MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1997

Problem Set 1 Solutions

Exercise 1: This exercise introduces the students to the Scheme system. Student responses to this exercise describe what they did when acting as searchers and hiders in the game. The naive hider always hides at the number 42.

Exercise 2: The code for the naive hider works as follows. The naive hider always hides at the value in the valid range that is closest to 42. If the range is entirely above 42, it hides at the lowest value in the range. If the range is entirely below, it hides at the highest value in the range.

The first two `if` statements capture this behavior. The first `if` checks whether 42 is too small to be in the valid range (low..high), and if it is, it "selects" the lowest value (a value that is always `not-higher` than the `split-point`). The second `if` accomplishes the same goal, but for case that 42 is too large to be in the valid range.

```
(naive-hider 0 12 33)
;Value: #t
;; #t means higher
(naive-hider 50 60 70)
;Value: #f
;; #f means not-higher
```

Exercise 3: **3A:** The `implacable-searcher` does very badly against the `elbow-room hider`. The number of questions is always $(high - low)$. The `implacable-searcher` always chooses the lowest number and the `elbow-room-hider` always responds `higher`. The number that the `elbow-room-hider` always picks is `high`.

```
(define searcher implacable-searcher)
(define hider elbow-room-hider)
(play 1)
;Value: 1
(play 100)
;Value: 100
(play 1004)
;Value: 1004
```

3B: To obtain a tail-recursive procedure, we introduce a variable `num-ques` which is kept equal to the number of questions asked so far. By coding the tail recursive procedure as a helper procedure, the change from recursive to tail recursive can be made invisible to the user.

```
;; This code works.  See examples below.
(define (number-of-questions-tr n)
  (define (tr-helper n num-ques)
     (if (= n 0)
         num-ques
         (tr-helper (floor (/ n 2)) (+ num-ques 1))))
  (tr-helper n 0))
(define searcher binary-searcher)
(define hider elbow-room-hider)
(play 3)
;Value: 2
(number-of-questions-tr 3)
;Value: 2
(play 4)
;Value: 3
(number-of-questions-tr 4)
;Value: 3
(play 8)
;Value: 4
(number-of-questions-tr 8)
;Value: 4
(play 10000)
;Value: 14
(number-of-questions-tr 10000)
;Value: 14
```

Exercise 4:

The number of questions the `random-searcher` uses increases twice as quickly as the number of questions the `binary-searcher` uses. The binary searcher uses very close to $\log_2 n$ questions where n is the size of the range. The `random-searcher` also uses a logarithmic number of questions with respect to the size of the range. The `random-searcher` ends up guessing near the middle only 50% of the time; which accounts for the inefficiency.

Exercise 5: The idea is to use the processes captured by `random-searcher` and `binary-searcher` to build a searcher that behaves like one on an odd interval and like the other on an even interval. The number of numbers in the range [low..high] is $(high - low) + 1$. If the size of the range is even, `binary-searcher` is called. If the size of the range is odd, `random-searcher` is called.

```
;; This code works as shown by the examples below.
(define (random-binary-searcher high low)
  (if (even? (inc (- high low)))
      (binary-searcher low high)
      (random-searcher low high)))
```

```
(define searcher random-binary-searcher)
(searcher 0 100)
;Value: 30
(searcher 0 100)
;Value: 67
(searcher 0 99)
;Value: 49.
(searcher 0 99)
;Value: 49.
```

Exercise 6:

6A.

```
(define searcher implacable-searcher)
(define hider random-answer-hider)
(tournament 10 100)
    ;;  game no: 1  game-length: 2
    ;;  game no: 2  game-length: 4
    ;;  game no: 3  game-length: 2
    ;;  game no: 4  game-length: 4
    ;;  game no: 5  game-length: 1
    ;;  game no: 6  game-length: 1
    ;;  game no: 7  game-length: 1
    ;;  game no: 8  game-length: 2
    ;;  game no: 9  game-length: 2
    ;;  game no: 10  game-length: 3
The MEAN game length was 2.2, with DEVIATION 1.0770329614269005
;Value: "Tournament is over."
(tournament 10 1000)
    ;;  game no: 1  game-length: 1
    ;;  game no: 2  game-length: 6
    ;;  game no: 3  game-length: 1
    ;;  game no: 4  game-length: 1
    ;;  game no: 5  game-length: 1
    ;;  game no: 6  game-length: 3
    ;;  game no: 7  game-length: 3
    ;;  game no: 8  game-length: 4
    ;;  game no: 9  game-length: 1
    ;;  game no: 10  game-length: 1
The MEAN game length was 2.2, with DEVIATION 1.6613247725836147
;Value: "Tournament is over."
```

```
(tournament 10 100000000)
    ;;  game no: 1  game-length: 1
    ;;  game no: 2  game-length: 1
    ;;  game no: 3  game-length: 1
    ;;  game no: 4  game-length: 3
    ;;  game no: 5  game-length: 1
    ;;  game no: 6  game-length: 4
    ;;  game no: 7  game-length: 1
    ;;  game no: 8  game-length: 5
    ;;  game no: 9  game-length: 1
    ;;  game no: 10  game-length: 6
The MEAN game length was 2.4, with DEVIATION 1.8547236990991407
;Value: "Tournament is over."
```

