Introduction to Analysis of Algorithms	Final Exam Information and Practice
CS4820/5820 Fall 2021	"Due" December 16 2021

Rooms, Time, and Review for Final

- Final Exam is Thursday, December 16th 7pm in Statler 185 (possibly moved to Barton) and online.
- A google form will be posted on Tuesday morning, where you can indicate whether you want to take the exam online. Not filling in the form by Tuesday midnight means you are taking the exam in-person.
- We'll have review sessions by TAs on the 10th and the 14th. Times and locations will be announced on Ed.
- Solutions to the practice questions will be released on Monday, Dec 13, but you can use office hours to discuss solution with the TAs any time before the prelim.

Guidelines for the Final

The final is cumulative, covering all topics from the beginning of the course. It will be closed book and closed notes.

In particular, the topics include

- From Prelim 1: (To review material, see the review packets from the prelim and homeworks 1-5. New practice questions are included below.)
  - Stable matching from Chapter 1.
  - Greedy algorithms from Chapter 4, including unweighted interval scheduling (and the greedy-stays-ahead proof technique), scheduling to minimize lateness (and the exchange-argument proof technique), minimum spanning tree (Kruskal, Prim).
  - Dynamic programming from Chapter 6, including weighted interval scheduling, knapsack, segmented least squares, sequence alignment (a.k.a. edit distance), Bellman-Ford shortest path algorithm.
  - Divide and conquer from Chapter 5, including integer multiplication, quickselect (see also randomized algorithms), finding the closest pair of points in the plane.

  - Randomized Algorithms from Section 13, including expected linear-time median finding, finding the closest pair of points in the plane (using randomization and hashing)

- From Prelim 2: (To review material, see the review packets from the prelim and homeworks 6-9. New practice questions are included below.)
  - Network flows from Chapter 7, including the Ford-Fulkerson max flow algorithm, equivalence of max flows and min cuts, and network flow applications.
  - NP-completeness from Chapter 8, including reductions and the problems SAT, Independent Set, Vertex Cover, Hamiltonian Path/Cycle, TSP, Subset Sum, and problems from the homeworks/exams.

- Since Prelim 2: (see summary and review questions below, as well as homeworks 10.)
  - Computability: Turing Machines, the non-decidability of the halting problem, reductions to show problems are undecidable, Rice's Theorem.
  - Approximation Algorithms from Chapter 11 and the notes posted, including the greedy knapsack 2-approximation algorithm (see notes on the web site), arbitrarily good approximation for the knapsack problem, simple 2-approximation algorithm for unweighted vertex cover and linear programming rounding 2-approximation algorithm for weighted vertex cover.

- Similar to the prelim exams, you will not be tested on sections in the afore-mentioned chapters if we did not discuss them in lecture (the course outline (linked from canvas) shows which sections we did cover). Some topics we covered briefly, such as hashing, and you will not be tested on such topics.

Post-Prelim 2 Topics

The questions on computability will test your understanding of Turing Machines, and your ability to decide which problems are computable, and which are not. Topics you need to know about:

- Turing Machine definition, equivalence of different models (e.g. multiple and single tapes, two-way and one-way infinite tapes),
- definition of recursive and recursively enumerable sets,
- definition of decidability, showing a problem is decidable,
- the Halting Problem, and empty-string-acceptance problem,

- proving a problem is undecidable by reduction from an undecidable problem (e.g. the Halting Problem) or using Rice's Theorem.

The questions on approximation algorithms will test your ability to do the following:

- showing an algorithm may give a solution that is much worse than optimum (we gave examples of this for knapsack and vertex cover on November 29 and December 3)
- analyze greedy methods that give a good approximation algorithm: such as knapsack (see handout), unweighted vertex cover (lecture December 3)
- use of dynamic programming to get an arbitrarily good approximation for knapsakc (Section 11.8)
- use linear programming and rounding, (Section 11.6)

How to study

Here are a couple of recommendations on how to review:

- Read the lecture notes or book chapter on a specific topic, and take notes to summarize the key concepts. Go take a walk, and try to tell yourself (or a friend) the problem, algorithm, and analysis that you just read *from memory*.

