Lecture Notes

Day 1:

- Fibonacci number improvement
    - Use array to store every value
    - Achieve O(n)

Day 2: Max Bandwidth / DFS

- Graph reachability
    - X: explored vertices
    - F: reached vertices
    - U: unexplored vertices
- Runtime:
    - O(|V| + |E|)

Day 3:

- Max Bandwidth
    - Add edges from the highest weigh to the lowest. Stop when there is a path s to v
    - Let n = |V|, m = |E|, $m \le n^2$
    - Run either BFS / DFS on E_B:
        - Then worst time, needs to run DFS on E_B m times
            - Worst case: every edge has different weight. Not find until reach the smallest edge
    - Total runtime: O(m(m + n))
    - **Improvement: binary pick edge weight**
        - **Start with the median weight**
        - Runtime: O(logm (m + n)) → O(logn (m + n))
            - Since $m \le n^2$, logm → $\log n^2$ → 2logn → O(logn)
- Back edge:
    - In G but not in the DFS tree
    - If b.e then cycles
    - How to check?
        - If (u, v) is an edge, pre(u) ≠ v && pre(v) ≠ u
    - Removing a b.e will not **disconnect** the graph
- In undirected G:
    - Explore only reaches one connected component
- In directed G: for edge (u, v)
    - Tree edge / forward edge: in DFS tree $[_u\ [_v\ ]_v\ ]_u$
    - Back edge: leads to ancestor $[_v\ [_u\ ]_u\ ]_v$
    - Cross edge: leads to neither $[_v\ ]_v\ [_u\ ]_u$

Day 4

- Back edge is different between in directed and undirected G
- Cycles in D.G: start and end at the same vertex

        - D.G has a cycle **iff** its DFS output tree has a back edge
        - How to test → check the post number
- Linearization of **DAG**
    - No cycles
    - Every edge in DAG goes from highest post number to the lowest
- Source and sink
    - Source
        - Vertex without incoming edges → highest post number
    - Sink
        - Vertex without outgoing edges → highest post number
    - All DAGs have at least one source and one sink
- Strongly connected
    - Two vertices u and v are strongly connected if there is a path from u to v and a path from v to u
    - SC graph
        - Every pair is strongly connected
    - Every directed G is a DAG of its SCCs
        - Some SCCs are sources, some are sinks
    - **SCC**
        - How to look for SCCs
            - Find the sink SCC and remove it; repeat
            - How to find sink SCC
                - Source SCC has the highest post number
                - But the lowest post number **is not necessarily** the sink
                - **Use G^R and find its source → sink of G**
- Decomposition
    - Run DFS on G^R and keep track of the post number
    - Run DFS on G and order the vertices in decreasing order of post number
- DFS **not good for**
    - **Finding the shortest distance**

<u>Day 5</u>

- Graph reachability
    - Differences in F:
        - Stack → DFS
        - Queue → BFS
        - Priority queue → Dijkstra
- BFS:
    - Computes layer by layer
    - Works on find the shortest path on G whose edges have equal weight
    - For difference weight → add edges
- Dijstrak
    - Priority queue: O(|V| + |E|)

        - Use array as PQ: $O(|V|^2)$

Hw1

- Time analysis: induction. Formula. N! Omega 2^n; sum of i^k power series
- Recursive relation: Fibonacci
- Check triangular: for every edge, check all other vertices
- Correctness prove and time.

Day 6

- Structure in DJ
    - Graph: adjacency list
    - X: insert, check membership, array of booleans
    - F: find and delete key
- Use PQ:
    - Array as PQ:
        - Insert: O(1)
        - Deletemin: O(n)
        - decreaseKey: O(1)
        - DJ takes $O(|V|^2)$
    - Binary heap as PQ:
        - makeHeap: O(n)
        - deleteMin: O(log|v|)
        - decreaseKey: O(log|v|)
        - DJ takes O(log|v| * (|V| + |E|))
    - **Fibonacci takes O(|v|log|v| + |E|)**
- MST
    - Delete the max edge that does not disconnect the graph → Prim's
    - Keep adding the lightest edge that does not create a cycle → Kruskal's
        - Sort edges
        - How to check create a cycle → hw2.q2 → O(|v| + |E|)
            - Remove e and check if the graph is still connected
            - If yes, then e is a part of cycle; otherwise, e is not

Day 7

- DSDS
    - Tree
        - Undirected G with no cycles
        - A undirected G with n vertices is a tree **iff** it has n - 1 edges
    - Runtime of Kruskal's
        - Version 1:
            - makeset : O(1)
            - find(u): O(1)
            - Union: O(|V|)
            - Total: $O(|V|^2)$
        - Version 2: Trees

- Total: O(V + E log V + V log V + E log E)
- Version 3:
    - Total: O(|V| + |E| + |E|log|E|)

Day 8

- Optimization Problem:
    - Instance: input
    - Solution: output
    - Constraints: what property must a solution have
    - Objective function: quantity we are maxing or minimizing.

