O 1 < n < nlogn<<n^2<2^n< n!  9log10.1n, n^310, 2(log96n)^2√−0.4, 101000⋅n!, n^n,2^n1.00001

12th: TRUE The shortest path problem (i.e. input is directed graph with arbitrary costs but with no negative cycle and output a shortest s−t path for given nodes s and t) is in NP.

11th: FALSE For the weighted interval schedule problem the values p(1),…,p(n) can be computed with O(n) comparisons

10th: TRUE Given two numbers with n hexadecimal digits each, we can multiply them in time O(n3√)

9th: TRUE Let T be an MST of a graph G with each edge e having cost ce. Then for every number Δ (which can be negative or positive), T is also an MST for the same graph G but with new edges costs c′e=ce+Δ.

8th: FALSE Every graph G (with costs ce>0 for every e∈E) has a unique MST.

7th TF: FALSE Let G be a graph G=(V,E), where each edge has length ℓe≥0. Let P be a shortest s−t path (according to the edge lengths ℓe). Then for every Δ>0, P is still a shortest s−t path if the edge lengths are changed to ℓ′e=ℓe+Δ

6th TF: FALSE For any connected graph G=(V,E) there is always a start vertex s∈V, such that there is always at least two distinct BFS trees possible when starting the BFS at s.

5thTF: FALSE Every directed graph on n vertices that has no cycles has at most n−1 edges

4th: FALSE f(N) is O(g(N)), then 2f(N) is O(2g(N))

3rd: FALSE Every algorithm in the worst case on inputs of size N needs to read all the N items in the input

2nd: FALSE In every Stable Matching problem instance where a man m and woman w have each other as their least preferred partner, the following is true. There is no stable matching for the instance where (m,w) are matched

1st: TRUE Given n numbers a1,…,an such that for every i∈[n] (we will use [n] to denote the set of integers {1,…,n}) we have ai∈{0,1}. That is, we are given n numbers each of which is a bit. Then we can sort these n numbers in O(n) time

The input size for BFS is Θ(n +m):: because the adjacency list representation takes Θ(m +n) spaceN = Θ(n +m)

Every graph has a a unique BFS tree for it.FALSE(4 node loop)

Directed graph has at most n^2 edges because at most n^2 pairs of verticies

Adjacency matrix takes Θ(n 2 ) space from lecture comparing graphs

N3 to n2 B[i, j] = max(B[i, j −1], A[j]) matrix

Hypercube cycle (vi , vi+1) Hamiltonian cycle

Hw0 goat-sort-#ofperfectmatches-assnote

Perfect matchings k×(k−1)!=k!k×(k−1)!=k!.

 log10.1n=log9nlog910.1<log9nlog10.1⁡n=log9⁡nlog9⁡10.1<log9⁡n, then 9log10.1n<

hw1 – program galeshapely – tv channel – n^n/3 stable pairs

hw2 – medstudents – home wrecker – big g interns

hw3-uppertraingle vector mult – matrix – fft triple matrix

hw4 adjacency and nodedistance-adfseating –avgvsmaxTreediameterdistance

hw5-largest cycle – butterflys – listing triangles

hw6-tennis/scheduling – high speed internet – swimming biking running

hw7 weightedgraph – network set intersection – shortest path in dags

hw8 – Minimum spanning trees – exponents – optimal prims

hw9 – closest pair of points – finding sinks – faster interger multiplication O(nlog35), which is O(n1.47)

hw10 – shortests paths from one node to all other nodes – work scheduling – library internet schedule

sample final: Every undirected connected graph on n vertices has exactly n −1 edges. FALSE (3.2) Consider G that is a cycle

BFS is a linear time algorithm. TRUE BFS runs in time O(n +m) and the input size (i.e. the graph) is also Θ(n +m

If all the edge weights of an undirected connected graph G are distinct, then G has a unique minimum spanning tree. TRUE

Given n numbers a1,...,an, the median of the smallest ten numbers and the largest ten numbers among them can be computed in O(n) time. TRUE

The maximum spanning tree problem is an NP-complete problem. FALSE

If f(n) = c · g(n), then 2f(n) = 2 g(n) c for every real number c.

This follows since for any real numbers a,b,c we have (a b ) c = a b·c

2 O(n) is O(2n ). == 2O(n) is O(2n ). (Or more precisely, every function f (n) that is 2O(n) is also O(2n ).)

Given n positive integers a1, . . . , an such that each ai ≤ n O(1); they can sorted in O(n log n) time. == Use mergesort and note that since each ai can be represented with O(logn) bits, we can compare any pair of them in O(1) time.

Given n numbers a1, . . . , an, where for every 1 ≤ i ≤ n, ai ∈ {−5, 9, 100}; their sorted order can be output in O(n) time == Given n numbers a1,...,an, where for every 1 ≤ i ≤ n, ai ∈ {−5, 9, 100}; their sorted order can be output in O(n) time.

Given two numbers with n bits, they can be multiplied in O(n 3/2 ) time. == This follows from HW 9 Q3 and the fact that log3 5 < 3/2

Given two numbers with n octal digits (i.e. the numbers are in base 8), they cannot be multiplied in time asymptotically faster than O(n 2 ) == FALSE The algorithm we saw in class (that runs in time O(n log2 3 ) works for any base that is constant. Thus we can multiply octal number faster than O(n 2 ) time

Given an undirected graph with n nodes and m edges with positive integer weights (that are polynomially in n large) and two distinct vertices s 6= t, the shortest s − t path can be computed in O((m + n) log n) time == Use Dijkstra’s algorithm (and note that since all weights are O(logn) bits, all arithmetic operations in the algorithm can be implemented in O(1) time)

Given an undirected unweighted graph G in n vertices and m edges and two distinct vertices s 6= t, the shortest s−t path can be computed in O(m +n) time == TRUE Run BFS on G starting with s. The shortest path between s and t is the unique path between them in the BFS tree.

