not in E. The set of nodes S is an independent set of $G = (V, E)$ iff S is a clique of $\bar{G}$ i.e. nodes in S have all possible edges between them in $\bar{G}$.

- Boolean circuit:
  OR: $y \geq x_1, y \geq x_2, y \leq x_1 + x_2$
  AND: $y \leq x_1, y \leq x_2, y \geq x_1 + x_2 - 1$
  NOT: $y = 1 - x$

- 3D matching $\rightarrow$ ZOE: column = triple, row = b, g, p.

- $T(0) = N$.
  $T(K+1) \geq \frac{1}{2^M}$ where M is the number of mistakes made by the best expert.
  $T(K+1) \leq (1 - \frac{1}{1+\epsilon})T(K)$ if weighted majority errors.
  Number of mistakes by the algorithm is upper bounded by $2(1+\epsilon)M + \frac{2\log N}{\epsilon}$.
  Hedge:
  $T(0) = N$.
  $T(K+1) \geq w_i^{(K+1)} \geq \exp(-\epsilon M)$.
  $T(K+1) = \sum_i w_i^{(K)} \exp(-\epsilon m_i^{(k)})$.
  It can be concluded that the total expectedd cost of Hedge is not much worse than the total cost of any individual (or best) expert.
  $$\sum_k \frac{w_i^t}{T(k)} m_i^{(k)} \leq \sum_k m_i^{(k)} + \frac{\ln N}{\epsilon} + \epsilon K$$
  Randomized:
  $T(K+1) \geq (1-\epsilon)^M$.
  $T(K+1) = \sum_i w_i^{(k)} (1-\epsilon)^{m_i^k} \leq T(K)(1 - \epsilon \sum_k \frac{w_i^t}{T(K)} m_i^{(K)})$.
  Thus
  $$\sum_k \frac{w_i^t}{T(k)} m_i^{(k)} \leq \frac{\ln N}{\epsilon} + M(1+\epsilon)$$

- Estimating frequency: $f_j - \frac{n}{k} \leq n_j \leq f_j$. Time $O(nk)$, space $O(k(\log n + \log m))$.
  Estimating number of distinct elements within factor of $1 \pm \epsilon$.

- If $f(n) = O(g(n))$, it should be the case that $\frac{f(n)}{g(n)}$ goes to some constant (i.e. does not go to infinity) as n goes to infinity. For $f(n) = \Omega(g(n))$, it should be the case that $\frac{f(n)}{g(n)}$ goes to a positive value (i.e. does not go to zero) as $n \to \infty$. For Θ, you want both to hold (i.e. it goes to some positive constant as n goes to infinity).

- Any exponential dominates any polynomial: 3^n dominates n^5 (it even dominates 2^n).
  Any polynomial dominates any logarithm: n dominates $(\log n)^3$. This also means, for example, that n^2 dominates $n \log n$.

- Number of bits in the binary representation of N: $\lceil \log(N+1) \rceil$. Depth of a complete binary tree with N nodes: $\lfloor \log N \rfloor$. $\log N = \sum \frac{1}{i} + \gamma$

- The sum of any increasing geometric series is, within a constant factor, simply the last term of the series.

- Master Theorem: If $T(n) = aT(\lceil \frac{n}{b} \rceil) + O(n^d)$ for some constants $a > 0, b > 1, d \geq 0$, then
  $$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}.$$

- $f(n) = n, g(n) = (\log n)^{\log \log n}$, then $f(n) = O(g(n))$??

- $f(n) = 2^{\sqrt{n}} \Rightarrow f(n) \in \Omega(n^c), \forall c > 0, f(n) \in O(\alpha^n), \forall \alpha > 1$.
  This shows that there are algorithms whose running time grows faster than any polynomial but slower than any exponential.

- For partial geometric series, consider $c > 1, c = 1, c = 1$ which might give different convergence.

- $T(n) = 2T(\sqrt{n}) + 3, T(2) = 3$
  The recursion tree is a full binary tree of height h, $n^{\frac{1}{2}^h} = 2 \Rightarrow h = \Theta(\log \log n)$. The work done at every node of this recursion tree is constant, so the total work done is simply the number of nodes of the tree, which is $2^{h+1} - 1 = \Theta(\log n)$, so$T(n) = \Theta(\log n)$.

- In a directed graph, acyclicity = linearizability = the absense of back edges during a DFS.

- The strongly connected components can be linearized by arranging them in decreasing order of their highest post numbers.
- Duplicate graph.
- Reverse graph.
- Edges: directed $\leftrightarrow$ undirected $\leftrightarrow$ bidirected.
- Given s and t, run algorithm on each as source.
- Sort, especially for linear problems.
- Add dummy nodes.
- Interger $\rightarrow$ polynomial $\rightarrow$ apply FFT.
- For DFS, consider multiple roots.
- FFT: must pad with 0 so that the degree becomes power of 2.
- For $C(x) = A(x)B(x)$, the size of FFT matrix should correspond to the padded degree of C not A or B.
- Use SCC to reduce to a dag problem. For a dag problem, process nodes in linearized order or the reverse, iteratively.
- Huffman encoding:

```
procedure Huffman(f)
    H = makequeue([1...n], key=frequency)
    for k = 1 to n:
        insert(H, i)
    for k = n+1 to 2n-1:
        i = deletemin(H), j = deletemin(H)
        create a node k with children i, j
        f[k] = f[i] + f[j]
        insert(H, k)
```