Final Examination

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 180 minutes to earn 180 points. Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- This quiz is closed book. You may use **three** $8\frac{1}{2}'' \times 11''$ or A4 crib sheets (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required.
- When asked for an algorithm, your algorithm should have the time complexity specified in the problem with a correct analysis. If you cannot find such an algorithm, you will generally receive partial credit for a slower algorithm **if you analyze your algorithm correctly**.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Parts	Points	Grade	Grader
1	7	21		
2	9	54		
3	3	15		
4	3	20		
5	4	25		
6	4	25		
7	3	20		
Total		180		

Name: ______________________________

Friday Recitation:	Zuzana **10 AM**	Debmalya **11 AM**	Ning **12 PM**	Matthew **1 PM**	Alina **2 PM**	Alex **3 PM**

Problem 1. True or False [21 points] (7 parts)

For each of the following questions, circle either T (True) or F (False). There is no penalty for incorrect answers. You are not required to give any justification for your answer.

(a) T F [3 points] Every vertex reachable from a vertex v in an undirected graph G is either a descendant or an ancestor of v in any DFS tree of G.

(b) T F [3 points] In the DAG representation of a dynamic program, an edge between two sub-problems indicates that they are identical.

(c) T F [3 points] Assuming simple uniform hashing, the collision probability of a new key being inserted in a hash table using open addressing is at least as much as in a hash table using chaining. (Assume that both hash tables are of the same size and contain the same set of keys.)

(d) T F [3 points] If problem A is polynomial-time reducible to problem B and A is in **P**, then B is in **P** as well.

(e) T F [3 points] **Heapsort** is a *stable* sorting algorithm.

(f) T F [3 points] (The 2-SAT problem is a constrained version of the SAT problem where every clause may contain at most two literals.)
2-SAT is **NP**-hard since it is a special case of SAT.

(g) T F [3 points] We can design an algorithm that sorts any 5 numbers using 6 comparisons.

Problem 2. Short Answers [54 points] (9 parts)

For each of the following questions, you are expected to give a short answer that fits in the space provided for each question.

(a) [7 points] Suppose that an algorithm runs on a tree containing n nodes. What is the time complexity of the algorithm if the time spent per node in the tree is proportional to:

- [3 points] the number of children of the node? (Assume that the algorithm spends $O(1)$ time per leaf.)

- [4 points] the number of grandchildren of the node? (Assume that the algorithm spends $O(1)$ time for every node that does not have a grandchild.)

(b) [5 points] Solve the following recurrence:

$$\begin{aligned} T(n) &= T(\sqrt{n}) + O(1) \quad \text{if } n > 2 \\ T(n) &= O(1) \quad \text{otherwise.} \end{aligned}$$

(c) [8 points] For each of the following schemes in an expanding hash table, explain why it does or does not result in $O(1)$ amortized cost per insertion operation. You may assume that no key is deleted.

- [2 points] Increase the size by 1 on every insertion.

- [2 points] Increase the size by 50 every time the load factor is at least 0.75.

- [2 points] Square the size every time the load factor is at least 0.75.

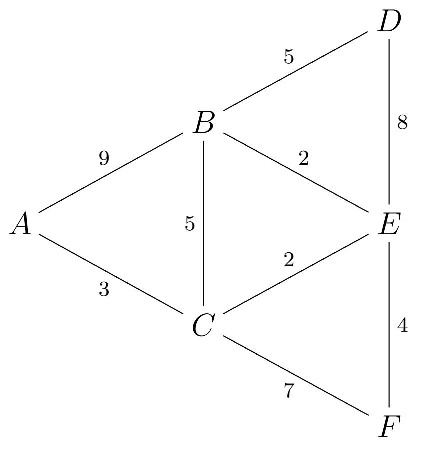
- [2 points] Grow the size by 1% every time the load factor is at least 0.30.

(d) [5 points] Assuming simple uniform hashing, show that the probability of no collision when n keys are inserted in a hash table of size n^2 is at least 1/2.
Hint: You might want to use the union bound which states that for a set of k events $E_1, E_2, \ldots, E_k$,

$$\text{Prob}[\cup_{i=1}^{k} E_i] \leq \sum_{i=1}^{k} \text{Prob}[E_i].$$

(e) [6 points] Given a weighted undirected graph, the k-*Traveling Salesman Problem* (k-TSP) asks whether the weight of the shortest cycle containing all vertices, is at most k. Show that k-TSP is **NP**-hard by a reduction from the Hamiltonian cycle problem. (*Hint*: Consider the k-TSP problem on a graph with only two types of edges—edges with weight 1 or ∞.)

(f) [6 points] Let G be the following weighted undirected graph:



Suppose that we were to run Dijkstra's algorithm starting from vertex A. In what order are the vertices extracted from the priority queue?

(g) [8 points] Suppose that you have a processor that enables you to perform a 3-way comparison in $O(1)$ time. In other words, given three distinct numbers a, b, c, the comparator outputs $a < b < c$ or $a < c < b$ or $b < a < c$ or $b < c < a$ or $c < a < b$ or $c < b < a$ depending on the relative magnitudes of the numbers. Using such a comparator, is it possible to sort any 6 distinct numbers using three 3-way comparisons? Justify.

(h) [4 points] In modern software development, a useful utility called *make* is usually employed to manage the compilation order of files. The problem is: you are given n files that need to be compiled. Some of the files can only compile when some other files have been compiled. Your job is to find an ordering to compile the n files. Can you solve this problem in polynomial (in n) time by some algorithm you learned in this course (you only need to spell out the name of the algorithm)? If so, what is the running time of your algorithm?

