* Administrative
  + Team Name:
    - * Group 73
  + Team Members:
    - * Andrew Rubinstein, Sebastian Robalino, Zach Szabo
  + Github user names:
    - * drewrubinstein, sebrobalino, Zach-Szabo
  + Link to GitHub repo:
    - * <https://github.com/drewrubinstein/P3>
  + Link to Video demo (See below)
    - * <https://www.youtube.com/watch?v=83RgTWnhoVg&ab_channel=DrewRubinstein>
* Extended and Refined Proposal [Suggested 2 Pages]
  + Problem: What problem are we trying to solve?
    - * We want to create a program that enables users to find new movies depending on a movie they input into the UI.
  + Motivation: Why is this a problem?
    - * Streaming services are more popular now than ever, so with all of the options available out there, it is a struggle for people to find movies that they would like. People are always looking for new movies to watch and this can help users find movies based on previous movies they

like

* + Features implemented:
    - * We will give back a list of 10 similar movies based on the movie input. We will output this separately for Dijktras and BFS based on the data set.
  + Description of data
    - * CSV files with columns, Movie ID, Movie Title (Year), and its genre, as well as ratings from multiple users, relevance scores, and tags for each movie.
      * <https://grouplens.org/datasets/movielens/25m/>
  + Tools/Languages/APIs/Libraries used
    - * QT Creator, C++
  + Algorithms implemented
    - * Dijkstra’s, BFS
  + Additional Data Structures/Algorithms used
    - * Adjacency List
  + Distribution of Responsibility and Roles: Who did what?
    - * Drew: Back-end code (Dijkstra's, BFS, and CSV Parsing), Video, Video editing
      * Sebastian: Program Analysis/Report, Video, main.cpp
      * Zach: Porting back end code to GUI, main.cpp, QTCreator, video
* Analysis [Suggested 1.5 Pages]
  + Any changes the group made after the proposal? The rationale behind the changes.
    - * In the original proposal we said that we were also going to take in actor’s names as an input. However, this was changed after we could not find a suitable dataset that also included the actors in each movie.
  + Big O worst case time complexity analysis of the major functions/features you implemented
    - * **parseMovies(const string &filePath)**:
        + Reads each line of a csv file, each line representing a movie, and creates a MovieNode object for each, storing them in an unordered\_map using the Movie IDs as keys. Splitting string (O(length of string)) + Insertion in unordered map (O(1)). Time complexity: **O(n \* k)**, n is number of movies, k is length of string
      * **parseRatings(const string &filePath, unordered\_map<int, MovieNode> &movies)**:
        + Reads each line from a CSV file, changing the MovieNode object in the unordered\_map for each movie as it iterates through. It skips invalid entries and updates the movie’s ranking data if the MovieID exists in the map. The function goes through each line independently, and for each line ‘m’, it would lookup the MovieID in the map and if it exists it updates the map. Time Complexity: **O(m\* k)**, m is number of ratings, k is average processing time per line
      * **buildGraph()**:
        + Constructs a graph were each movie is a vertex and the edges between the movies are similarity scores. For each movie, it is compared with every other movie. This leads to 2 loops through the list of movies. Time Complexity: **O(n^2)**, n is number of movies
      * **similarity(MovieNode movie1, MovieNode movie2)**:
        + Calculates the similarity between 2 movies based on genre overlap and rating differential. Time Complexity: **O(n log n)**, n is number of genres.
      * **findSimilarMoviesBFS(const MovieNode& startMovie)**:
        + Uses BFS from a given start movie to find the 10 most similar movies. Time Complexity: **O( V \* E )**, V is number of vertices, E is number of edges
      * **findSimilarMoviesDijkstra(const MovieNode& startMovie)**:
        + Uses Dijkstra’s Algorithm from a given start movie, to find 10 similar movies, taking similarity score into consideration for path cost. Time Complexity: **O((v+e) log v)**, v is number vertices, e is number of edges (min priority queue implemented)
* Reflection [Suggested 1-1.5 Page]
  + As a group, how was the overall experience for the project?
    - * Overall, we enjoyed the project. It is very interesting to have created a program that can actually solve a problem for you in real life, and in this case that is choosing what movie to watch.
  + Did you have any challenges? If so, describe.
    - * We had to choose what language, dataset, and GUI to use. But once we analyzed our options we figured out our best options. It was also very hard to find a dataset that we wanted so I had to use 2 different CSV files (from the same database) to accurately have the program work.
  + If you were to start once again as a group, any changes you would make to the project and/or workflow?
    - * We would probably start earlier as well as spacing out our work throughout multiple weeks. It would also have likely been easier to do in python compared to C++.
  + Comment on what each of the members learned through this process.
    - * Drew: I learned how to properly code BFS and Dijkstras with adjustments to fulfill my needs for the project. I also learned how much of a difference an adjacency list has compared to an adjacency matrix. I also learned how to properly parse multiple different CSV files.
      * Sebastian: I learned how to analyze and compare data given in the form of CSV files and find ways to traverse through the data. I also learned how to analyze functions to find their time complexities.
      * Zach: I learned how to create user interfaces in C++ using the QT library. I also gained a better understanding of the Dijkstra's and BFS algorithms and how they can be applied into useful programs.