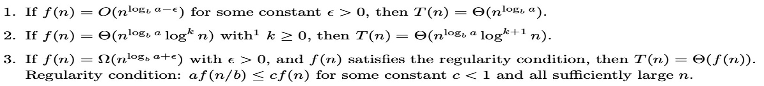
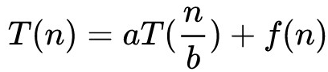
**Asymptotic Analysis:** T(n) is O(f(n)) if there exist constants c>0 and n0>=0 such that for all n>= n0 we have T(n)<=c・f(n). If false set Counterexample.

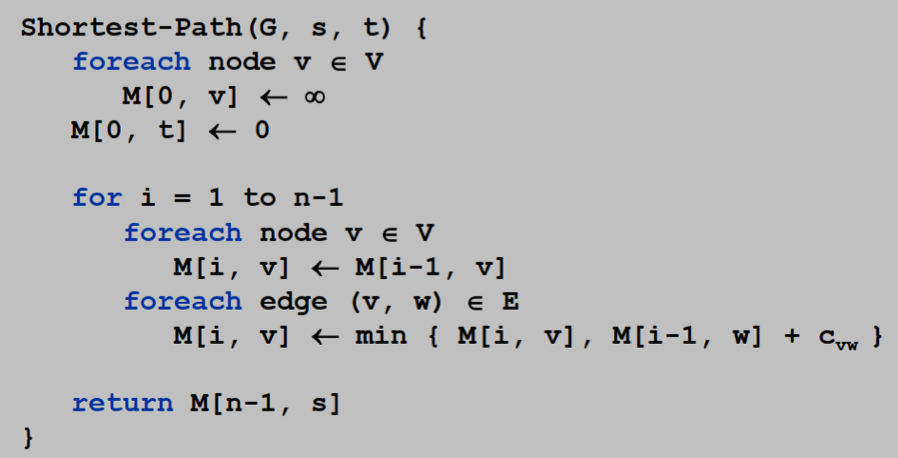
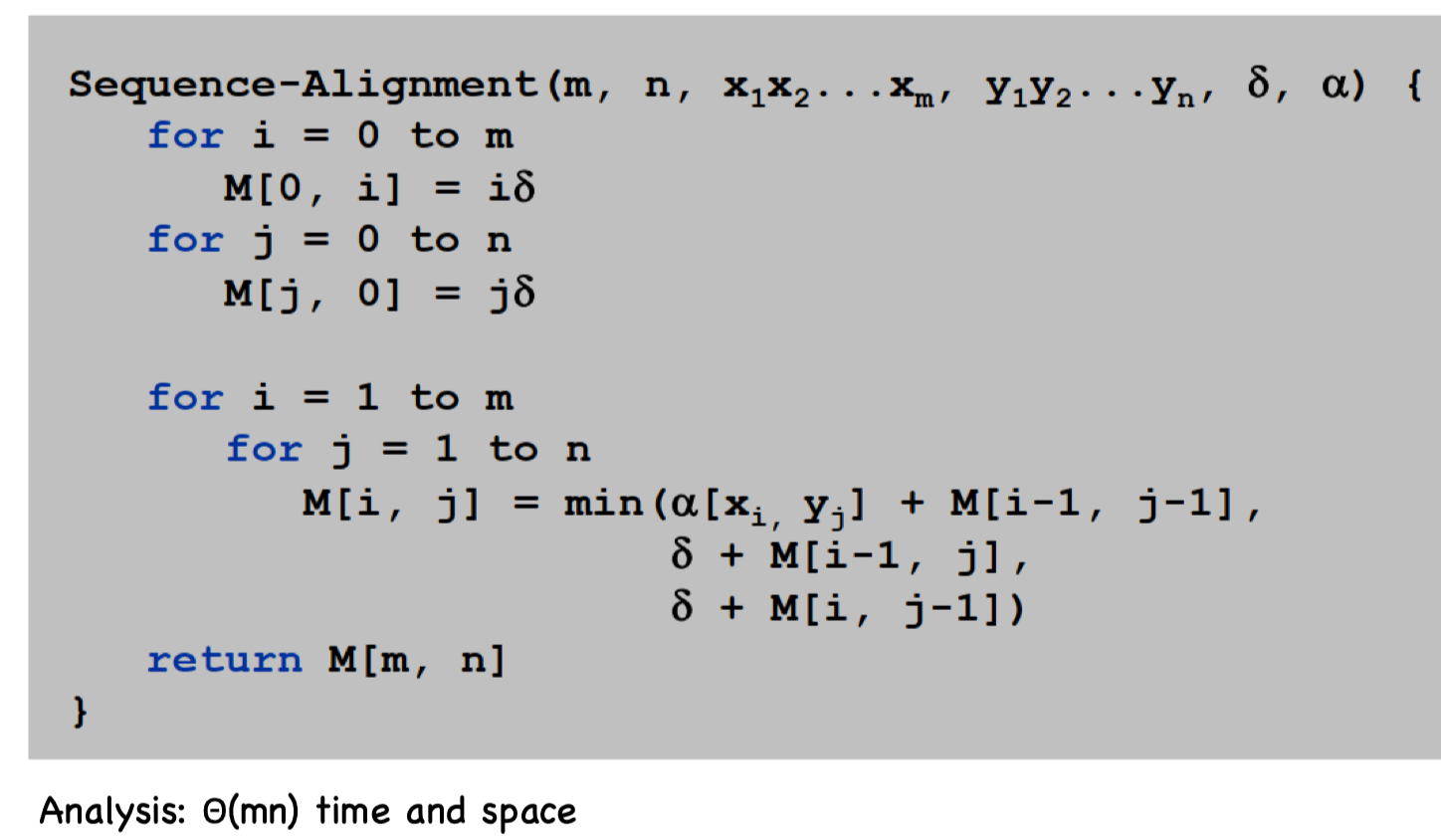
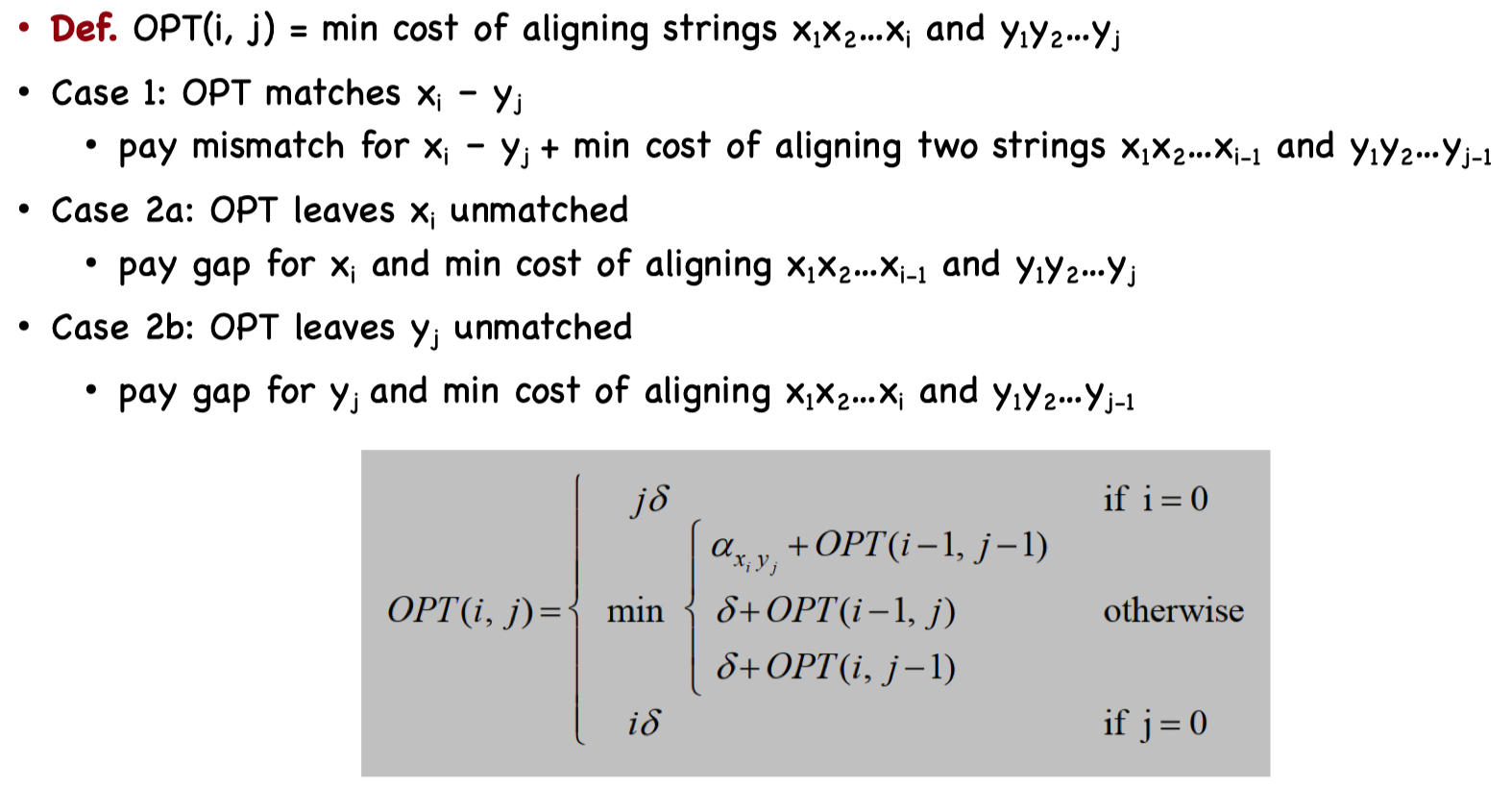
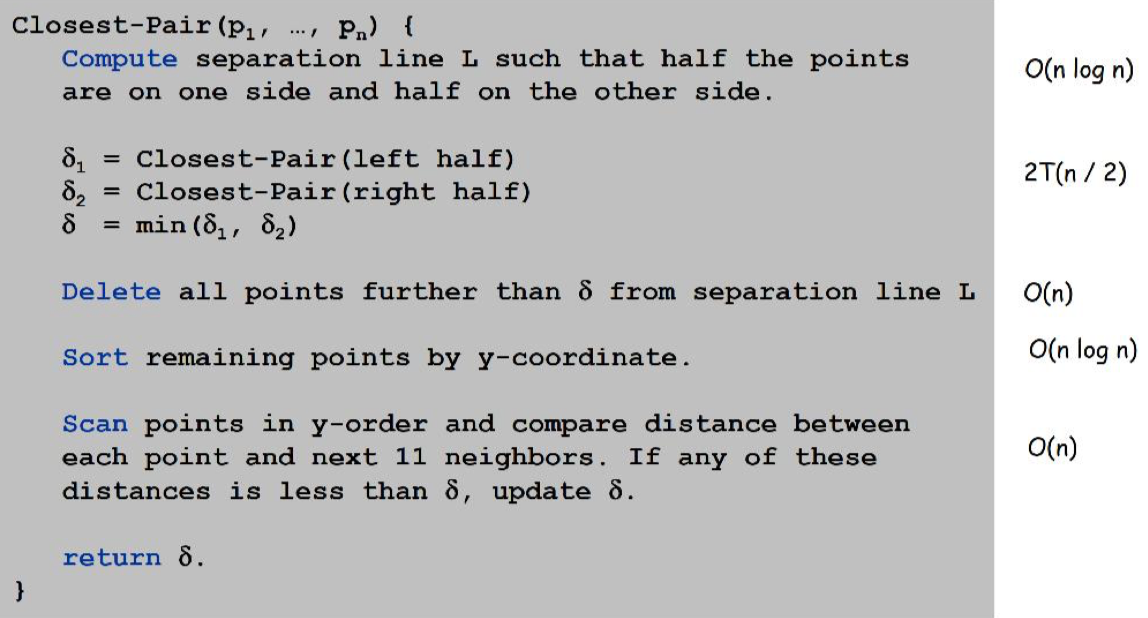
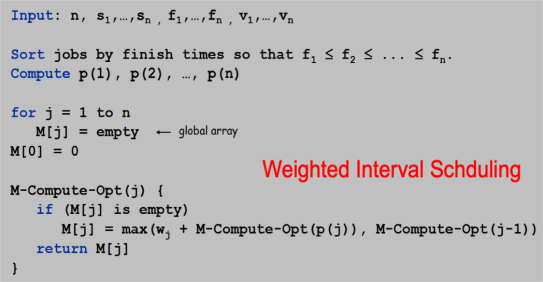
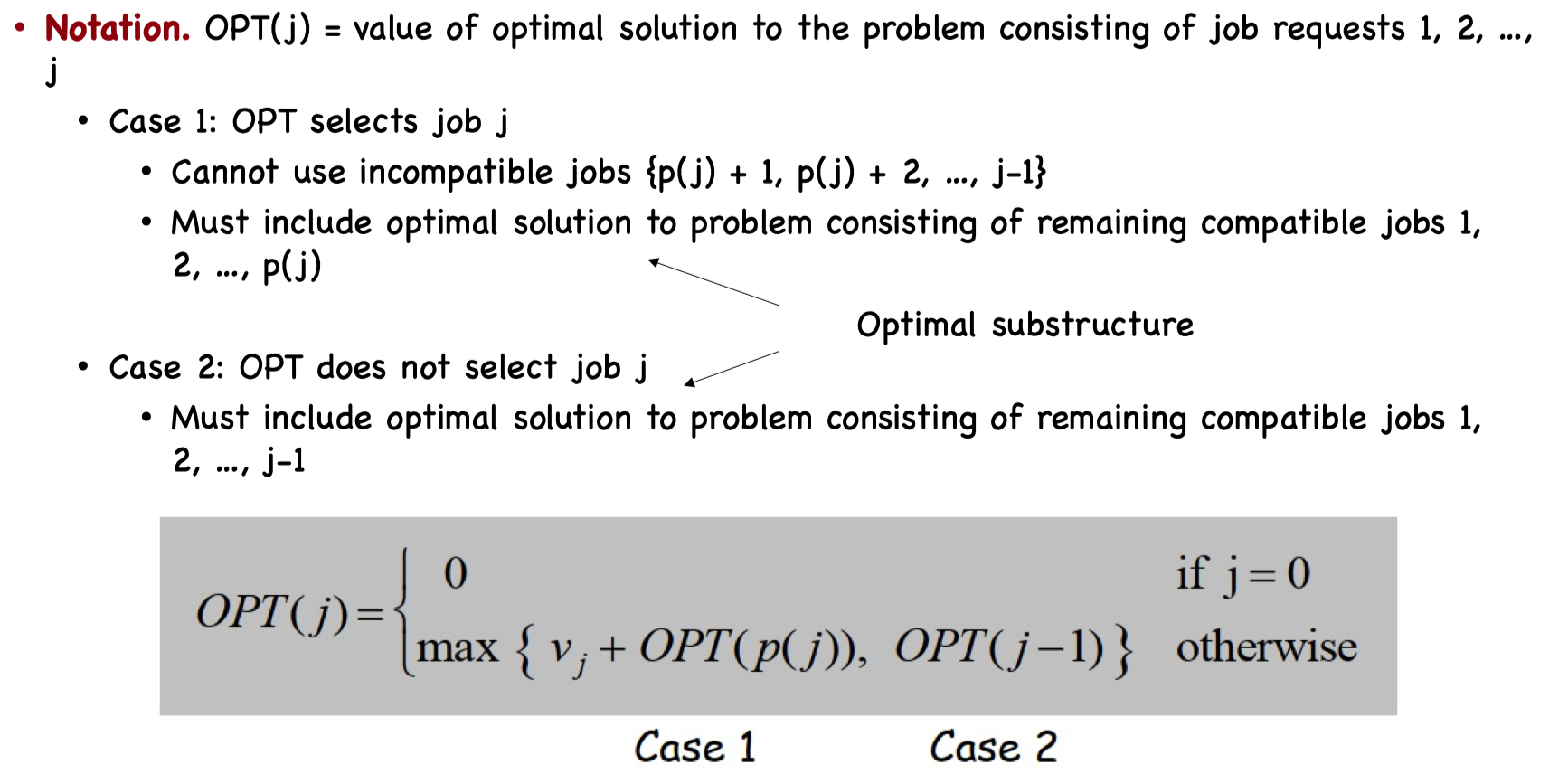


**Greedy Algorithms:** (a)An optimal algorithm is to schedule the jobs in decreasing order of w/t. We prove the optimality of this algorithm by an exchange argument. Consider any other