Первостепенно следует разбить задачу на некоторые подзадачи, чтобы определить наиболее эффективные реализации для каждой из подзадач и проверить их совместимость:

1) Необходимо пройти через каждую команду в файле команд и найти её рейтинг.
2) Чтобы найти рейтинг команды, в файле игроков найти и просуммировать рейтинги.
3) Отсортировать структуру данных с номерами и рейтингами команд по рейтингу.
4) Использовать алгоритм удаления лишнего для нечётного числа команд.

Приступ к разбору каждой из подзадач:

0) Какую структуру данных использовать? Наиболее оптимальным вариантом в нашем случае будет массив или вектор, ввиду того, что это позволяет нам обратиться к ячейке N, что будет соответствовать команде N. Также в дальнейшем нам нужен будет произвольный доступ для сортировки. В данной реализации выбран вектор, который не требует заранее заданного размера, что даёт большую безопасность при нарушениях работы программы. Однако допустимо использовать массив, при этом посчитав количество <переводов строки+1> для файла команд, создавая заранее массив, что в случае с большими наборами данных ускорит работу с данными, так как не будут необходимы промежуточные выделения памяти и возможные переаллокации данных. Таким образом вектор выбран из соображения безопасности, в то время как массив был бы выбран для производительности.

1) Так как необходимо пройти последовательно через весь файл с командами, то смысла копировать информацию в структуру данных нет, поэтому будем производить чтение из файла, для каждого номера команды создавая новую ячейку(номер команды + её рейтинг) нашей финальной структуры данных, а после записывая туда рейтинг, когда будет достигнут конец строки.

2) Для каждого встреченного номера игрока необходимо найти соответствие в файле игроков, однако игроки разбросаны по номерам в командах в случайном порядке. Было опробовано две реализации этого поиска:

А) Игнорирование (input ignore) N-1 строк файла и чтение второго значения строки.

Б) Запись содержимого файла в вектор и произвольное обращение к N элементу.

В итоге вариант с записью в вектор оказался эффективнее по времени и был оставлен, однако в отличие от чтения напрямую из файла, он требует выделения значительного объёма памяти, в данном случае критерием была скорость выполнения.

3) Учитывая, что количество игроков было достаточным для использования алгоритма сортировки с O(nlog(n)), то было принято решение использовать STL sort либо qsort, в результате за счёт inline и некоторых оптимизаций STL sort выигрывал по производительности и был оставлен.

4) Для чётного числа команд достаточно отсортированного вектора, из которого мы будем брать попарно команды и составлять из них пары. Однако если команд нечётное

количество, то нам следует придумать ухищрения. Так как это основная сложность задания, то мной будет предложено несколько модификаций решения.

А) Следует пройти по всему вектору от первого до предпоследнего элемента, проверяя расстояние справа от команды, в итоге находя максимальную дистанцию. Когда максимальная дистанция будет найдена, программа вернёт нам порядковый номер команды, стоящей слева от дистанции. Если эта команда нечётная(чётная, если от 0), то её мы и удаляем из матчмейкинга, иначе берём нечётную, что находится справа от дистанции. Это позволяет нам охватить и начальное, и конечное значения.

Однако данный метод приведёт к тому, что произойдёт смещение пар. В данном случае то, что мы анализируем расстояние до соседей для каждого элемента, тут же учитывает и невозможность появления новой максимальной дистанции больше прежней. Однако, по факту, мы наверняка выигрываем в расстоянии только для одной пары команд, в то время как остальные команды в сумме могут дать противоположный эффект.

В таком случае эффективно записать прежнюю среднюю дистанцию и сравнить с новой, полученной в результате «смещения». Если дистанция увеличилась, то необходимо последовательно исключать из возможного удаления команды, при удалении которых нарушается баланс до тех пор, пока дистанция не будет уменьшена. Сравнить среднюю дистанцию рейтинга с некоторым значением X, предельным отклонением рейтинга, полученным из статистических данных для конкретной игры, при котором не происходит дестабилизация игровой обстановки. Если отклонение выше этого трешхолда, то, вероятно, игроков недостаточно, чтобы сформировать сбалансированный матчмейкинг, в таком случае имеет смысл применить второй алгоритм.

Б) Использованный при моей реализации в данной задаче алгоритм чрезвычайно прост и не требователен к ресурсам, в отличие от вышеописанного, однако вышеописанный алгоритм включает в себя данный. Проводится анализ крайних элементов и удаляется тот, что больше всего отстаёт от других. С точки зрения игрового матчмейкинга эффективнее удалять игрока с меньшим рейтингом, так как игроков такого уровня большее число. В то же время данный алгоритм плохо справляется с задачей исключения невписывающегося звена, но ускоряет формирование очереди. Для большого числа игроков маловероятно сильное расхождение в рейтинге, поэтому нужды в применении сложного алгоритма нет, по крайней мере в данной задаче, где неучтены многие нюансы реальной очереди игроков и динамического добавления различных участников.

Вероятно, что было бы эффективно проанализировать расхождение команд после сортировки, и если оно не превышает некоторое X в среднем, то выполнить упрощённый алгоритм, отсекающий одну из крайних команд, иначе выполнить сложный алгоритм, чтобы избежать недопустимого нарушения баланса.