CS 161 **Problem Set 2**
Fall 2025 **Due:** Friday October 10 at 11:59pm on Gradescope.

Style guide and expectations: Please see the top of the "Homework" page on the course webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards.

Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Collaboration policy: You may do the HW in groups of size up to three. Please submit one HW for your whole group on Gradescope. (Note that there is an option to submit as a group). See the "Policies" section of the course website for more on the collaboration policy.

LLM policy: Check out the course webpage for best practices on how to productively use LLMs on homework, if you use them at all.

Exercises

We recommend you do the exercises on your own before collaborating with your group. The point is to check your understanding.

---

1. **(6 pt.)** Compute the *best* (that is, the smallest) upper-bound that you can on the following recurrence relations. You may use any method we've seen in class (Master Theorem, Substitution Method, algebra, etc.)

   (a) **(2 pt.)** $T(n) = 2T(\lfloor n/2 \rfloor) + O(\sqrt{n})$, where $T(0) = T(1) = 1$.

   (b) **(2 pt.)** $T(n) = T(n-2) + 1$, where $T(0) = T(1) = 1$.

   (c) **(2 pt.)** $T(n) = 6T(\lfloor n/4 \rfloor) + n^2$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$.

   [**We are expecting:** *For each part, a statement of your answer of the form "$T(n) = O(\_\_\_\_)$", as well as a short justification. You don't need to give a rigorous proof, but your justification should be convincing to the grader. If you use the master theorem, state the values of a, b, d.*]

2. **(3 pt.)** Consider the following algorithm, which takes as input an array A:

```
def printStuff(A):
    n = len(A)
    if n <= 4:
        return
    for i in range(n):
        print(A[i])

    printStuff(A[:(n//3)]) # recurse on the first floor(n/3) elements of A
    printStuff(A[(n//3):2*(n//3)]) # recurse on the second floor(n/3) elements of A
    return
```

What is the asymptotic running time of `printStuff` on an array of length n?

[**We are expecting:** *The best answer you can give of the from "The running time of* ***printStuff****(n) is $O(\_\_\_\_)$, and a short explanation, including a clear statement of the relevant recurrence relation.*]

3. **(4 pt.)** [**Fishing.**] Plucky the Pedantic Penguin is fishing. Plucky catches n fish, but plans to keep only the k largest fish and to throw the rest back. Plucky has already named all of the fish, and has measured their length and entered it into an array F of length n. For example, F might look like this:

$$F = \begin{bmatrix} \text{(Frederick the Fish, 14.2in)} \\ \text{(Fabiola the Fish, 10in)} \\ \text{(Farid the Fish, 12.35in)} \\ \vdots \\ \text{(Felix the Fish, 6.234523in)} \\ \text{(Finlay the Fish, 6.234524in)} \end{bmatrix}$$

Give an $O(n)$-time deterministic algorithm that takes F and k as inputs and returns a list of the names of the k largest fish. You may assume that the lengths of the fish are distinct.

Note: Your algorithm should run in time $O(n)$ even if k is a function of n. For example, if Plucky wants to keep the largest $k = n/2$ fish, your algorithm should still run in time $O(n)$.

[**We are expecting:** *Pseudocode* ***AND*** *a short English description of your algorithm. You may (and, hint, may want to...) invoke algorithms we have seen in class. You do not need to justify why your algorithm is correct or its running time.*]

Problems

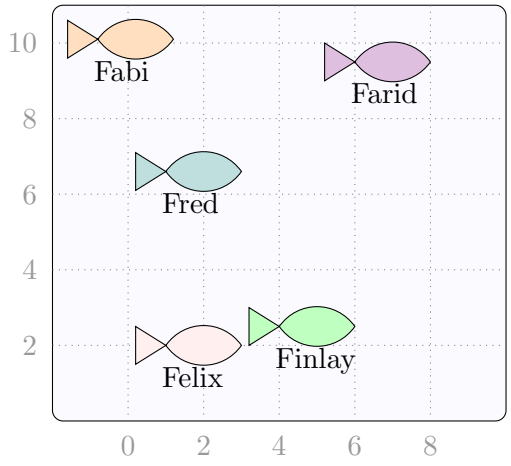
---

4. **(17 pt.) (Fish Friends)**

   You are given a list F of fish and (two-dimensional) coordinate locations in a pond, in feet. For example, the input might look like this:

$$F = \begin{bmatrix} \text{(Frederick the Fish, 2, 6.6)} \\ \text{(Fabiola the Fish, 0.2, 10.1)} \\ \text{(Farid the Fish, 7, 9.5)} \\ \vdots \\ \text{(Felix the Fish, 2, 2)} \\ \text{(Finlay the Fish, 5, 2.5)} \end{bmatrix}$$

   which corresponds to the two-dimensional layout:



   Your goal is to return the distance between the closest pair of fish. In the example above, Felix and Finlay are closest, at distance $\sqrt{(5-2)^2 + (2.5-2)^2} = 3.04$ feet, so on input F, your algorithm should return 3.04.

   Note: The fact that the fish have names is not important for solving this problem, we just included them to make the example more clear. If you like, you can assume that F is an array of coordinate pairs (x, y) with no names, rather than an array of triples (name, x, y). If you drop the names, state so clearly in your solution.

   (a) **(5 pt.)** Design a divide-and-conquer algorithm that determines whether there exists a pair of fish within 1 foot of each other, on an input list F with n fish. Your algorithm should run in time $O(n \log n)$. You may assume that n is a power of two if you like.

   [**Hint:** *Depending on how you do the problem, the following may eventually be helpful: Suppose that you have a bunch of fish, all of distance at least d from each other. Then at most 8 of these fish can be contained in any $2d \times d$ rectangle. (You can take this as a fact, but it might be fun to convince yourself of this).*]

[**We are expecting:** *Pseudocode and a clear English description of what it is doing. You do not need to prove correctness or justify the running time (yet). You may use any algorithm we have seen in class as a black box.*]

(b) **(4 pt.)** Design a divide-and-conquer algorithm that returns the distance between the closest pair of fish, on an input list F with n fish. Your algorithm should run in time $O(n \log n)$. You may assume that n is a power of two if you like.

[**Hint:** *We do not expect that it will be useful to use your solution from part (a) as a black box, but we do expect that similar ideas will be helpful for this part.*]

[**We are expecting:** *Pseudocode and a clear English description of what it is doing. Your response should be self-contained, meaning that it should not only state what is different than your part (a) solution, the whole pseudocode should be in this part. You do not need to prove correctness or justify the running time (yet). You may use any algorithm we have seen in class as a black box.*]

(c) **(4 pt.)** Explain why your algorithm in part (b) is correct. You don't need to give a formal proof, but your explanation should be convincing to the grader.

[**We are expecting:** *A clear explanation. If you don't think that your algorithm is correct, you can say what's wrong for partial credit.*]

(d) **(4 pt.)** Explain why the running time of your algorithm in part (b) is $O(n \log n)$.

[**We are expecting:** *A short explanation. Writing down a recurrence relation, explaining why it is relevant, and explaining why it has the desired solution is sufficient. If you don't think that the running time of your algorithm is $O(n \log n)$, say what you think the running time is and justify that instead for partial credit on this part.*]