- Try solving all the homework and exam problems again *without looking at the solutions*, including your own solutions.
- Review the comments the TAs left for your solutions, and the solutions posted on canvas.
- Review the rubric for the homework/exam problems. These are brief summaries of what you need to solve the problem; familiarizing yourself with them helps to get a good understanding of what it is your answer should include and how to allocate your time when writing your answers.

Practice Questions on Approximation Algorithms

1. Consider the **Move-It-Quik** truck problem: items arrive one at-a-time and the truck weight capacity is K pounds. The loading site only has room for one truck to be loaded at a time. Rather than planning which items should go in which truck in

advance, the movers just load one item at a time in whatever order they find items to pack, disregarding their weight, until the next item wouldn't fit. Then, they move on to the next truck.

Consider the pseudocode for a greedy truck-loading :

TRUCK LOADING:
$c = K$ is the capacity remaining for the current truck
$m = 1$ is the number of trucks used so far.
While there is an item i with weight w_i left to load:
    If $c \geq w_i$:
        // The item fits; load item i on the current truck
        Set $c := c - w_i$
    Else:
        // The item won't fit; send off the current truck
        // and get a new one to load item i
        Set $m+ = 1$
        Set $c = K - w_i$
    Endif
Endwhile
Return m as the total number of trucks used.

Suppose at the end of one of these moving jobs, n items with weights $w_1, w_2, \ldots, w_n$ are packed up in that order.

(a) Prove that the number of trucks used by this algorithm, m, is at most a factor of 2 larger than the minimum number possible for taking the n items. That is, if OPT denotes that minimum number of trucks possible by carefully ordering the n items, show that $m \leq 2OPT$. **Solution:** First, we upper-bound how well the optimal packing OPT can get. We know that the optimal solution cannot pack trucks beyond their weight capacity. Let $W = \sum_i w_i$: any optimal solution must use at least $\lceil \frac{W}{K} \rceil$ trucks to pack all the items.

Next, we show that for any pair of consecutive trucks, the combined load of the two trucks must be greater than K. Suppose by way of contradiction that truck j has some load l_j and truck $j+1$ has load l_{j+1}, where $l_j + l_{j+1} \leq K$. In this case, all of the items in these two trucks together could have fit on the same truck, so this implies a contradiction: there was still room to load the first item on truck $j+1$ on truck j. Thus their combined load must be more than K.

Because every pair of trucks is more than half full, we can consider this summed over all possible trucks: if the number of trucks used is even, we get that on average (averaging truck $2j-1$ and truck $2j$), every truck must have at a load of more than $K/2$. This means that the total number of trucks used must be

less than $W/\frac{K}{2} = 2W/K$, i.e. $2W/K > ALG \geq OPT \geq W/K$. Rewriting, we can use this to find that $\frac{ALG}{OPT} < \frac{2W/K}{W/K} = 2$, or that the ratio of trucks used between the mover's algorithm and the optimal solution is upper-bounded by 2. If the number of trucks uses is odd, say $ALG = 2m + 1$, the first $2m$ trucks have total weight greater than mK, which means we need at least $m + 1$ trucks, so we get $OPT \geq m + 1$ and $ALG = 2m + 1$. In this case we also have $\frac{2m+1}{m+1} = ALG/OPT \leq 2 = \frac{2m+2}{m+1}$ as well.

(b) In part (a), you showed that if OPT is the number of trucks used in the optimal solution, $2OPT$ is an upper bound on the number of trucks used by this algorithm. In general, an upper bound is called "tight" if there is an instance where the upper-bound is actually achieved: in this case, we know there can't possibly be a smaller upper bound. Show that this 2-optimal bound is tight: describe how to construct a sequence of objects for any value of OPT that will ensure that while the optimal algorithm uses OPT trucks, the TRUCK LOADING algorithm uses at least $2OPT - 2$ trucks. *An example that works for some single value of OPT but not for any OPT can get 2/4 points.*

Solution: Consider a loading problem with truck capacity $K = 1$ and the alternating sequence $w_1 = 1/2, w_2 = 2/n, w_3 = 1/2, \ldots w_{n-1} = 1/2, w_n = 2/n$. This gives us $n/2$ items of weight $1/2$, which can be loaded into $n/4$ trucks, completely filling the trucks. The remaining $n/2$ items all have weight $2/n$ and thus can fit in a truck together. In total, we use $OPT = n/4 + 1$ trucks in this optimal solution, and every truck is full. The algorithm, however, will always load every even item with weight $2/n$ on with the previous odd item of weight $1/2$, which will force the movers to start a new truck every other item and using a total of $n/2 = 2OPT - 2$ trucks.

