Final Exam Review

True-false questions

(1) **T F** The **best case** running time for INSERTION SORT to sort an n element array is $O(n)$.

Answer: True

(2) **T F** By the master theorem, the solution to the recurrence $T(n) = 3T(n/3) + \log n$ is $T(n) = \Theta(n \log n)$.

Answer: False

(3) **T F** Given *any* binary tree, we can print its elements in sorted order in $O(n)$ time by performing an inorder tree walk.

Answer: True. For each node in the tree we will be calling INORDER-TREE-WALK recursively exactly twice (once for the left subtree and once for the right subtree).

(4) **T F** Computing the median of n elements takes $\Omega(n \log n)$ time for any algorithm working in the comparison-based model.

Answer: False

(5) **T F** Every binary search tree on n nodes has height $O(\log n)$.

Answer: False

(6) **T F** Given a graph $G = (V, E)$ with cost on edges and a set $S \subseteq V$, let (u, v) be an edge such that (u, v) is the minimum cost edge between any vertex in S and any vertex in $V - S$. Then, the minimum spanning tree of G must include the edge (u, v). (You may assume the costs on all edges are distinct, if needed.)

Answer: True

(7) **T F** Computing a^b takes exponential time in n, for n-bit integers a and b.

Answer: True (Output size is exponential in n).

(8) **T F** There exists a data structure to maintain a dynamic set with operations Insert(x,S), Delete(x,S), and Member?(x,S) that has an expected running time of $O(1)$ per operation.

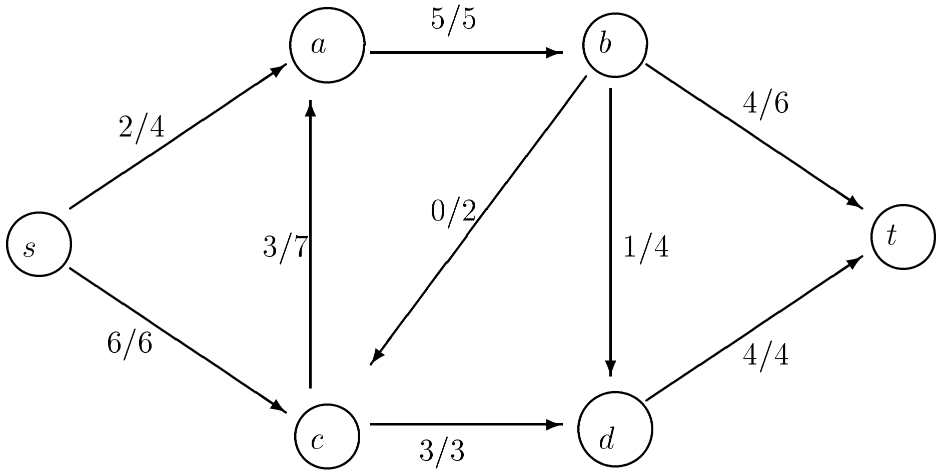
Answer: True. Use a hash table.

(9) **T F** The total amortized cost of a sequence of n operations (i.e., the sum over all operations, of the amortized cost per operation) gives a lower bound on the total actual cost of the sequence.

Answer: False. This only gives an upper bound on the actual cost.

(10) **T F** The figure below describes a flow assignment in a flow network. The notation a/b describes a units of flow in an edge of capacity b.

True or False: The following flow is a maximal flow.



Answer: True. The flow pushes 8 units and the cut $\{s, a, c\}$ vs. $\{b, d, t\}$ has capacity 8. The cut must be maximum by the Max-Flow Min-cut theorem.

(11) **T F** Let $G = (V, E)$ be a weighted graph and let M be a minimum spanning tree of G. The path in M between any pair of vertices v_1 and v_2 must be a shortest path in G.

Answer: False. Consider the graph in which $V = \{a, b, c\}$ and the edges are (a, b), (b, c), (c, a). The weight of the edges are 3,3 and 4 respectively. The MST is clearly (a, b), (b, c). However the shortest path between a and c is of cost 4 not 6 as seen from the MST.

(12) **T F** $n \lg n = O(n^2)$

Answer: True. Clearly $n \lg n \leq n^2$.

(13) **T F** Let P be a shortest path from some vertex s to some other vertex t in a graph. If the weight of each edge in the graph is increased by one, P remains a shortest path from s to t.