schedule: As is standard in exchange argument, we observe that this schedule must contain an inversion. A pair of jobs i, j, for which i comes before j in the alternation solution, and j comes before i in the greedy algorithm. For this pair, we have w/tj ≥ w/ti, by the deﬁnition of the greedy schedule. If we can show that swapping this pair i, j does not increase the weighted sum of completion times, then we can iteratively do this until there are no more inversions, arriving at the greedy schedule without having increased the function we are trying to minimize → the greedy algorithm is optimal. (b)**Proof by contradiction:** Assume Greedy is different from OPT. Let’s see what’s different. Let i1 , i2 , …, ik denote the set of jobs selected by greedy. Let j1 , j2 , …, jm denote set of jobs in the optimal solution. Find largest possible value of r such that i1=j1,i2=j2,…,ir=jr. By definition of Greedy, job i r+1finishes before jr+1. Why not replace job jr+1with job ir+1. Solution still feasible and optimal, but contradicts maximality of r. **Cut lemma:** Let S be any subset of nodes, and let e be the min cost edge with one endpoint in S. Then every MST must contain e. **Prim:** 逐个加点. **Kruskal:** 逐条加边. Proof of Kruskal: 1.prove Kruskal produce a tree(acyclic and connected graph); 2.prove every edge chosen is in MST(Cut lemma); 3. Prove