

# CS375 Project 3

Due Dec. 16th 10PM ET

First off, all credit to Eric Aaron for the creation of the majority of this project.

## 1 Improve Time Efficiency

Your team will choose one of the eight problems from project 1 on which to give a presentation and focus for the other parts of this project. In the remainder of this project assignment, I'll use the variable name  $P$  to refer to the problem on which your team chooses to focus (just so that there's a name for it!).

Your primary objective will be to improve upon the time efficiency of that exhaustive search solution.

In general—not just for this project, but in general for algorithm design—there are a few ways to think of improving upon an algorithm's time efficiency. They include:

1. You can make it somewhat faster with small(-ish) changes that streamline but do not substantially redesign the algorithm or change its asymptotic complexity class.
2. You can make it **much** faster with a substantial redesign—perhaps even giving a new algorithm altogether—which might even improve its asymptotic complexity class.
3. You can focus on *special cases* of the input that can be solved very efficiently. That is, instead of coming up with a more efficient solution that works for *all* possible inputs in the problem's specification, come up with a solution that is much more efficient—i.e., a better asymptotic complexity class—for *some* of the possible inputs.

For example, imagine that you've solved a problem with this input specification:

**Input:**  $k$ , a positive integer

But you then come up with a much faster algorithm that will work only when  $k$  is an even number. That faster algorithm does not meet the original problem specifications, but it is an improvement in the special case of an even integer input.

**NOTE:** Smaller input sizes are rarely considered special cases in this sense. If a suggested "improvement" is "It's the same algorithm, but it's super-fast on small inputs!", that likely is not actually a useful improvement. Please see me if your team has a suggested improvement that requires looking at only very small inputs, to confirm that it's worthy of including in your project write-up / presentation!

For this project, your team will come up with improvements upon the efficiency of your original exhaustive search algorithm for problem  $P$ . You are encouraged to think of improvements in terms

of the three ways listed above, and you are especially encouraged not to restrict yourself to only the first or second of them—special-case improvements can be very helpful, even on very simple-seeming special cases! **One important constraint: you can not use an algorithm that we have covered in class (eg. the Dynamic Programming solution for SubsetSum).**

There are no fixed criteria for this project about exactly how much you must speed up your initial exhaustive search algorithm—or, with the “special cases” approach, how broadly applicable your improvements might be, to cover as many cases as possible. Your team can also propose **up to three** algorithms or modifications that improve the efficiency of your original algorithm for problem P, although submitting three improvements is not *necessarily* better than one. Ultimately, your team will earn more credit on this part of the project for improvements that show more depth of thought about the problem and its solutions, achieve greater time efficiency, apply more broadly to possible inputs, and are more thoroughly and helpfully analyzed and described—but in some cases, one substantial improvement might achieve that better than three small ones. (I hope these criteria make intuitive sense to you. As always, please feel free to ask me questions!)

**Hint:** You are advised **not** to try to come up with polynomial-time solutions for the general cases of any of the eight problems presented. You might, however, create polynomial time solutions for special cases, which you could choose to include in your write-up / presentation! If you do come up with a polynomial-time solution for any of the eight problems, don’t tell anyone and come see me :)

**To Submit for this part of the project** In your presentation for this project, your team should present not only your original exhaustive search algorithm for problem P but also your improvements to it. In the write-up that accompanies your presentation, please include the following:

- A full description of your exhaustive search algorithm for problem P
- English descriptions of each improvement. Each description should include 1–2 sentences about how you came up with the ideas behind that proposed improvement.
- Pseudocode showing what each improvement does.
- A concise, high-level complexity analysis showing how much each improvement actually improved the time efficiency of the original exhaustive search algorithm. Although you do not need to use formal definitions of asymptotic complexity in your analysis, you might want to use some part of them—in particular, if your improvement doesn’t change the asymptotic complexity class, you might describe its improvement in terms of a lower *leading constant* for complexity analysis.
- An argument of correctness for whatever your algorithm’s improvements are.

More details about your presentation and its write-up are in Section 3 below.

## 2 Reduction

Sometimes, we can incorporate solutions to previously solved problems as subroutines in an algorithm we’re designing. For this project, your team will do that in a specific way: You’ll *reduce* another problem to problem P

Informally, in general, reducing problem *A* to problem *B* means creating an algorithm that, if you plugged in a subroutine that solved problem *B*, would immediately be able to solve problem

$A$ —the algorithm reduces the task of solving problem  $A$  to the task of solving problem  $B$ . We'll call such an algorithm a *reduction* from  $A$  to  $B$ .