- **Greedy algorithm**
    - Immediate benefit Vs Opportunity cost
    - Optimal if IB > OC.
        - MIN: cost(OS) ≥ cost(GS)
        - MAX: value(OS) ≤ value(GS)

    - Event Scheduling
        - List of event E_1 E_2 … E_i = (s_i, f_i).
        - Non overlap, maximize size of subset
        - **Greedy: earliest end time.**
        - Implementation:

```
Initialize queue S
Sort the intervals by finish time
Put E_1 in S
Set F = f_1;
For i = 2 … n:
      If S_i ≥ F: enqueue (E_i, S)
      F = f_i
Return S.
```

Day 9:

- General Proof template:
- **Modify the solution:**
    - Let g be the first greedy choice
    - Let OS be a solution achieved by not choosing g.
    - **Show how to transform OS into some solution OS' that chooses**
        - **<u>Must show that OS' is a valid solution and OS' is better than OS</u>**
    - Use 1 - 3 as an inductive argument:
        - Base case: show greedy strategy works for instance of size 1
        - Assume greedy works for any instance I, |I| < n

- Let OS be any solution for instance I, |I| = n. Then there is another solution OS', such that |OS| ≤ |OS'| and OS' includes the 1st greedy choice g.
    - |OS| ≤ |OS'| = |{g} U OS(I')| ≤ |{g} U GS(I')| = GS(I)
- **Inductive template:**
    - 1. Let g be first greedy decision. Let I' be "rest of problem given g"
    - 2. GS= g + GS(I')
    - 3. OS is any legal solution.
    - 4. OS' is defined from OS by the MtS argument (if OS does not include g)
    - 5. OS' = g + some solution on I'.
    - 6. Induction: GS(I') at least as good as some solution on I'
    - 7. GS is at least as good as OS', which is at least as good as OS.
- Greedy stays ahead
- Achieves the bound
- Greedy Approximation

Day 10

- **Greedy stays ahead**
    - Define progress measure
    - Order the decisions in OS to line up with GS
    - Prove by induction that the progress after the i'th decision in GS is at least as big as that in OS
    - Assume that OS is strictly better than GS
    - Use progress argument to arrive at contradiction.
- **Achieves the bound**

Day 11 Divide and conquer

- Observation: (KS mult)
    - (a + b x) (b + c y) can be done with **three** multiplication, since bc + ad = **(a + b) (c + d) - ac - bd**.
    - $$x = \boxed{x_L}\,\boxed{x_R} = 2^{n/2}x_L + x_R$$ $$y = \boxed{y_L}\,\boxed{y_R} = 2^{n/2}y_L + y_R.$$
    - $$xy \;=\; (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \;=\; 2^n\,x_Ly_L \;+\; 2^{n/2}\,(x_Ly_R + x_Ry_L) \;+\; x_Ry_R.$$
- Multiplication strategy:
    - General: T(n) = 4 * T(n/2) + O(n) → O(**n^2**)
    - Reduced: T(n) = 3 * T(n/2) + O(n) → O(**$n^{1.59}$**)
    - Proof idea:

        - changes in the branching factor of recursion tree
        - Geometric increase from O(n) (k = 0) to O ($n^{\log 2\ 3}$) (k = $\log_2 n$).
        - DPK p. 52 - 53
- Cook-Toom algorithm
    - Split the problem into 2k - 1 subproblems of size n/k
    - T(n) = (2k - 1)T(n / k) + O(n)
    - Runtime:
        - O($n^{\log(2k - 1) / \log k}$)
        - Can achieve near-linear time

Day 12
- Machine frequency range problem
- Call the day

Day 13
- Polynomial representation

Day 14
- T(n) = aT(n/b) + O(n^d)
- Shortest height → height: takes Ω(logn)
- D/C search
    - Start with a sorted list and a target. Output the index of the target
    - Break into sublist O(1)
        - Half size
    - Solve each one recursively O(n / 2)
    - Combine O(1)
    - Runtime: T(n) = T(n / 2) + O(1) → O(logn)
- Sorting
    - Expected time O(n^2)
        - Bubble sort
        - Insertion sort
        - Selection sort
    - Expected time O(nlogn)
        - MergeSort
        - Quick Sort
    - Lower bound
        - Ω(log(n!)) = Ω(nlogn)
    - **mergeSort**
        - If n > 1
            - ML = MS(a[1, …, n/2])
            - MR = MS(a[n/2 + 1, …, n])
            - return merge(ML, MR)
        - Else

        - Return a
    - Runtime
        - $T(n) = 2T(n / 2) + O(n)$
        - $O(nlogn)$
- **Median**
    - If sort, $O(nlogn)$
    - Better way?
        - All selection $\Omega(n)$
        - Selection k^{th} element
            - Pick a random pivot v
            - Divide into 3 groups S_L, S_V, S_R → takes $O(n)$
            - Runtime
                - Expected: $T(n) = T(n / 2) + O(n)$
                - Worst: $T(n) = T(n - 1) + O(n)$
- **Quick sort**
    - **Pretty much like selection, takes O(nlogn)**

<u>Day 15</u>

- D/C examples
    - Max overlap
    - Divide by starting value
        - 3 possibilities
            - Either left, right or overlap
    - Runtime:
        - $T(n) = 2T(n / 2) + O(nlogn)$
        - $O(nlogn)$

<u>Day 16</u>

-