(c) Now, assume that the system has no items that are too heavy. Concretely, assume that $w_i \leq K/3$ for all i. Prove that in this case the above algorithm satisfies a better bound of $m \leq 1.5OPT + 1$.

Solution: By the assumption that $w_i \leq K/3$, we won't start a new truck unless the previous one has at least $2K/3$ weight. So with m trucks the total weight must be more than $(m-1)2K/3$, and hence we need more than $OPT > \frac{2}{3}(m-1)$ trucks, implying that $m < \frac{3}{2}OPT + 1$.

2. Consider the following expert selection problem. For a movie project you need a bunch of different experts, from legal experts, through experts on the computer science company cultures, etc. You have a set S of possible experts that you can hire, but some of them are quite expensive. You need help with a set A of areas. Expert i needs a fee of c_i to be hired. The good news is that some of the experts have more than one expertise, though no single expert has more than two skills on your list of areas A. You would like to hire a subset I of the experts so that you have an expert

on the payroll in all areas in the set A, while minimizing the total cost $\sum_{i \in I} c_i$.

Give a 2-approximation algorithm for this problem. Your algorithm should run in polynomial time, and find a set of experts that covers all required topics, and cost at most 2 times the minimum possible.

Solution: The algorithm is greedy, and repeatedly adds the expert i that is cheapest and covers at least one new area that is not yet covered. Running time is $O(n \log n)$ with n experts, starts with sorting the fees, and keep an array of all the needed skills so we can mark the skills not yet covered.

This is a 2-approximation for the following reason: For each skill a let $i^*(a)$ be the expert providing this skill in the optimal solution, and $i(a)$ be the expert covering this skill in the greedy algorithm. Now consider the experts in order of the greedy algorithm. For each skill a, we have $c_{i(a)} \leq c_{i^*(a)}$ by the order elements are chosen. The algorithm's total cost is at most $\sum_a c_{i(a)}$ (can be less if some expert chosen covers multiple skills). If I^* denotes the optimal solution, we have

$$\sum_a c_{i(a)} \leq \sum_a c_{i^*(a)} \leq 2 \sum_{i \in I^*} c_i$$

where the last inequality follows as any expert $i \in I^*$ can cover at most 2 skills.

Practice Questions for the Final

The following sample questions are designed to give you a few extra questions to practice. They are designed to cover most of the content of the course (but refer to the prelim practice materials as well because not everything is represented here!)

Note: Some topics not covered here are Divide-and-Conquer, Randomized Algorithms, Stable Matching Problem. That does not mean those are not covered on the final exam!

1. **Short answer.** Each of these questions asks for a true/false or decidable/undecidable answer and a short explanation. Follow the instructions for each question. **Detailed proofs are not required in this section.**

   (a) Consider an undirected graph $G = (V, E)$ with integer costs $c_e \geq 1$ for each edge $e \in E$, and a minimum-cost spanning tree T in this graph. Suppose we subtract 1 from the cost of each edge: $c'_e = c_e - 1$ on all edges. *True or False*: does T remain the minimum cost spanning tree. If true, give a brief explanation; if false, give a counter example.

Solution: True. The choice of the MST only depends on the sorted order of the edge weights by Kruskal's algorithm, and decreasing all costs by 1 doesn't change the order.

(b) Consider an undirected graph $G = (V, E)$ with integer costs $c_e \geq 1$ for each edge $e \in E$, and a shortest path P between two nodes $s, t \in V$ in this graph. Suppose we subtract 1 from the cost of each edge: $c'_e = c_e - 1$ on all edges. *True or False*: does P remain the shortest path from s to t? If true, give a brief explanation; if false, give a counter example.