As a concrete, very simple example consider problem  $A$  with these specifications:

**Input:** List  $L = [c_1, c_2, \dots, c_n]$  of numbers.

**Output:** *True* if the first element of  $L$ ,  $c_1$ , is 375; *False* otherwise.

And consider problem  $B$  with these specifications:

**Input:** List  $M = [d_1, d_2, \dots, d_k]$  of numbers.

**Output:** *True* if the last element of  $M$ ,  $d_k$ , is 375; *False* otherwise.

For these problems, a reduction from  $A$  to  $B$  would take some input  $L$ —remember, this reduction is an algorithm for some problem  $A$ , so it has to take an input intended for  $A$ —and create a new list  $M = [c_1]$  that contains only the first element of  $L$ . Then, the reduction would use  $M$  as input to a subroutine that solved problem  $B$ . If that subroutine returned *True*, that would mean  $c_1 = 375$  (**do you see why?**), which in turn means that the first element of  $L$  is 375, so the reduction solving  $A$  should return *True*. On the other hand, if that subroutine for  $B$  returned *False* on input of list  $M$ , that would mean  $c_1$  is not 375, so the reduction solving  $A$  should return *False*.

There are other possible reductions that could have worked—for instance,  $M$  could instead have been created as  $M = [c_n, \dots, c_1]$ , built from all elements of  $L$  in reverse order, rather than just a list with one element; the rest of the reduction would have been exactly the same. For this project, your team just needs to present one correct reduction. (Please note again that this is a very simple example—your work for this project may not be quite this simple!)

For this project, your team will create a reduction *from* problem  $P$  *to* any other of the eight problems given in the project. Here is the specific way to think of it for this assignment:

- An all-powerful creature has bestowed upon you a wonderful gift of magic! They've given you a magic subroutine for problem  $P$ —that will solve it in  $O(1)$  time! To use one this magic subroutine, just give it some input that matches the input specifications for the problem, and then it will instantly (well, in  $O(1)$  time) give you a correct *True* or *False* answer for that input to that problem!
- There's a catch, though: You only get to use it *once*. After that, the subroutine disappears!
- Your task is to write a new algorithm to solve another problem  $Q$  that makes use of this magic subroutine. It shouldn't be an exhaustive search algorithm anymore; the magic subroutine can do the hard work of exhaustive search! In fact, if you are careful in choosing and employing the magic subroutine, you could even come up with a *polynomial time* algorithm for problem  $Q$ !

Your reduction should thus transform any possible input  $q$  for problem  $Q$  into an input  $p$  for problem  $P$ , such that when you get a *True* or *False* answer about input  $p$  for  $P$ , you can use that to come up with a correct *True* or *False* answer about input  $q$  for problem  $Q$ .

For this part of the project, in addition to creating the reduction as described above, you will also do all of the following:

- Analyze the time complexity of your reduction, under the assumption that the solution for  $Q$  comes in  $O(1)$  time. **For maximal credit, your reduction should be in polynomial time— $O(n^k)$  for some  $k$ —but don't worry about what constant  $k$  you use.** Every correct polynomial time reduction will be equally good for this project!

- Include the reduction in your presentation (see Section 3 below). Be sure to include problems Q and P you're reducing from and to, a short description of your reduction algorithm (pseudocode is not required, though you may include it if you think it helps your presentation!), a short explanation of correctness, and a short complexity analysis.
- Describe the reduction in the write-up document accompanying the presentation. Again, state the problems P and Q you're reducing from and to, and give a helpfully complete description of the reduction—an English description is required; pseudocode is optional—along with a short explanation of correctness and your complexity analysis of the reduction.

Your explanation of correctness does not need to use loop invariants—your reduction will probably be straightforward enough that loop invariants aren't required. If your team thinks loop invariants might be a good idea to use, though, feel free to do so, or feel free to ask me about it!

**Important note:** A correct reduction must be **exactly consistent** with the Input / Output specifications for both P and Q. For full credit, your explanation of correctness should explicitly refer to those specifications.

Your presentation and accompanying write-up document will contain all the work you need to submit for this part of the project. Please make sure your reduction is clearly and concisely described in the presentation itself, and all helpful details for understanding the reduction are included in the write-up!

### 3 Create and Give a Presentation

At this point, your team has done a lot of work on problem P. Let's hear about it!

Your team will give a technical presentation about the algorithms you've created for problem P. For your presentation, create slides (in PowerPoint, Google Slides, or some other application of your choice) and take 10–15 minutes to present all of the material needed. Presentations that are too long or too short may not receive full credit (too short often indicates that some important material was not well presented; too long often indicates that additional preparation would have resulted in a more effective talk), so it is recommended that you target a 16–18 minute presentation. If you think your talk will be much longer or shorter than that, please discuss that with me—I will be happy to help you find a good balance for your presentation.