Let G be a connected graph where each edge was weight 1. Then any BFS tree is an MST for G. == Since all edge weights are 1 and all spanning trees have n−1 edges, it implies that all spanning trees have cost exactly n − 1. Thus, all spanning trees (including a BFS tree) are MSTs

Let G be an undirected connected graph. If G has a unique minimum spanning tree, then all the edge weights in G are distinct. FALSE.

You’re given the Internet as a graph G = (V,E) such that |V | = n and |E| = m. Further for each edge e ∈ E, you’re given it’s probability 0 ≤ pe ≤ 1 == (a) Construct a new weighted graph G ′ = (V ′ ,E ′ ) with V ′ = V and E ′ = E but for every e ∈ E ′ define ℓe = log(1/pe ). (b) Run Dijkstra’s algorithm on G ′ (and edges lengths ℓe as above). Return the shortest s − t path from the run of the Dijkstra’s algorithm as the path with the smallest probability of failing in G

a bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint and independent sets { U} U and { V} V such that every edge connects a vertex in { U} U to one in { V} V

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage[1] with the intent of finding a global optimum.

Mon, Dec 3 More on Bellman-Ford Fri, Nov 30 Bellman-Ford algorithm

Wed, Nov 28 Shortest path problem Mon, Nov 26 Dynamic program for weighted interval scheduling Mon, Nov 19 Weighted Interval SchedulingFri, Nov 16 Kickass Property Lemma

Wed, Nov 14 Closest Pair of PointsMon, Nov 12 Couting InversionsFri, Nov 9 Integer MultiplicationWed, Nov 7 Runtime analysis of MergesortMon, Nov 5 Mergesort

Fri, Nov 2 Cut Property LemmWed, Oct 31 Minimum Spanning TreeMon, Oct 29 Dijkstra's Algorithm

Fri, Oct 26 Shortest Path ProblemWed, Oct 24 Exchange argument

Mon, Oct 22 Greedy algorithm for minimizing maxiumum lateness

Oct 12 Minimizing maximum latenessOct 10 Correctness of greedy algorithm

Oct 8 Greedy algorithm for interval schedulingOct 5 Interval Scheduling

Oct 3 Analysis of TopOrd algorithmOct 1 Topological OrderingSep 28 Runtime Analysis of BFS

Sep 26 Explore algorithmSep 24Computing Connected ComponenSep 21 Trees

Sep 19 Graph BasicsSep 17 Runtime analysis of Gale-Shapley algorithm

Sep 14 Analysing runtime of an algorithm Sep 12Asymptotic Analysi Sep 10Gale Shalpey stable

Sep 7 Gale Shapley algorithmsSep 5 Algorithms for StableAug 31Stable Matching

Dynamic programming -> subproblems

Fibonacci heap, the find-minimum operation takes constant ([*O*](https://en.wikipedia.org/wiki/Big_O_notation)(1)) amortized time.[[1]](https://en.wikipedia.org/wiki/Fibonacci_heap#cite_note-1) The insert and decrease key operations also work in constant amortized time.[[2]](https://en.wikipedia.org/wiki/Fibonacci_heap#cite_note-Fredman_And_Tarjan-2) Deleting an element works in *O*(log *n*)

Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.[1] It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers

Kruskal’s algorithm for Minimum Spanning Tree. Like Kruskal’s algorithm, Prim’s algorithm is also a Greedy algorithm

dijkstra O((m+n)logn)

csinterview:To make the description of the algorithm easier we first re-state the problem equivalently. We are given the input in the array A[0,..., 2n −1] and output should be as follows: for 0 ≤ i ≤ n −1, A[2i] is the original A[i] and A[2i +1] is A[n +i]. Now note that if we swap the 2nd the 3rd quarter of the original array A and then break it up into the middle, then we have two independent instance of the problem each with 2 ·n/2 = n numbers. We make sure that we perform the swap in O(n) time and use only constant space to do this. We then recurse and the runtime analysis and space usage follows by solving the simple recurrence relations. Here is the statement of the algorithm (we will use A as a global array that all recursive calls can access and we begin with the call Swap(0, 2n −1)): Swap(ℓ, r ) //ℓ is left most index and r is right most index into A m ← r −ℓ+1 If m = 2 then return. For i = 0...m/4−1 tmp ← A[ℓ+m/4+i] A[ℓ+m/4+i] ← A[ℓ+m/2+i] A[ℓ+m/2+i] ← tmp Swap(ℓ,ℓ+m/2−1) Swap(ℓ+m/2, r ) Correctness. Note that when n = 1 (i.e. m = 2 above) then the array A already has permuted order. For the more general case, the argument follows from the discussion above the algorithm description. 5 Resource Usage. It is easy to check that the run time T (n) and the temporary space usage S(n) are given by the following recurrences: • T (2) ≤ c and T (n) ≤ cn +2T (n/2). • S(1) ≤ c and S(n) ≤ c +S(n/2). We have seen in class

bellmanford-fresh:Consider the following graph G = ([n],E), where E has the following edges and costs: • (i,i +1) ∈ E for every 1 ≤ i ≤ n −1 with ce = −2. • (i,n − j) ∈ E for every 1 ≤ i ≤ n −2 and 0 ≤ j ≤ n −i −1 with ce = j. The terminal vertex will be n.