CS325: Analysis of Algorithms, Fall 2025

Group Assignment 1*

Due: Tue, 10/14/25

Homework Policy:

1. Students should work on group assignments in groups of preferably three people. Each group submits to CANVAS a zip file that includes their source code and their *typeset* report. Specifically, for this assignment your zipped folder should contain two files named `assignment1.pdf`, and `assignment1.py`. One submission from each group is sufficient.
2. The goal of the homework assignments is for you to learn solving algorithmic problems. So, I recommend spending sufficient time thinking about problems individually before discussing them with your friends.
3. You are allowed to discuss the problems with other groups, and you are allowed to use other resources, but you *must* cite them. Also, you must write everything in your own words, copying verbatim is plagiarism.
4. Algorithms should be explained in plain english. You can use pseudocodes if it helps your explanation, but the grader will not try to understand a complicated pseudocode.
5. More items might be added to this list. ☺

You are a visitor at a political convention with n delegates; each delegate is a member of exactly one political party (there may be more than 2 parties). It is impossible to tell which political party any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the same party by introducing them to each other. Members of the same political party always greet each other with smiles and friendly handshakes; members of dffierent parties always greet each other with angry stares and insults. Suppose more than half of the delegates belong to the same political party.

Your assignment is to create an efficient algorithm that finds out the size of the majority party. The efficiency of your algorithms is measured in terms of the number of pairs of delegates that you introduce to each other. We expect that you need about $O(n \log n)$ introductions.

Report (60%). In your report, include the description of your algorithm, a running time analysis (running time here is the number of pairs of delegates that you introduce to each other, you can still use big-O notation), and show the correctness of your algorithm. Algorithms should be explained in plain english. You can use pseudo-code if it helps your explanation, but the grader will not try to understand complicated pseudocode.

*The problem is from Jeff Erickson's lecture notes. Looking into similar problems from his book chapter on recursion is recommended.

Code (40%). You will implement and submit your algorithm in python3 using the template provided in the `assignment1.py` file. You need to implement the function *majority_party_size*. This function has two parameters: n and *same_party*. The integer $1 \leq n \leq 1000$ is the total number of delegates. The delegates have indices $0, 1, 2, \ldots, n-1$. You can call the function *same_party*(i, j) for integers $0 \leq i, j < n$, to check if i, j belong to the same party. The function *majority_party_size* should return the size of the largest party size, exactly one integer (Note you can assume that a majority party exists).

```
'''
    This file contains the template for Assignment1.  You should fill the
    function <majority_party_size>.  The function, recieves two inputs:
      (1) n: the number of delegates in the room, and
      (2) same_party(int, int): a function that can be used to check if two members are
      in the same party.

    Your algorithm in the end should return the size of the largest party, assuming
    it is larger than n/2.

    I will use <python3> to run this code.
'''


def majority_party_size(n, same_party):
    '''
        n (int): number of people in the room.
        same_party (func(int, int)): This function determines if two delegates
            belong to the same party.  same_party(i,j) is True if i, j belong to
            the same party (in particular, if i = j), False, otherwise.

        return: The number of delegates in the majority party.  You can assume
            more than half of the delegates belong to the same party.
    '''

    # Replace the following line with your code.
    return 1

```

We provide a file *assignment1_helper.py* which contains the code to test your algorithm and a small set of test cases. The file *assignment1_helper.py* will read test cases from files and calls your function *majority_party_size*. It also contains the implementation of the function *same_party*. It writes an output file, which contains two integers: the return value of your function, and the number of times it calls *same_party*. Our scripts will use the content of this file to grade your code. Each test case is worth two points: one for correctly identifying the size of the majority party, and one for meeting the query budget. For the query budget, we impose a hard cutoff of $3n \log n$ queries – you receive the point if you use fewer than this many queries, if you exceed the budget you don't.

Note that you should not change anything in the *assignment1_helper.py* file, except possibly its last line if you want to try different inputs for test cases.

Tests We test your algorithm against multiple test cases. For each test case, to receive the points your algorithm must return the correct value for the number of delegates in the largest party. Further, it should not use too many calls of the *same_party* function. Note that we are after an $O(n \log n)$ time algorithm. You may receive extra points if you can find out the size of the largest party with a particularly small number of calls to the *same_party* function compared to other codes.