Final Exam : Mile Times

CSE 130: Algorithm Design

English Summary of Solution

Read the running times from a file into a list of tuples – the first being the name and the second being the mile time in seconds. Next, create another list representing the fastest runners. Loop through the first list a second time and, every time a runner is found that is faster than 360 seconds, add it to the fastest runner list. Finally, display the results on the screen.

Pseudocode

```
done ← FALSE
WHILE not done
  PROMPT for threshold time
  GET threshold_time
  IF threshold_time > 0
     done ← true

PROMPT for filename
GET filename
READ runners FROM filename

fastest ← empty
FOR runner IN runners
   ASSERT 0 ≤ runner.time
   ASSERT 0 ≤ threshold_time
   IF runner.time ≤ threshold_time
      fastest.append(runner)

IF NOT fastest.empty()
   FOR runner IN fastest
      ASSERT runner.time ≤ threshold_time
      PUT fastest
ELSE
   PUT there are no fast runners!
```

Metrics

Efficiency

The following is O(1) because it is independent of the size of the input

```
done ← FALSE
WHILE not done
  PROMPT for threshold time
  GET threshold_time
  IF threshold_time > 0
     done ← true

PROMPT for filename
GET filename
```

The following is O(*n*) where *n* is the number of elements in the file.

```
READ runners FROM filename
```

The following is O(*n*) where *n* is the number of elements in the file/list.

```
fastest ← empty
FOR runner IN runners
   IF runner.time ≤ threshold_time
      fastest.append(runner)
```

The following is O(*m*) where *m* is the number of fast runners. Note that $m \leq n$ so this can be approximated with O(*n*).

```
IF NOT fastest.empty()
   FOR runner IN fastest
      PUT fastest
```

This can only be executed zero or one times. Thus, it is O(1)

```
ELSE
   PUT there are no fast runners!
```

Malleability

This is configurable because the runner's times come from an external source (a file) and the threshold time comes from the user.

Understandability

All the variable names are self-evident. There is nothing clever or unclear about the algorithm.

Quality

Asserts are in place to ensure that the runners times make sense.

Here is a file for the trace:

```
Bob 234
Sam 361
Sue 360
Sly 700
```

The key part of the algorithm:

```
A  fastest ← empty
B  FOR runner IN runners
C     IF runner.time ≤ threshold_time
D        fastest.append(runner)
```

Trace with a threshold set to 360. We expect only Bob and Sue to make the list.

Line	fastest	runner
A	[]	/
B	[]	Bob,234
C	[]	Bob,234
D	[(Bob,234)]	Bob,234
B	[(Bob,234)]	Sam, 361
C	[(Bob,234)]	Sam, 361
B	[(Bob,234)]	Sue, 360
C	[(Bob,234)]	Sue, 360
D	[(Bob,234)(Sue,360)]	Sue, 360
B	[(Bob,234)(Sue,360)]	Sly, 700
C	[(Bob,234)(Sue,360)]	Sly, 700