Solution: False. Consider a four node graph with nodes s, v, w, t and costs $c_{sv} = 2, c_{vw} = 2, c_{wt} = 2$ and $c_{st} = 5$. Now the shortest path is the one edge direct path $P = (s, t)$. Subtracting one from the cost of each edge changes the shortest path to $P' = (s, v, w, t)$ which now has cost 3.

(c) Consider a directed network G with source s and sink t, integer capacities $c_e > 0$ on the edges e of G, a maximum value flow f from s to t. Given an edge $\hat{e}$ in a minimum capacity (s, t)-cut (A, B), and $0 < f(\hat{e}) = c_{\hat{e}}$. *True or false:* decreasing the capacity of edge $\hat{e}$ necessarily decreases the value of the max flow. Please explain your answer.

Solution: True, the max flow value equals the min cut value, decreasing the capacity by 1 decreases the min cut and hence must also decrease the max flow.

(d) Consider a directed network G with source s and sink t, integer capacities $c_e > 0$ on the edges of G, and a maximum value flow f with value v_f from s to t. Given an edge $\hat{e}$ in a minimum capacity (s, t)-cut (A, B), and $0 < f(\hat{e}) = c_{\hat{e}}$. *True or false:* increasing the capacity of edge $\hat{e}$ necessarily increases the value of the max flow. Please explain your answer.

Solution: Not true, The max flow value equals the min cut value, but there can be many min-cuts. Increasing the capacity by 1 increases this min cut but there can be other cuts with the same capacity.

2. For the three problems below provide one of the following two answers:

   - There is a polynomial-time algorithm for the problem; or
   - the problem is NP-complete.

   Explain why your answer is correct.

   (a) You need to select a set of k judges for a competition. Unfortunately, all judges know some of the participants socially, which represents a form of conflict of interest. Each possible judge i had to declare the list L_i of all the participants with whom he or she has a mild conflict of interest. There are so many conflicts

that it turned out to be impossible to avoid them all. One option for how to deal with them is as follows.

The JUDGE SELECTION PROBLEMasks whether it is possible to select k judges so that for any participant j at most one of the selected judges has a conflict of interest with j. **Solution:** NP-complete. It is clearly in NP; checking if a set of k judges have this property is easy.

To prove it is NP-complete, we show that INDEPENDENT SET $\leq$ JUDGE SELECTION PROBLEM. Input to INDEPENDENT SET is a graph $G = (V, E)$ with k. We let the nodes V be the judges, and the edges be the participants. each judge has a conflict of interest with the edges adjacent to the node. There are k judges with the property claimed, if and only if G has an independent set of size k.

- If G has an independent set of size k, the judges corresponding to the nodes satisfy the required property, as no two of them are adjacent to a shared edge.
- If there is a set of judges with the property above, then the set of nodes corresponding to the judges forms an independent set.

Construction is done in linear time.

(b) You are given a project consisting of a set A of n tasks, and a team of n members to complete the tasks. However, not all team members can do all tasks. For each team member i you are given a list $S_i \subset A$ representing the set of tasks this member can take care of, but any one member can be assigned to at most one task. The TEAM ARRANGEMENT PROBLEM asks if there is a way to assign members to the tasks and have all tasks completed. **Solution:** This problem is in P. Set this up as a bipartite matching problem with a node for each team member and each task, and an edge between member i and task j if $j \in S_i$. A perfect matching M in this graph gives a solution to TEAM ARRANGEMENT (by assigning task j to member i if $(i, j) \in M$) and a solution to TEAM ARRANGEMENT gives a perfect matching (by adding (i, j) to the matching if i is assigned to task j). (You can also reduce this to maximum flow, of course!)

(c) You need to select $k > n$ committee members from a group of $2n$ representatives. Exactly n of the representatives are democrats and n of the representatives are republicans. Unfortunately, there are some pairs of representatives of opposite parties who cannot stand each other, and those cannot serve on the committee together; you have a list E of these pairs, where each pair (u, v) in E consists of a democrat u and a republican v.

The COMMITTEE SELECTION PROBLEM asks whether a committee of size k exists that contains no pair of representatives in the set E.

