

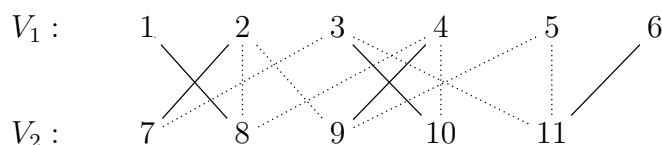
Name:
ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

Quick links [1a](#) [1b](#) [1c](#) [2a](#) [2b](#) [2c](#) [3a](#) [3b](#)

1. A *matching* in a graph G is a subset $E_M \subseteq E(G)$ of edges such that each vertex touches at most one of the edges in E_M . Recall that a bipartite graph is a graph G on two sets of vertices, V_1 and V_2 , such that every edge has one endpoint in V_1 and one endpoint in V_2 . We sometimes write $G = (V_1, V_2; E)$ for this situation. For example:



The edges in the above example consist of all the lines, whether solid or dotted; the solid lines form a matching.

The *bipartite maximum matching* problem is to find a matching in a given bipartite graph G , which has the maximum number of edges among all matchings in G .

Name:

ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (a) (6 pts total) Prove that a maximum matching in a bipartite graph $G = (V_1, V_2; E)$ has size at most $\min\{|V_1|, |V_2|\}$.

Name:
ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (b) (6 pts total) Show how you can use an algorithm for max-flow to solve bipartite maximum matching on undirected simple bipartite graphs. That is, give an algorithm which, given an undirected simple bipartite graph $G = (V_1, V_2; E)$, (1) constructs a directed, weighted graph G' (which need not be bipartite) with weights $w : E(G') \rightarrow \mathbb{R}$ as well as two vertices $s, t \in V(G')$, (2) solves max-flow for $(G', w), s, t$, and (3) uses the solution for max-flow to find the maximum matching in G . Your algorithm may use any **max-flow** algorithm as a subroutine.

Name:

ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (c) (7 pts total) Show the weighted graph constructed by your algorithm on the example bipartite graph above.

Name:

ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

2. In the review session for his Deep Wizarding class, Dumbledore reminds everyone that the logical definition of NP requires that the number of *bits* in the witness w is polynomial in the number of bits of the input n . That is, $|w| = \text{poly}(n)$. With a smile, he says that in beginner wizarding, witnesses are usually only logarithmic in size, i.e., $|w| = O(\log n)$.

Name:

ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (a) (7 pts total) Because you are a model student, Dumbledore asks you to prove, in front of the whole class, that any such property is in the complexity class P.

Name:

ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (b) (6 pts total) Well done, Dumbledore says. Now, explain why the logical definition of NP implies that any problem in NP can be solved by an exponential-time algorithm.

Name:

ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (c) (6 pts total) Dumbledore then asks the class: “So, is NP a good formalization of the notion of problems that can be solved by brute force? Discuss.” Give arguments for both possible answers.

Name:
ID:

CSCI 3104
Problem Set 10

Profs. Grochow & Layer
Spring 2019, CU-Boulder

3. (20 pts) Recall that the *MergeSort* algorithm is a sorting algorithm that takes $\Theta(n \log n)$ time and $\Theta(n)$ space. In this problem, you will implement and instrument MergeSort, then perform a numerical experiment that verifies this asymptotic analysis. There are two functions and one experiment to do this.
- (i) `MergeSort(A,n)` takes as input an unordered array A , of length n , and returns both an in-place sorted version of A and a count t of the number of atomic operations performed by `MergeSort`.
- (ii) `randomArray(n)` takes as input an integer n and returns an array A such that for each $0 \leq i < n$, $A[i]$ is a uniformly random integer between 1 and n . (It is okay if A is a random permutation of the first n positive integers.)

CSCI 3104
Problem Set 10

Name:
ID:
Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (a) (10 pts total) From scratch, implement the functions `MergeSort` and `randomArray`. You may not use any library functions that make their implementation trivial. You may use a library function that implements a pseudorandom number generator in order to implement `randomArray`.
Submit a paragraph that explains how you instrumented `MergeSort`, i.e., explain which operations you counted and why these are the correct ones to count.

CSCI 3104
Problem Set 10

Name:

ID:

Profs. Grochow & Layer
Spring 2019, CU-Boulder

- (b) (10 pts total) For each of $n = \{2^4, 2^5, \dots, 2^{26}, 2^{27}\}$, run `MergeSort(randomArray(n), n)` five times and record the tuple $(n, \langle t \rangle)$, where $\langle t \rangle$ is the average number of operations your function counted over the five repetitions. Use whatever software you like to make a line plot of these 24 data points; overlay on your data a function of the form $T(n) = A n \log n$, where you choose the constant A so that the function is close to your data.

Hint 1: To increase the aesthetics, use a log-log plot.

Hint 2: Make sure that your *MergeSort* implementation uses only two arrays of length n to do its work. (For instance, don't do recursion with pass-by-value.)