ECL-Take-Home

Note: This take-home uses the Bun.sh runtime rather than NodeJS. Some benefits of this include:

- Directly running TypeScript without transpiler step
- Easier end-to-end command line testing by running bash commands in TypeScript
- Compiling an executable allowing users to run programs without installing the Bun runtime
- And much more!

How do I use this?

```
# With Bun.sh (Requires installing the Bun runtime)
bun i # installs the node dependencies
bun run cli.ts <pathname> <N highest scores>

# Running with compiled executable:
./highest ./data/basic.data 2
```

Running tests (requires Bun installed)

See `/tests` folder for comprehensive unit and end-to-end testing. All tests can be run with the command:

```
bun test
```

Building the executable (optional)

A tested executable for MacOS, Linux and Windows have been added. `(./highest and ./highest.exe).` If for some reason the executable does not run on your OS, Bun can build the executable for only the host operating system with the command: For additional documentation see here.

```
bun build ./cli.ts --compile --outfile highest
```

Folder Structure

- `data`: .data files for testing and `gen.js`
- `highest.exe`: Program executable for running on Windows
- `highest`: Program executable for running on MacOS / Linux
- `tests`: Folder containing unit tests
  - `endToEnd.test.ts`: Runs the program as a bash command for end-to-end testing
  - `sort.test.ts`: Unit testing for sort functions
- `assignment-instructions`: Given documentation for assignment
- `cli.ts`: Main program for the command line program
- `utils.ts`: Utility functions and types for implementation and testing

Optimizations and Performance

Rather than the typical approach of sorting an entire list with the built in sorting functions (quicksort / mergesort), this program uses the heap data structure to optimize the sorting time complexity. This reduces the overall time complexity from O(N log N) to O(N log K) where K is the size of the output and where N is the number of records.

- Space Complexity: O(N) where N is the number of records
- Time Complexity: O(N log K) where N is the number of records and K is the size of the output