Answer: False Consider the graph $G = (V, E)$ where $V = \{a, b, c\}$ and $E = \{(a, b), (b, c), (a, c)\}$. Let the weight of the edges be $w(a, b) = 1, w(b, c) = 1, w(a, c) = 2.5$. The shortest path from a to c is a–b–c. However if the weights are increased by 1, the new shortest path is a–c.

(14) **T F** Suppose we are given n intervals (l_i, u_i) for $i = 1, \cdots, n$ and we would like to find a set S of non-overlapping intervals maximizing $\sum_{i \in S} w_i$, where w_i represents the weight of interval (l_i, u_i). Consider the following greedy algorithm. Select (in the set S) the interval, say (l_i, u_i) of maximum weight w_i, remove all intervals that overlap with (l_i, u_i) and repeat. This algorithm always provides an optimum solution to this interval selection problem.

Answer: False. Consider the following 3 intervals (1,10), (2,5) and (6,9) with weights 10, 6 and 6. The above greedy strategy would choose the intervals $\{(1, 10\}$ which has total weight 10 while the optimal set of intervals is $\{(2, 5), (6, 9)\}$ which has total weight 12.

(15) **T F** Given a set of n elements, one can output in sorted order the k elements following the median in sorted order in time $O(n + k \log k)$.

Answer: True. Find the median and the partition about it to obtain all the elements greater than it. Now find the kth largest element in this set and partition about it and obtain all the elements less than it. Output the sorted list of the final set of elements. Clearly, this operation costs $O(n + k \lg k)$ time.

(16) **T F** Consider a graph $G = (V, E)$ with a weight $w_e > 0$ defined for every edge $e \in E$. If a spanning tree T minimizes $\sum_{e \in T} w_e$ then it also minimizes $\sum_{e \in E} w_e^2$, and vice versa.

Answer: True. Only the relative order of the weights matter, the actual weights do not matter for the MST. And as if $w(e_1) \leq w(e_2)$, then $(w(e_1))^2 \leq (w(e_2))^2$, the result holds.

(17) **T F** The breadth first search algorithm makes use of a stack.

Answer: False. BFS uses a queue.

(18) **T F** In the worst case, merge sort runs in $O(n^2)$ time.

Answer: True. Merge sort runs in $O(n \lg n)$ time which is $O(n^2)$.

(19) **T F** A heap can be constructed from an unordered array of numbers in linear worst-case time.

Answer: True. A heap can always be constructed in $O(n)$ time using the BUILD-HEAP procedure.

(20) **T F** No adversary can elicit the $\Theta(n^2)$ worst-case running time of randomized quicksort.

Answer: True. No adversary has control over which random numbers the algorithm will use, and no adversary can determine which random numbers the algorithm will use. Therefore, although it is true that RANDOMIZED-QUICKSORT runs in $\Theta(n^2)$ time in the worst case, no adversary can force this behavior.

(21) **T F** Radix sort requires an "in place" auxiliary sort in order to work properly.

Answer: False. Radix sort needs a stable sorting algorithm.

(22) **T F** A longest path in a dag $G = (V, E)$ can be found in $O(V + E)$ time.

Answer: True

(23) **T F** The Bellman-Ford algorithm is not suitable if the input graph has negative-weight edges.

Answer: False. Bellman Ford is used when there are negative weight edges, it's Dijkstra's algorithm that cannot be used.

(24) **T F** Memoization is the basis for a top-down alternative to the usual bottom-up version of dynamic programming.

Answer: True

(25) **T F** Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, the shortest path between every pair of vertices $u, v \in V$ can be determined in $O(V^3)$ worst-case time.

Answer: True. Floyd-Warshall's algorithm performs exactly this in $O(V^3)$ time.

(26) **T F** For hashing an item into a hash table in which collisions are resolved by chaining, the worst-case time is proportional to the load factor of the table.

Answer: False. Even with a low load factor (say $\frac{1}{2}$) in the worst case you can get all n elements to hash to the same slot

(27) **T F** A red-black tree on 128 keys must have at least 1 red node.

Answer: True

(28) **T F** The move-to-front heuristic for self-organizing lists runs no more than a constant factor slower than any other reorganization strategy.

Answer: True. In recitation we've seen that it's *4-competitive*, which means that the best algorithm that can exist ("God's algorithm") could only do better by a factor of 4.

(29) **T F** Depth-first search of a graph is asymptotically faster than breadth-first search.