Solution: This problem is in P. Set this up as a minimum cut problem, with a node for each representative, plus a source s and sink t. There is an edge

from s to the nodes for the democratic representatives with capacity 1, and edge from the nodes for the republicans to t with capacity 1. Also, there is an edge from democrat u to republican v if $(u, v) \in E$ with capacity ∞. Let (A, B) be a minimum s-t cut in this graph with capacity $c(A, B)$; we select the democrats whose nodes are in A, and the republicans whose nodes are in B. The construction of the graph ensures that we cannot have democrat $u \in A$ and republican $v \in B$ if $(u, v) \in E$ (because then the capacity of the cut would be infinite). So this gives a committee with no pair of representatives in the set E, and the size of the committee is equal to $2n - c(A, B)$. We can similarly argue that every such committee of size k gives a finite s-t cut with capacity $2n - k$. So a committee of size k exists if and only if the minimum s-t cut capacity is at most $2n - k$.

3. Determine whether the following decision problem is decidable or not. Explain why your answer is correct.

   Given a Turing Machine M and two input strings x and y. Does M output the same result for x and y? That is, is it the case that, if M accepts, rejects or doesn't halt on x, does it do the same for y?

   Solution: It is undecidable. We show that if we could decide this problem, we could also decide the halting problem, which is a contradiction.

   Reduction: Given an input M, x to the Halting Problem, we create a Turing Machine M' which does the following on input y': it checks if the input string y' on its tape is empty, and if yes, M' goes into an infinite loop. If the input on M''s tape is not empty, M' erases y' from its its tape, writes x on it and runs M on x. If this halts, then M' halts and accepts.

   Now to decide whether M halts on x, we consider the problem whether M' agrees on input x' being the empty string, and input y' being any non-empty string.

   Correctness: Observe that M' doesn't halt on halts on input x', because x' is the empty string. For input y', M' will halt and accept if M halts on x and if M doesn't halt on x, then M' doesn't halt either. So M' agrees on x' and y' if and only if M doesn't halt on x.

   So if it was decidable whether the machine M' outputs the same result on x' and y', then we could use this to decide whether M halts on x; a contradiction to the fact that the Halting Problem is undecidable.

4. You are helping a group of astronomers observe a subset of possible important events in the sky. Events occur at times $t_1, t_2, \ldots, t_n$. They will be hiring students to observe the events. Students can be hired for different time intervals. They are given a list of options, each with an interval and a cost: student j costs c_j and can cover events

in the range $[t_{s_j}, t_{f_j}]$. It is OK to have observers overlap in time. They would like to hire observers to record all events as cheaply as possible.

In the example below, assume that there are 8 events.

	student 1	student 2	student 3	student 4	student 5
interval	$[t_1, t_5]$	$[t_1, t_3]$	$[t_2, t_6]$	$[t_4, t_8]$	$[t_6, t_8]$
cost	10	2	5	4	3

Possible solutions are to hire students 1 and 5 for a cost of 10+3=13, or hire students 2, 3 and 5 for the cost of 2+5+3=10, or can hire students 2 and 4 for the cost of 2+4=6, which is the cheapest.

(a) Consider the following greedy algorithm for the problem.

Let C = 0
While not all events observed
    Let t_i be the earliest not yet observed event
    Hire the student who can observe t_i, and is as cheap as possible.
    Add the cost of this hire to C
Endwhile
Output C, and the solution found.

Show that this algorithm does not always find the best solution.

Solution: Swapping the cost of the students 3 and 4 in the above example, would make the algorithm hire students 2, 3, and 5 for a total cost of 2+4+3=9 (as student 4 is now cheaper than student 3). However, the best solution is still students 2 and 5 (for the cost of 2+5=7).

(b) Give a greedy algorithm that finds an optimal solution if all costs are the same. Prove your algorithm is correct.

Solution:

Let C = 0
While not all events observed
    Let t_i be the earliest not yet observed event
    Hire the student j who can observe t_i, and for whom t_{f_j} is as large as possible.
Endwhile
Output C, and the solution found.

