CS 124 Homework 3: Spring 2021

Your name:

Collaborators:

No. of late days used on previous psets:
No. of late days used after including this pset:

Homework is due Wednesday 2021-03-03 at 11:59pm ET. You are allowed up to **twelve** (college)/**forty** (extension school) late days through the semester, but the number of late days you take on each assignment must be a nonnegative integer at most **two** (college)/**four** (extension school).

Try to make your answers as clear and concise as possible; style will count in your grades. Be sure to read and know the collaboration policy in the course syllabus. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan in your results to turn them in.

For all homework problems where you are asked to design or give an algorithm, you must prove the correctness of your algorithm and prove the best upper bound that you can give for the running time. Generally better running times will get better credit; generally exponential time algorithms (unless specifically asked for) will receive no or little credit. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Answers that consist solely of pseudocode will receive little or not credit. Again, try to make your answers clear and concise.

1. Solve the following two problems using heaps. (No credit if you're not using heaps!)

   (a) **(12 points)** Give an $O(n \log k)$ algorithm to merge k sorted lists with n total elements into one sorted list.

   (b) **(12 points)** Say that a list of numbers is k-close to sorted if each number in the list is less than k positions from its actual place in the sorted order. (Hence, a list that is 1-close to sorted is actually sorted.) Give an $O(n \log k)$ algorithm for sorting a list of n numbers that is k-close to sorted.

2. **(10 points)** You are given a graph $G = (V, E)$ and a minimum spanning tree T. The weight of one of the edges in $E \setminus T$ (that is, not in T) is decreased. Give a linear time algorithm that determines whether the MST changes and, if so, returns a new MST.

3. **(15 points)** Suppose you are given a weighted graph $G = (V, E)$ and an edge $e \in E$. You are asked whether there exists a minimum spanning tree of G containing e. Give a linear time algorithm for the problem. (If it helps, first consider the case where all edge weights are distinct – this will get you most of the credit – and then try the more general case.)

4. Recall that we ask you to put each problem set solution on its own page. For purposes of this problem, assume that each solution may be on at most one page. You've solved the n problems of a problem set, with values v_1, v_2, …, v_n, but you only have available a set of k pieces of paper p_1, p_2, …, p_k of various shapes and sizes. Each page can fit only some problem solutions: for each $i \in [n]$ and $j \in [k]$, you know either that solution i can fit on page p_j (in which case we say $a_{i,j} = 1$) or not ($a_{i,j} = 0$). (Note that it's possible that solution i fits on page j but not j' and solution i' fits on page j' but not j.) You want to submit the set of solutions that maximizes your score.

   (a) **(5 points)** Consider the following "greediest" algorithm: starting with the highest-value problem and working down, assign a solution to an arbitrary empty page that can fit it if there is one, and throw it out if not. Show that the greediest algorithm doesn't always maximize your score. (To show this, you should give an example instance of the problem—an n, k, set of problem values, and a description of which problems fit on which pages—and show a solution produced by the greedy algorithm and another solution that gives you a better score.)

   (b) **(20 points)** To design a less greedy algorithm, we need a subroutine. Suppose we have an assignment of a set S of solutions to pages, and we're given a new solution t. We'd like to know whether it's possible to assign all the solutions $S \cup \{t\}$ to pages (possibly rearranging which solutions are assigned to which pages) and output an assignment if so or "impossible" if not. Give an algorithm to do so. (Hint: First, given a solution, find all the other solutions whose pages it could commandeer, making a graph on the solutions. Search that graph, looking for a solution that could be assigned to a new page.)

(c) **(15 points)** Consider the following "less greedy" algorithm: starting with the highest-value problem and working down, attempt to assign a solution to a page using the subroutine from the previous problem part. If there is a way, do it; if not, throw out the solution. Show that the "less greedy" algorithm finds the optimal score. (Hint: Suppose you have two assignments of solutions to pages: an "optimal" one and one found by the "less greedy" algorithm. It suffices (why?) to show that you can match each solution s assigned to a page in only the "optimal" assignment with a solution s' at least as valuable as s that's assigned to a page in only the "less greedy" algorithm. To find such a matching, try swapping whether pages hold the solution from the "optimal" algorithm or the solution they held in the "less greedy" algorithm, starting with the page that held s.)

5. Consider the following scheduling problem: we have two machines, and a set of jobs $j_1, j_2, j_3, \ldots, j_n$ that we have to process. To process a job, we place it on a machine; each machine can only process one job at a time. Each job j_i has an associated running time r_i. The load on the machine is the sum of the running times of the jobs placed on it. The goal is to minimize the completion time, which is the maximum load over all machines.

   Suppose we adopt a greedy algorithm: each job j_i is put on the machine with the minimum load after the first $i-1$ jobs. (Ties can be broken arbitrarily.)

   (a) **(5 points)** For all $n > 3$, give an instance of this problem for which the output of the greedy algorithm is a factor of $3/2$ away from the best possible placement of jobs.

   (b) **(15 points)** Prove that the greedy algorithm always yields a completion time within a factor of 3/2 of the best possible placement of jobs. (Hint: Think of the best possible placement of jobs. Even for the best placement, the completion time is at least as big as the biggest job, and at least as big as half the sum of the jobs. You may want to use both of these facts.)

   (c) **(10 points)** Suppose now instead of 2 machines we have m machines. Prove the best upper bound you can on the ratio of the performance of the greedy solution to the optimal solution, as a function of m?

   (d) **(5 points)** Give a family of examples (that is, one for each m – if they are very similar, it will be easier to write down!) where the factor separating the optimal and the greedy solutions is as large as you can make it.

6. **(0 points, optional)**[1] Consider the following generalization of binary heaps, called d-heaps: instead of each vertex having up to two children, each vertex has up to d children, for some integer $d \geq 2$. What's the running time of each of the following operations, in terms of d and the size n of the heap?

   (a) delete-max()

   (b) insert(x, value)

   (c) promote(x, newvalue)

   The last operation, promote(x, newvalue), updates the value of x to $newvalue$, which is guaranteed to be greater than x's old value. (Alternately, if it's less, the operation has no effect.)

[1] We won't use this question for grades. Try it if you're interested. It may be used for recommendations/TF hiring.