Answer: False. They both run in time $\Theta(V + E)$.

(30) **T F** Dijkstra's algorithm is an example of a greedy algorithm. **Answer:** True

(31) **T F** Fibonacci heaps can be used to make Dijkstra's algorithm run in $O(E + V \lg V)$ time on a graph $G = (V, E)$.

Answer: True

(32) **T F** The Floyd-Warshall algorithm solves the all-pairs shortest-paths problem using dynamic programming.

Answer: True

(33) **T F** A maximum matching in a bipartite graph can be found using a maximum-flow algorithm.

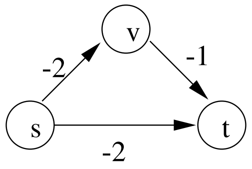
Answer: True. We've seen this in recitation.

(34) **T F** For any directed acyclic graph, there is only one topological ordering of the vertices.

Answer: False. Consider a graph with two vertices and no edges. Either order is a topological ordering.

(35) **T F** If some of the edge weights in a graph are negative, the shortest path from s to t can be obtained using Dijkstra's algorithm by first adding a large constant C to each edge weight, where C is chosen large enough that every resulting edge weight will be nonnegative.

Answer: False. Consider the following figure. The shortest path from s to t goes through v. But if we add 2 to each edge, so that we make the costs nonnegative, the edge from s to t becomes a shorter path.

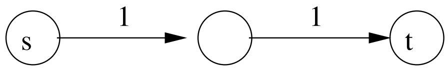


(36) **T F** If all edge capacities in a graph are integer multiples of 5 then the maximum flow value is a multiple of 5.

Answer: True. Consider the minimum cut. It is made up of edges with capacities that are multiples of 5, so the capacity of the cut (sum of capacities of edges in the cut) must be a multiple of 5. By the maxflow-mincut theorem, the maximum flow has the same value.

(37) **T F** For any graph G with edge capacities and vertices s and t, there always exists an edge such that increasing the capacity on that edge will increase the maximum flow from s to t in G. (Assume that there is at least one path in the graph from s to t.)

Answer: False. There may be more than one minimum cut. For example, consider the following graph. There is no single edge you can increase to increase the flow.



(38) **T F** Heapsort, quicksort, and mergesort are all asymmptotically optimal, stable comparison-based sort algorithms.

Answer: False

(39) **T F** If each operation on a data structure runs in O(1) amortized time, then n consecutive operations run in $O(n)$ time in the worst case.

Answer: True. Recall that amortized analysis does not make any assumptions about the possible distribution of inputs (that is, it doesn't take into account what happens "on average")

(40) **T F** A graph algorithm with $\Theta(E \log V)$ running time is asymptotically better than an algorithm with a $\Theta(E \log E)$ running time for a connected, undirected graph $G(V, E)$.

Answer: False. Compare $\log V$ and $\log E$. We know that $E \leq V^2/2$ (one edge for each pair of vertices), which means $\log E \leq 2 \log V$. That factor of 2 won't do us any good asymptotically

(41) **T F** In $O(V + E)$ time a matching in a bipartite graph $G = (V, E)$ can be tested to determine if it is maximum.

Answer: True: use DFS to determine there is an augmenting path in the corresponding flow graph, which can be done in linear time

(42) **T F** n integers each of value less than n^{100} can be sorted in linear time.

Answer: True. Use radix sort, for example

(43) **T F** For any network and any maximal flow on this network there always exists an edge such that increasing the capacity on that edge will increase the network's maximal flow.

Answer: False. There may be *two* min-cuts, with the edge in question increasing the capacity of only one of them. The other one will remain as it is, preventing the max-flow from increasing any further

(44) **T F** If the depth-first search of a graph G yields no back edges, then the graph G is acyclic.

Answer: True

(45) **T F** Insertion in a binary search tree is "commutative". That is, inserting x and then y into a binary search tree leaves the same tree as inserting y and then x.

Answer: False. Consider inserting x, y in an empty binary search tree to see why.

(46) **T F** A heap with n elements can be converted into a binary search tree in $O(n)$ time.

Answer: False. Building a heap takes $O(n)$ time. If we could convert that heap into a binary search tree in $O(n)$ time, then we could perform an *inorder tree walk* (which we know takes $O(n)$ time) and thus sort n elements in a total of $O(n)$ time. This contradicts the fact that any comparison-based sorting algorithm takes $\Omega(n \log n)$ time to sort n elements