This is the same problem as the loci-covering problem from question 2(a) on Homework 2. You can prove correctness using greedy-stays-ahead. Let $j_1, j_2, \ldots, j_a$

be the students selected by the greedy algorithm (in the order they were selected), and let $j_1^*, j_2^*, \ldots, j_o^*$ be the students in an optimal solution (ordered by increasing right endpoint of their time interval, so that $t_{f_{j_i^*}} \leq t_{f_{j^*_{i+1}}}$).

Claim: If the first i students in the optimal solution cover an event, then so do the first i students in the greedy solution, for $i = 1, \ldots, \min\{o, a\}$.

Proof: By induction. The base case holds because greedy selects the first student so that they cover the first event, and then have the right endpoint of their interval be as large as possible. The optimal solution must also cover the first event, so the right endpoint of j_1^*'s interval is at most as large as the right endpoint of j_1's interval. So that means that the first student in the greedy solution observes every event observed by the first student in the optimal solution. For the inductive step, suppose the claim holds for i. Then the next unobserved event for the greedy algorithm is also not observed by the first i students in the optimal solution. Greedy chooses the next student j_{i+1} so their interval covers this event, and so that the right endpoint of their interval is as large as possible. The optimal solution must also cover this event, so it has a student among $j_{i+1}^*, \ldots, j_o^*$ whose right endpoint is at most as large as the right endpoint of j_{i+1}'s interval. So that means the right endpoint of j_{i+1}^*'s interval is at most as large as the right endpoint of j_{i+1}'s interval. So any event not yet covered by $j_1, \ldots, j_i$ that is covered by j_{i+1}^* is also covered by j_{i+1}.

(c) Give a polynomial-time algorithm that finds the set of students that can observe all events $\{t_1, t_2, \ldots, t_n\}$ as cheaply as possible. Your algorithm only needs to return the cost, and not the set.

*You must **explain how your algorithm works** (for example, explain the meanings of all variables other than loop counters) and **analyze its running time**, but you do not need to provide a proof of correctness.*

Solution: We will use dynamic programming. We will compute the minimum cost to cover the events $\{t_1, t_2, \ldots, t_i\}$ for all i. We'll call this cost $Opt(i)$. Let S_i denote the set of students who can cover event i.

Let $Opt(0) = 0$
For $i = 1, \ldots, n$
    $Opt(i) = \min_{j \in S_i} c_j + Opt(s_j - 1)$
Endfor
Output $Opt(n)$.

Running time. The algorithm computes Opt values, for n values, and for each, there may be up to m possible students j to consider, for a total running time of $O(mn)$.

Alternate Solutions. Of course, it is just as good to set on the "backwards", with $Opt(i)$ denoting the optimal solution for times $t_i, t_{i+1}, \ldots, t_n$.

It is great if they set this up as a shortest path (Bellman-Ford or actually Dijkstra in this case) application. Nodes are the times to observe nodes, say with an artificial start node t_0, so we have $n + 1$ nodes. Will look for a path from 0 to n. Edges go backwards in time for 0 cost, and forwards in time corresponding to each student. For a student available for interval $[t_{s_j}, t_{f_j}]$ we add an edge from $s_j - 1$ to t_j at the cost of c_j. A path in this graph from 0 to some node i corresponds to a sequence of students hired that cover events $t_1, t_2, \ldots, t_i$ with backwards edges corresponding to overlapping student times.

5. For the following two subproblems provide one of the following two answers:

   - There is a polynomial-time algorithm for the problem; or
   - the problem is NP-complete.

   Fully explain why your answer is correct.

   (a) Given a complete directed graph on a set of n nodes V and costs $c_{ij} \geq 0$ for traveling from node i to node j for every pair of nodes, and a number $\gamma > 0$. Is there a simple cycle C through all n nodes with total cost at most γ?

   Solution: Problem is NP-complete. To prove it is NP-complete, we show it is in NP: given a cycle, we can verify it goes through all the nodes, and its total cost less than γ by adding the costs. It is NP-complete, as its harder than the Hamiltonian cycle problem. For solving the HC problem by this problem, use $c_{ij} = 0$ if (i, j) is an edge, and 1 if it's not, then cycle through all the nodes with cost $\gamma = 1$ is exactly a Hamiltonion cycle.