(i) [5 points] Suppose you want to estimate the value of $1000/7$. Show that if the starting value is $x_0 = 10^9$ in Newton's method for division, then you will never get to a value that is accurate to 6 decimal places.

Problem 3. Finding Bridges [15 points] (3 parts)

Given an undirected, connected graph $G = (V, E)$, a *bridge* in the graph is an edge whose removal would break the graph into two pieces. (In other words, a bridge is an edge $e \in E$ such that $G = (V, E)$ is connected but $G' = (V, E - \{e\})$ is disconnected.)

(a) [5 points] Show that every bridge must appear in every BFS tree of the graph.

(b) [5 points] Show that for any edge e, you can test whether e is a bridge or not in $O(E)$ time.

(c) [5 points] Combine parts (a) and (b) to design an algorithm that finds *all* bridges in G in $O(V \cdot E)$ time.

Problem 4. Maximum Weight Matching [20 points] (3 parts)

Let T be a tree where the weight of an edge e is denoted by $w(e)$. The *maximum weight matching* (MWM) in T is a subset of edges of maximum total weight such that no two edges are incident on the same vertex. We will use dynamic programming to design an algorithm for finding an MWM in T.

(a) [7 points] Clearly state the set of subproblems that you will use to solve this problem. (**Hint**: Observe that the MWM of a subtree contains either 0 or 1 edge among those incident on the root of the subtree. If one edge is present, the other endpoint of the edge cannot have another edge incident on it in the MWM. This requires you to expand the set of subproblems.)

(a) [7 points] Write a recurrence relating the solution of a subproblem to solutions of smaller subproblems.

(a) [6 points] Analyze the running time of your algorithm, including the number of subproblems and the time spent per subproblem. Your algorithm should have a time complexity of $O(n)$ overall, where n is the number of vertices in the tree.

Problem 5. Subsequence Sum [25 points] (4 parts)

You are given a sequence of n integers $a_1, \ldots, a_n$ (the integers can be both positive and negative). Your goal is find two indices i and j such that $\sum_{k=i}^{j} a_k$ is maximum.

(a) [3 points] What is the time complexity of an algorithm that tries all subsequences of length 1, length 2, …, length n and takes the maximum of all these sums?

(b) [6 points] Since you learned dynamic programming, you can store previous computation results in a table to save time. Using this idea, what would be the running time of your algorithm?

(c) [8 points] A useful technique you learned in this class is *divide-and-conquer*. If you divide the sequence into two equal halves and solve them recursively (of course, you need an efficient method for combining the results from the two halves to get the answer to the whole sequence), how would you design your new algorithm to solve this problem? What is the running time of your new algorithm? (This algorithm should run faster than your previous algorithm).

(d) [8 points] Consider the following algorithm. Scan the sequence from left to right and keep track of only two quantities: *maximum sum so far* and *maximum sum ending at the current integer*. Design an algorithm based on this idea. What is the running time of this algorithm? Why is this the *asymptotically* best possible running time?

Problem 6. **k-Sorting** [25 points] (4 parts)

An array $A[1 \ldots n]$ is said to be k-unsorted if each element's index is at most k away from its index in the sorted array (assume that all elements in A are distinct). We will design two different algorithms to sort a k-unsorted array in $O(n \log k)$ time in this problem.

(a) [7 points] Let $A[1 \ldots n]$ be a k-unsorted array. Suppose you have a sorted array S containing the smallest i numbers in $A[1 \ldots k + i]$ and a data structure R containing the remaining numbers in $A[1 \ldots k + i]$. What data structure (that you have learned in class) would you use to implement R so that you can obtain S and R for $A[1 \ldots k + i + 1]$ from S and R for $A[1 \ldots k + i]$ in $O(\log k)$ time?

(b) [6 points] Use part (a) to give an $O(n \log k)$-time algorithm for sorting a k-unsorted array.

(c) [5 points] Now, we use the *divide-and-conquer* paradigm to solve the same problem. Suppose that you have recursively sorted $A[1 \ldots n/2]$ and $A[n/2+1 \ldots n]$. Which are the only elements that might be out of their correct sorted position in $A[1 \ldots n]$? How long would you take to sort them thereby obtaining the final sorted array?

(d) [7 points] Use part (c) to give an $O(n \log k)$-time divide-and-conquer algorithm for sorting a k-unsorted array.

Problem 7. Approximate Minimum Vertex Cover [20 points] (3 parts)

Recall that a *vertex cover* (VC) of an undirected graph is a subset of vertices such that every edge is incident on at least one vertex of the subset. A *minimum vertex cover* (MVC) is a VC of minimum size. Consider the following algorithm for finding a vertex cover of an undirected graph $G = (V, E)$.

Repeat the following until the graph has no edge left: *Choose an edge arbitrarily and add both its endpoints to the vertex cover. Remove all edges incident on at least one of these two vertices.*

For completeness, this algorithm is also given in pseudocode below:

```
$S = \emptyset$
while $E \neq \emptyset$ do
    Choose an arbitrary edge $e = (u, v) \in E$
    $S := S \cup \{u, v\}$
    for each edge $f \in E$ do
        if $u$ or $v$ is an endpoint of $f$ then
            $E = E \setminus \{f\}$
        end if
    end for
end while
return $S$
```

(a) [6 points] Show that S is indeed a VC when the algorithm terminates.

(b) [7 points] Show that if a graph contains a set of k edges not sharing any endpoint, then any VC must contain at least k vertices.

(c) [7 points] Use part (b) to deduce that the number of vertices in S when the algorithm terminates is at most twice the number of vertices in any MVC of G.

SCRATCH PAPER

SCRATCH PAPER

SCRATCH PAPER