tree T is a MSL. **Reduced schedule claim:** any unreduced schedule can transform into reduced schedule with no more eviction.

**Divide and Conquer:** The important observation is that If A contains a majority element e, => e must be a maj. element in at least one of A[1..n/2] and A[n/2+1..n].Algorithm is then: Find majority e1, if it exists, in A[1..n/2], Find majority e2, if it exists, in A[n/2 + 1..n], If e1 exists, count how many times it occurs in A If more than n/2, report e1. If e2 exists, count how many times it occurs in A. If more than n/2, report e2.If neither exist, report, "no majority". The base case is when n=1, and we can simply return the only element as the majority. **Closest-pair:** T(n) = O(nlog2n)，可以优化至O(nlogn). • Divide: draw vertical line L so that roughly ½n points on each side. • Conquer: find closest pair in each side recursively. • Combine: find closest pair with one point in each side Θ(n^2 ). • Return best of 3 solutions.

**Dynamic Programming:** **Weighted Interval Scheduling:**

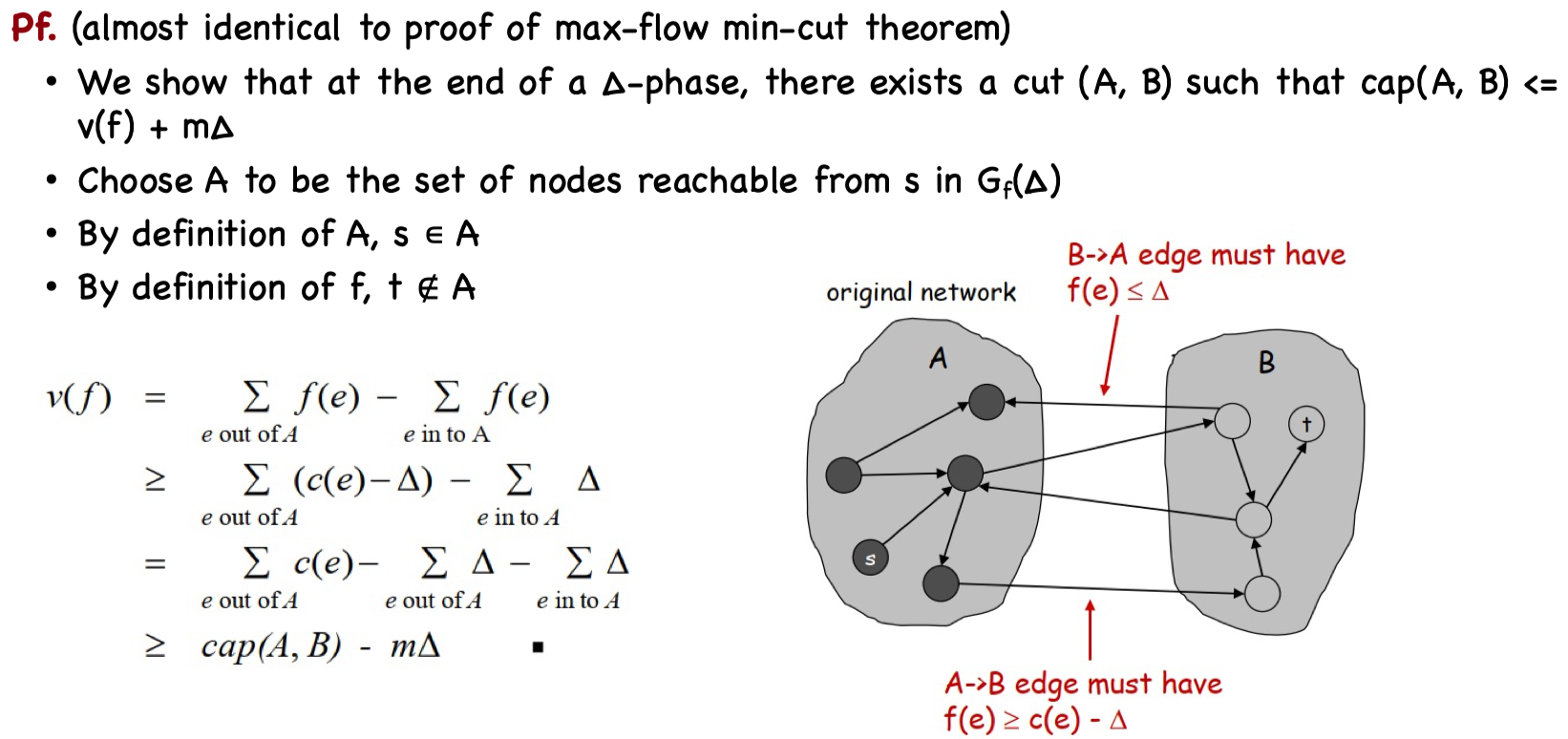
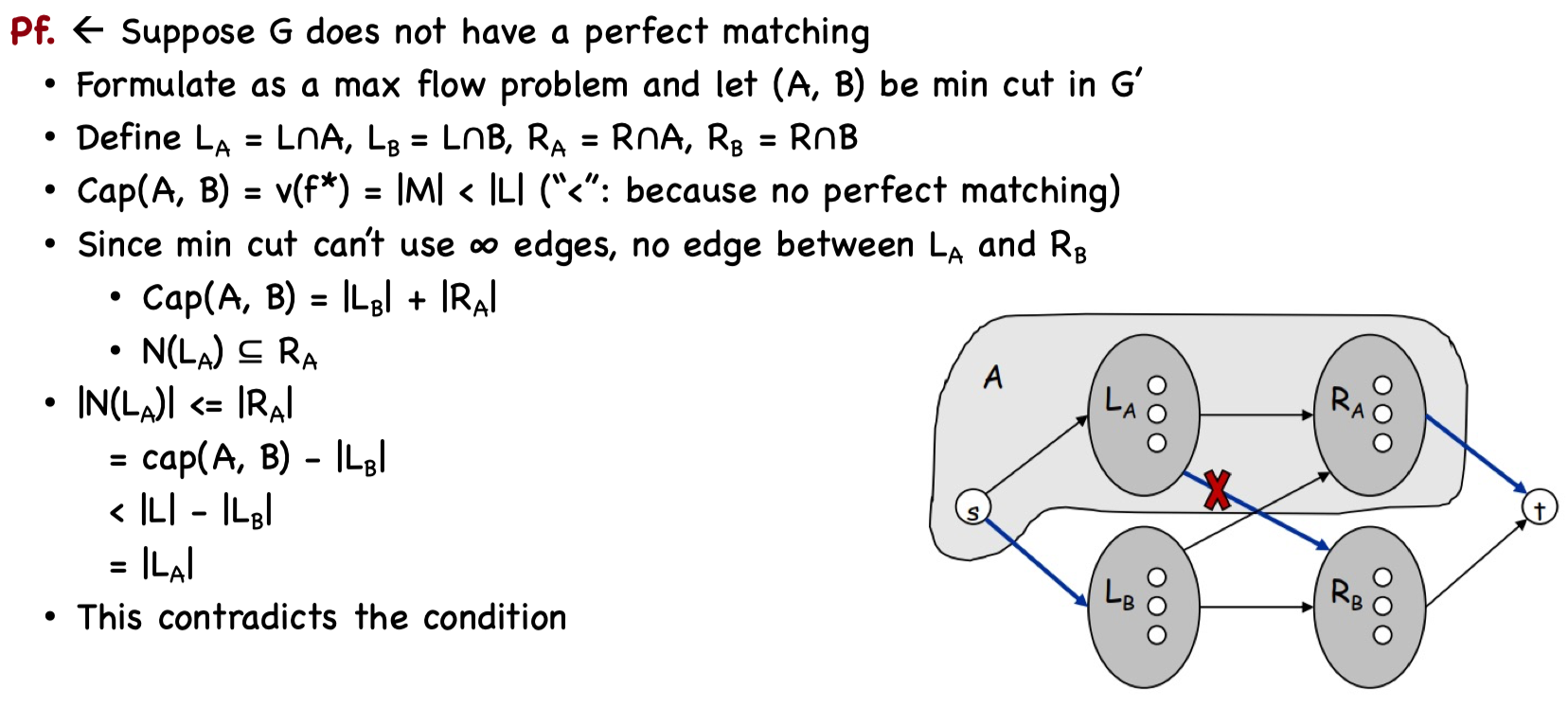
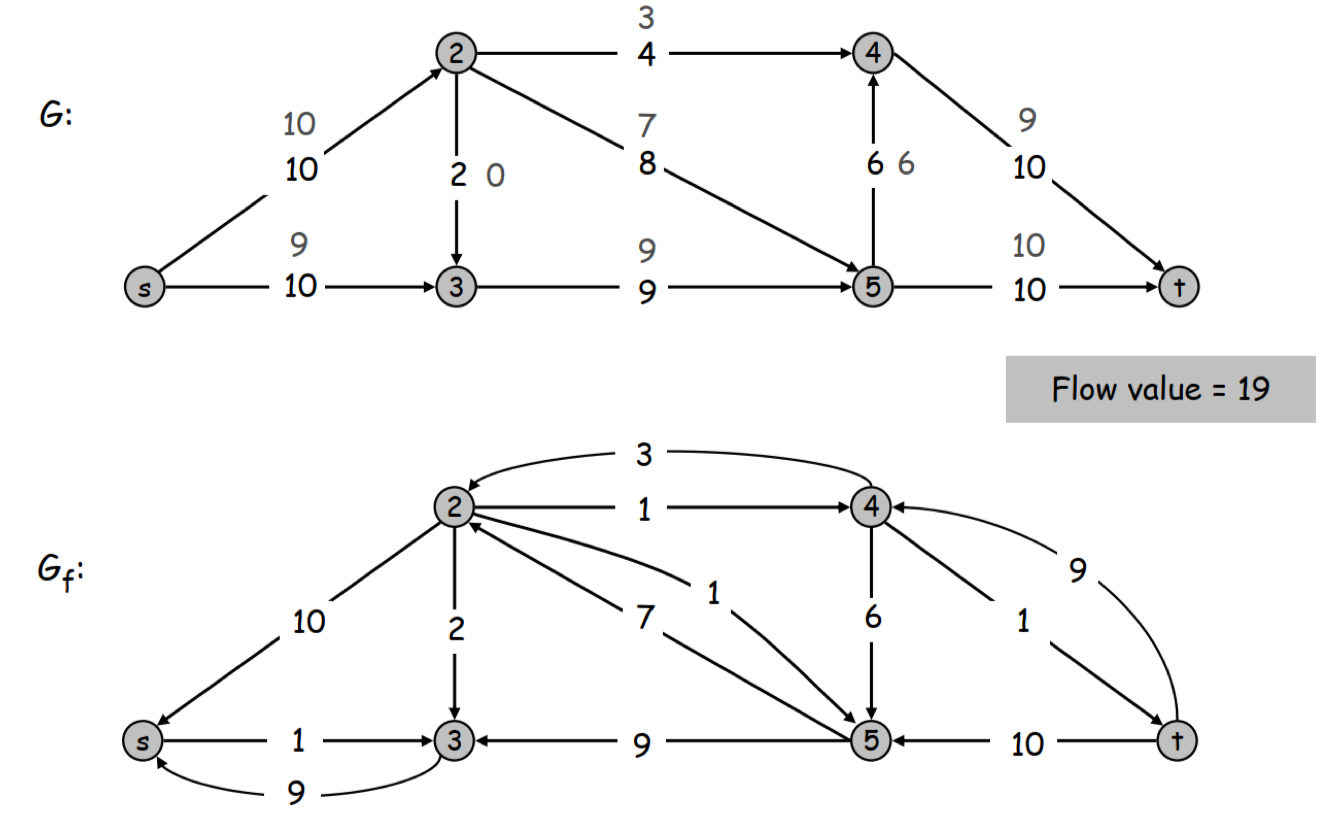
**Sequence Alignment:** time:O(mn) space:O(mn) to space:O(m+n). **Shortest Paths:** Θ(mn) time.Bellman-ford algorithm.Detecting Negative Cycles: If OPT(n,v) = OPT(n-1,v) for all v, then there is