   (b) Given a complete directed graph on a set of n nodes V and costs $c_{ij} \geq 0$ for traveling from node i to node j for every pair of nodes, and a pair of nodes s and t, and a number $\gamma > 0$, is there a simple cycle C through edge (s, t) with cost at most γ.

   Solution: Solvable in polynomial time. Delete edge (i, j), and find the shortest path from j to i in the remaining graph. The cycle exists if an only if the length of the path is at most $\gamma - c_{ij}$.

6. Given a Turing Machine M and an input x. Is it decidable whether M on input x halts in a number of steps divisible by 5? Prove your answer correct.

   Solution: The problem is not decidable by reduction from the Halting problem. Suppose that we want to decide if some arbitrary program M halts on some input x. We can construct a program M' that given some input string x, operates in the same way as M except that for every step of M the program M' performs four additional

dummy steps (e.g., move back and forth on the tape twice). By construction, if M halts after k steps on some input x, then M' halts after $5k$ steps on the same input x. Hence, deciding problem for the program M' and some input x is equivalent to solving the Halting problem for the program M and input x. (Otherwise, we could use the decider for this problem in order to decide the Halting problem, which would contradict the fact that the Halting problem is undecidable).

7. Consider the knapsack problem: given n items with values $v_1, \ldots, v_n$, weights $w_1, \ldots, w_n$ and a knapsack size W, the problem is to select a subset I of items so that $\sum_{i \in I} w_i \leq W$ and the total value $\sum_{i \in I} v_i$ is as high as possible.

   In class we showed that the better of two greedy algorithms is a 2-approximation algorithm. The second of these algorithms was the following

   > (Greedy B) Sort items in decreasing order of their value density $d_i = v_i/w_i$, and in this order take items as long as they fit (that is, satisfy the $\sum_{i \in I} w_i \leq W$ bound).

   While we have seen that this algorithm can output arbitrarily bad solutions, we have used it as part of a 2-approximation algorithm. In this problem we consider greedy B alone in the special case when all items are small relative to the knapsack size W: assume $w_i \leq W/3$ for all items i.

   (a) Let V_B denote the value of a solution found by algorithm Greedy B. Show that the first item k that didn't fit in the knapsack has value at most $v_k \leq V_B/2$.

   Solution: Let $r_k = v_k/w_k$, the value density of the first item that didn't fit. Note that all items previous to k have density at least r_k, and fill the knapsack to at least till $2W/3$ as $w_k \leq W/3$ and it no longer fits. So we get that $V_B \geq r_k \cdot 2W/3$, while $v_k = r_k w_k \leq r_k W/3$.

   Combining these two inequalities we get that $v_k \leq r_k W/3 \leq V_B/2$.

   (b) Show (using part (a)) that the Greedy B algorithm above is a 1.5 approximation in this case. That is, the ratio of the optimum value to the value of the solution found by algorithm (i) is at most 1.5.

   Solution: Recall from analysis in the notes that $V_{OPT} \leq V_B + v_k$. From (a) we know that $v_k \leq V_B/2$, so $V_{OPT} \leq 1.5 V_B$, as claimed.

   (c) Show that the bound of 1.5 proved in part (b) is asymptotically tight. For the nth example let r_n be the ratio of the optimum value to the value produced by Greedy B. You were asked to prove in part (b) that $r_n \leq 1.5$ for all examples. Give a sequence of examples such that $\lim_{n \to \infty} r_n = 1.5$, and show that the ratio of your sequence is converging to 1.5.

   A single example when greedy is not optimal (rather than a class of examples with $r_n \to 1.5$) will get at least 2 point.

Solution: Consider the following class of examples each with 4 items with values $v_1 = v_2 = 0.5 + 2\epsilon$, $v_3 = v_4 = v_5 = 1$, and $w_1 = w_2 = 0.5 + \epsilon$, $w_3 = w_4 = w_5 = v_3 = v_4 = v_5 1$, and $W = 3$. The optimum is to take items 3, 4, and 5 for a total value $Opt = 3$, while the algorithm takes only 1, 2, and 3 for a value of $2 + 4\epsilon$ as items 1 and 2 have the highest value density. The sequence of items is defined by making ϵ go to 0.