Midterm 1

THURSDAY, FEBRUARY 20, FROM 8–10PM AT WTHR 320

Name: _____

PUID: _____

- You are awesome.
- We trust you, and in particular, we expect you to uphold Purdue's honor code (https://www.purdue.edu/odos/osrr/honor-pledge/about.html) during the exam.
- Please note that there is scratch paper available in the back which we encourage you to use for all the problems. Please do not rip the scratch paper out (which disrupts the scanning process afterwards). You can write on the back of a page. If you run out of space for a word problem, please continue on a page of scratch paper and leave a sentence such as "go to page [...]" so we know where to find the rest of your solution.
- For the multiple choice, please write your answer on the line, and try to write darkly, to help the automated scanning.
- You may invoke algorithms from class or homework as a *black box*. You may also invoke the solution from one problem / subproblem as a black box when solving another problem / subproblem (even if you don't get the first one totally correct). You still need to prove your use of the black box is correct; see "For 'change the input' type solutions" below. (You *cannot* use things that we have *not* explicitly discussed or covered as a black box.)
- There are 19 short answer / multiple choice problems and 4 word problems. The short answer / multiple choice are each worth the same amount and together are worth one-third (1/3) of the final score. The word problems are worth two-thirds (2/3) of the final score.
- Each of the four word problems are scored out of 10 points.[1] Out of your four scores, the weight of the lowest one is dropped. (e.g., if you got 10, 9, 6, and 4, your total would be $10 + 9 + 6 = 25$ out of 30 possible.)
- We encourage you to read all the problems all the way through. If you get stuck on one, don't worry. Skip it and come back to it later! (This applies to both the short answer and the word problems.)
- Please keep in mind that many problems can be solved by "identifying the right subproblems" or "changing the input", which may be counterintuitive.
- On any problem, you may write "IDK" and get 25%. This includes short answer and multiple choice problems. You may also "IDK" for subproblems or parts of word problems (e.g., "IDK the running time").

[1] If a word problem has subproblems, each subproblem will be labeled with their respective contribution, out of 10.

- Some of the multiple choice / short answer problems describe a computational problem and asks you to either Identify the smallest real values $k, \ell \geq 0$ such that there is a $O\Big(m^k n^\ell \operatorname{poly}(\log(m), \log(n))\Big)$ time algorithm for the problem, and fill in for the values of k and ℓ where indicated.

  Here $\operatorname{poly}(\log(m), \log(n))$ allows for any fixed polynomial of $\log(m)$ or $\log(n)$. (That is, we don't care about polylogarithmic factors.) For graph problems, m refers to the number of edges and n refers to the number of vertices, and you may assume that $n \leq m \leq n^2$. (e.g., $O(m+n)$ becomes $O(m)$.) For other types of problems, m and n are inferred from the problem description.[2]

- **For "identify the subproblem" type solutions**, including all dynamic programming and divide-and-conquer algorithms (and recursive algorithms in general), we require the following parts explicitly.

  1. Recursive spec (i.e., a sentence) definition of the subproblem.
  2. How to extract the final solution from the subproblems.
  3. Psuedocode implementation of the recursive spec.
  4. The total running time of your algorithm. If you are using dynamic programming, explicitly mention that you are using DP/caching solutions to obtain the claimed running time. For other recursive algorithms including divide-and-conquer, you should state the running time recurrence that you are solving.

  We will treat your recursive spec as an induction hypothesis to justify the correctness of the algorithm, and no explicit proof is otherwise required. This means your recursive spec needs to be sufficient to complete a proof by induction. (Some justification / comments might help the grader's understanding in the event of partial credit.)

- **For "change the input" type solutions,** where a problem X is reduced to problem Y, you should describe an algorithm where an algorithm for problem Y is used as a black box to solve problem X. The correctness often involves establishing an "if and only if". (E.g., the answer to decision problem X is true iff the answer to the instance of problem Y is true, or there is an (s,t)-walk with an even number of edges of length L iff there is an $(s_{\text{EVEN}}, t_{\text{EVEN}})$-walk of length L in the auxiliary graph.) Justify both directions explicitly (even if you think its obvious). For polynomial time algorithms you also need to analyze the running time.

- **All other types of solutions** require a proof.

- In general, any explanation and justification for your algorithm will help the grader's understanding in the event of partial credit.

- We're not expecting anyone to be perfect. Have fun!

[2] If m does not appear in the problem, then that means $k = 0$ and we will only check the answer for ℓ. If n does not appear in the problem, then that means $\ell = 0$ and we will only check the answer for m.

The following are running times for some algorithms from class. For graph algorithms, we let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges.

- Binary search: $O(\log n)$ to search through a sorted array of n items.
- Merge-sort, heap-sort, inversion count, incomparable pairs, closest pair in $\mathbb{R}^2$: $O(n \log n)$ time.
- Selection: $O(n)$.
- The nth discrete Fourier transform, and the nth discrete conjugate Fourier transform: $O(n \log n)$ time.
- (Min-) Heaps support `min`, `insert` and `delete-min` in $O(\log n)$ time in a heap with n items. One can build a new heap over n items in $O(n)$ time.
- BFS, DFS, topological sort, strongly connected components: $O(m + n)$.
- Dijkstra's algorithm: $O(m + n \log n)$.
- Bellman-Ford[3]: $O(mn)$.
- Floyd-Warshall[4]: $O\left(n^3\right)$.
- Johnson's algorithm[5]: $O\left(mn + n^2 \log n\right)$

[3]The dynamic programming algorithm for single source shortest paths with real-valued edge lengths.

[4]The dynamic programming algorithm for all pairs shortest paths with real-valued edge lengths and no negative cycles.

[5]The alogrithm for all pairs shortest paths using potentials to reweight the edges.