The default expectation is that you will record your presentation as a screen recording in Zoom. If you believe another option would be better for your team, **please see me about it as soon as possible!**

This is a team presentation and a class assignment, so ideally, the entire team would learn about all parts of the topic being presented, and not only would the workload be balanced among team members, but it would also *appear* balanced to viewers. For that reason, your presentation should consist of each person presenting for roughly 2 minutes at a time, followed by a different teammate—so, for example, in a roughly 16 minute presentation, each person on a four-person team would take two non-consecutive shifts of presenting for roughly 2 minutes each. This structure might require a conceptual topic to be split among multiple individuals in the presentation, due to the impositions of time limits, but that's part of the pedagogical benefit of this—it encourages more people to engage with more different parts of the topic being presented. **Important note:** *Presentations not following this structure will not receive full credit* for this assignment. If there are questions about

what's expected in terms of the division among teammates of time spent presenting, please let me know!

Here are some things you should include in your presentation (not necessarily in this order!):

**Your exhaustive search algorithm** For your exhaustive search algorithm, please include:

- An accessible description of the problem  $P$  you solved.
- A high-level summary of your algorithm and how it works.
- A short example that you step through, to give your audience a sense of what problem  $P$  is and how your algorithm works. It may be appropriate to only step through a part of an example instead of an entire one, but you should do enough to fully illuminate how your algorithm works for your audience.
- A complexity argument of the algorithm, including what the worst-case and best-case complexities are, and how much space is needed beyond the original input.

**Your improvements to your exhaustive search algorithm** For each of the improvements you're presenting, please include:

- A high-level description of the improved algorithm.
- A short example that you step through, to give your audience a sense how the improvement differs from the original exhaustive search algorithm. Once again, you may not need to go through a full example, but you should do enough to illuminate the differences in your improved algorithm.
- Pseudocode of the improved algorithm, along with a correctness argument. As noted above, you need not use loop invariants for this (though you could if you thought it was necessary), but you do need to give a concise and convincing correctness argument. You can refer to your original exhaustive search algorithm and its correctness without re-explaining them.
- A complexity argument of the algorithm, including what the worst-case and best-case complexities are, and how much space is needed beyond the original input.
- A comparison of the complexity of your improved algorithm with that of your exhaustive search algorithm.

**Your reduction algorithm** For your reduction, please include:

- An accessible description of the problem  $Q$  you're reducing from.
- A high-level description of the reduction algorithm that solves  $Q$ , including how it uses the subroutine for  $P$  in that solution.
- A short example, to show your audience what the reduction does—transforming input to  $Q$  into input to  $P$ , and using output from the subroutine for  $Q$  to get a correct answer on the input to  $P$ .
- Pseudocode of the reduction, along with a correctness argument. As noted in Section 2 above, this will involve referring to the specifications of  $P$  and  $Q$ . You need not use loop invariants for this; just give a concise and convincing correctness argument that the reduction meets specifications and solves  $Q$  correctly (assuming the subroutine solves  $P$  correctly). You can refer to your original exhaustive search algorithm and its correctness without re-explaining them.

- Worst-case time complexity and space complexity arguments for the reduction, assuming the subroutine for P has  $O(1)$  time and space complexity. (Magic!)

You should assume that your audience is at the level of CS students who are familiar with asymptotic complexity and loop invariants but are not yet experts with them. For example, assume that your audience knows a set of size  $n$  has  $2^n$  subsets, a list of length  $n$  has  $n!$  permutations, and all about the relative growth rates of functions used in asymptotic complexity (including knowing what “polynomial time” means). You should also assume your audience has no previous knowledge of your algorithms or any problems involved, and they may not quickly grasp any subtleties.

**Your Accompanying Write-Up** In addition to the presentation itself, your team will create an accompanying write-up document, which should enable your audience to understand the highlights of your presentation even if they do not see your talk. This document must be typed (submitted in PDF) and contain some important details that you may not have time to include in your talk itself. (For example, some small but important details of complexity arguments might not fit in the 10–15 minutes of your talk, but they can be included in the write-up.) For full credit, your write-up must be polished, well formatted for a professional technical presentation, easy to read, and free of grammatical errors.

Please see individual sections above for more information about details to include in your presentation write-up about the exhaustive search, improvements, and reduction algorithms.

Depending on the margins / font size / etc. of your document, your write-up should probably be 7–10 pages in length. Please keep it as concise as it can be while still containing all relevant information. If your write-up is running longer or shorter than that range, please see me to check whether it contains unneeded material, or too little material; write-ups that are much too long or too short are not maximally effective and may not receive full credit.

Please feel free to ask me questions about them, if you’d like!