CompSci 260P Fall 2022 Project 4: Divide and Conquer Analysis
Due Friday December 2, 2022, at 11:59 PM, although late submissions are allowed. Note that the late period includes the final exam. Be aware of this. The time this is due is also different from typical due times in this class.

Getting Started

Choosing a project partner (or even two)

You have the option to work with a second person for this assignment. If you do so, I expect you to work via pair programming. That is, you may not split the assignment, such as by having one person implement the algorithms while the other person writes the report. I reserve the right to ask any non-empty subset of the project partners about the implementation and adjust the score accordingly if I believe they do not understand what was submitted.

Similarly, any academic dishonesty arising from a group will be treated as an offense by all involved.

All partners must fill out the following survey to register the partnership. It is not enough for one to do so. Be sure to include your UCINetID and your partner’s UCINetID. Failure to do so may cause some of you to not get credit. Make sure you know what your UCINetID is. It is probably not just numbers and it almost certainly does not include the @ symbol.

All partnerships must be registered by 5pm Irvine time on Monday, November 28, 2022

https://docs.google.com/forms/d/e/1FAIpQLScaezqSAh4qR5c-kk8znUPVxIwpTtu16ZHdFDpKeesAbygu9A/viewform

There are two parts of this project: programming and an experimental analysis of the algorithm. In particular you will implement two algorithms for finding the closest pair of a set of points:
- A brute force algorithm that takes quadratic time
- A divide and conquer algorithm, although with a parameter that will cause it to switch to brute force when the subproblem is under a threshold (which the parameter represents).

Your analysis will focus on trying to discern:
- How small should a subproblem be before brute force has a performance advantage over divide and conquer? Recall that the D&C is faster asymptotically but incurs overhead from recursion that may not make sense when there are, say, six points.
- How large of a subproblem is there a noticeable difference between the two algorithms?

Your report will focus on explaining how you found those two values.
Programming Requirements

For this project, you are going to implement the aforementioned two algorithms. You are also required to create a project on gitlab called project4. Note the spelling and case sensitivity: that is vital for my script to collect what you submit. If you are working as a partnership, the person indicated as the submitter must be the one who owns the project, similar to how this was dealt with in projects two and three.

Like project 3, I will not be requiring C++. However:

- Your program must be in C++, Go, Java, or Python3.
- You must include a README.txt file at the top level of your project 4 with instructions for how I can run your program to test it for correctness if I choose to do so. In order to get credit for the assignment, it must be very clear how I can do the following in the VM:
  - If you are using a language that is not installed on the VM already, you must include command-line instructions for how I would install a compiler for your language of choice in a new VM.
  - When I run your program, I must be able to easily run it in such a way that it runs one of the two algorithms. It must be clear to me how to run it:
    - With just the brute force algorithm
    - With the Divide and Conquer algorithm, while specifying a size under which brute force is used instead on recursive calls.
  - Your program, given a set of 500 points, must be able to compute the closest pair within three minutes on the same computer I use to grade other projects. Note that this does not mean your experiment should limit itself to inputs of size 500. Your README.txt file may specify a parameter for the size under which brute force is used to achieve this time constraint, if needed.
- Your code should be reasonably well written and commented.

For any language you choose, you may use standard libraries as appropriate, unless there is one that makes solving this problem trivial. I am unaware of any such part of the library. The standard priority queue class in C++ lives in <queue> for some reason.

You are explicitly permitted to use C++ standard library container classes (std::vector, std::set, etc). You are welcome to ask anything you want about these libraries, or to look up material about them online. Information about how to use an explicitly-permitted library may always be shared among classmates, but refrain from telling one another how you solved a problem in the assignment with them. For example, answering “how do I check if an element is in a std::set?” is great and encouraged, while answering “what did you use std::set for in your project?” is not.

A good reference for the STL container classes (such as those listed above, including std::map) is [http://www.cplusplus.com/reference/map/map/](http://www.cplusplus.com/reference/map/map/).
Remember that the purpose of this project isn’t to find an implementation of the closest pair algorithms, but rather to code them yourself. Submitting work that isn’t yours (for any reason) is a decidedly bad idea and one of the very few ways to do poorly in this class. If I find that you submitted code you found online, your grade will be worse than if you hadn’t turned in this project at all.

**The Report**

In addition to having a programming portion, you (and your group, if applicable) need to put together a short report. In this report, you will perform an experimental analysis of the algorithm(s) you implemented. You will need to convert your report, when done, to a PDF for upload to GradeScope. Your report must be typed. Your group for the report must be the same set of people with whom you did the programming if you had a group. **However**, if you worked alone on the programming portion, you may elect to partner with someone else (or a total of three people, yourself included) who also worked alone to produce a single report. In this case, you should run the same experiments with the code each of you produced and, if there are differences in behavior, discuss this in the report. You do not need to modify your code based on this, although you are allowed to do so. Different approaches to generating the approximate tour might produce different answers that are correct, after all.

If you worked alone on the programming but with a partner for the report, you do not need to (nor should you) submit the form above. One of you should submit the report to GradeScope and list both of you as authors/partners for it.

Your report should address the following things:

- How did you test your code to ensure that it behaves the way you intended?
  - For this project, I **explicitly authorize** the sharing of test cases between groups for purposes of testing the **correctness of the closest pair of points functions**. That is, you may generate a set of points, find the closest pair of points, and share the set with a classmate(s) to see if they find the same set with their code. **If you do this**, include that in your report about how you tested your code. Tell me who you shared yours with and what you found. **It is okay if you find that one of you has incorrect code and fixed it accordingly. State that in the project report.**

- How did you test the code for correctness?
  - Describe in general how you came up with test cases and how you ensured that they were reasonable, both in terms of correctness and in terms of expected running time.
  - Describe the purpose of 2-3 of them. You can refer to them by name -- you don’t need to copy the specifics of the test case to your report (in fact, please don’t copy/paste the specifics). Just make sure they’re in the collected code you submit (if you are using the provided code, I believe they would be in the /gtest/ folder).
- How did you measure the crossover point? How close to the optimal crossover point do you think you found?
  - It is very possible that different groups find different values. That’s okay, perhaps even expected.

A side note: Fall 2022 is the first time I am requiring a report for this project, or for that matter, this project. I am mostly interested in seeing how you went about this project and if you got the “big picture” ideas about experimental evaluation that I have tried to convey. I (Michael) plan to read each report myself, and the report is just as much a feedback mechanism for me as it is for you. Please treat it seriously and not as a last-minute addendum to programming. I plan to make a note of which reports are particularly insightful, which will help me add to any letters of recommendation I write for the authors in the future.

When you submit the report to GradeScope, it will ask you to “tag” the pages that the “response” is on. Please tag each page.

Report limit is five pages, must be typed, reasonable font size. If you require more pages, please let me know -- exceptions will be granted, although I’d prefer that you try to limit it to five pages, as I will have many to read in a short period of time.

**Deliverables**

Ensure that you have committed and pushed your code to your project4 repo.

**Can I submit after the deadline?**
Yes, it is possible, subject to the late work policy for this course, which is described in the section titled Late work in the course reference and syllabus. The short story is, the longer after the deadline, the less credit you get, and after 99 hours post-deadline, no credit. Note that the first exam is less than 99 hours post-deadline and you cannot move the exam in order to make time to do this project.

If you do not submit on time, your submission time is the later of when you submit the late form and when you submit the project 4 report on GradeScope.