no negative cycle with a path to t. We can use Bellman-ford to detect negative cost cycle in O(mn) time.



**Network Flow:** **Flow network:** Abstraction for material flowing through the edges. G = (V, E) = directed graph. Two distinguished nodes: s = source, t = sink. c(e) = nonnegative capacity of edge e. **s-t flow:** a function that satisfies: For each e∊E: 0<=f(e)<=c(e)(capacity), For each v∊V – {s, t}: f(e)(e in to v)= f(e)(e out of v). The value of a flow f is: f(e)(e out of s). **Max flow problem:** Find s-t flow of maximum value. **Residual graph:** G f= (V, E f), Residual edges with positive residual capacity, E\_f={e:f(e)<c(e)}∪{e^R:f(e)>0}. **s-t cut:** a partition (A, B) of V with s∊A and t∊B, The capacity of a cut (A, B) is: c(e)(e out of A). **Min s-t cut problem:** Find an s-t cut of minimum capacity. **Marriage Theorem:** G has a perfect matching iff |N(S)|>=|S| for all subsets S ⊆ L. -> This was the previous observation.

**Flow value lemma:** Let f be any flow, and let (A, B) be any s-t cut. Then, f(e)(e out of A)-f(e)(e in to A)=Value(f).

**Weak duality:** Let f be any flow. Then, for any s-t cut (A, B) we have v(f) <= cap(A, B).



**Max-flow min-cut theorem:** The value of the max flow is equal to the value of the min cut.

**Ford-Fulkerson Algorithm:** Choosing Good Augmenting Paths. Choosing path with high bottleneck capacity. ∆-residual graph Gf(∆). ∆取2的幂小于C. Capacity Scaling Running Time: •Lemma 1. The outer while loop repeats 1+log2 C times. Pf. Initially C/2 < ∆ <= C. ∆ decreases by a factor of 2 each iteration. • Lemma 2. Let f be the flow at the end of a ∆-scaling phase. Then the value of the maximum flow is at most v(f) + m∆. •Lemma 3. There are at most 2m augmentation per scaling phase.Let f be the flow at then end of the previous scaling phase. L2-> V(f\*) <= v(f) + m(2∆). Each augmentation in a ∆-phase increases v(f) by at least ∆. The scaling max-flow algorithm finds a max flow in O(mlogC) augmentations. It can be implemented to run in O(m^2 logC) time.