A game against the `implacable-searcher` immediately ends when a hider answers `not-higher`. The `random-answer-hider` returns `not-higher` half the time. So half the time, the game ends in one question, 1/4 of the time in two questions, 1/8 in three, ... This implies the average game between the `random-answer-hider` and the `implacable-searcher` is at most two questions, regardless of the size of the initial interval.

6B.

```
(define (random-location-hider low split-point high)
  (if (> (guess low (- high 1)) split-point)
      higher
      not-higher))
;; This code works as shown by the following examples.
```

```
(define hider random-location-hider)
(hider 0 3 10)
;Value: #t
(hider 0 3 10)
;Value: #t
(hider 0 3 10)
;Value: #t
(hider 0 3 10)
;Value: #t
(hider 0 3 10)
;Value: #f
(hider 0 3 10)
;Value: #t
(hider 0 3 10)
;Value: #f
(hider 0 3 10)
;Value: #f
(hider 0 3 10)
;Value: #t
(hider 0 3 10)
;Value: #t
;; Summary: 7 higher, 3 not-higher
```

It turns out that the average game between `random-location-hider` and `implacable-searcher` uses a number of questions equal to about half the size of the initial interval.

Exercise 7:

7A.

The only real change that needs to be made is in finish-round, but a number of other modifications make the code clearer. First, the helper functions have been renamed to avoid clobbering the definitions that regular `play` uses. Another way of doing this is to define the helper procedures in the `play-reporting-high` definition as is done in exercise 7B. The argument `num-questions` has been renamed to `num-highers` for clarity. Lastly and most importantly, `finish-round-high` only increments `num-highers` when the hider returns `higher`.

```
;; All of this code works.  Claim supported by examples below.
(define (play-reporting-high n)
  (play-loop-high 0 n 0))
(define (play-loop-high low high num-highers)
  (if (= low high)
      num-highers
      (ask-hider-high (searcher low high) low high num-highers)))
(define (ask-hider-high split-point low high num-highers)
  (finish-round-high (hider low split-point high)
     split-point
     low
     high
     num-highers))
(define (finish-round-high response split-point low high num-highers)
  (if (higher? response)
      (play-loop-high (inc split-point) high (inc num-highers))
      (play-loop-high low split-point num-highers)))
```

Here are a pair of test cases:

```
(define searcher implacable-searcher)
(define hider elbow-room-hider)
(play 100)
;Value: 100
(play-reporting-high 100)
;Value: 100
(define searcher binary-searcher)
(play 100)
;Value: 7
(play-reporting-high 100)
;Value: 3
```

7B.

```
;; This code works.
(define (tournament tournament-length n)
  ;; Define tournament-loop inside of tournament so it "sees"
  ;; tournament-length and n.
  (define (tournament-loop game-num sum sum-of-sq)
    (if (= game-num tournament-length)    ;is play done?
        (begin                            ;play is done
          (newline) (newline)
          (display "The MEAN game length was ")
          (display (/ sum game-num))
          (display ",")
          (display " with DEVIATION ")
          (display (deviation game-num sum sum-of-sq))
          (newline)
          "Tournament is over.")
        (display-outcome                  ;play is not done
         (play n)
         (inc game-num) sum sum-of-sq)))
  ;; Same for display-outcome
  (define (display-outcome
           game-length gnum-plus1 sum sum-of-sq)
    (begin
      (newline)
      (display "    ;;  game no: ") (display gnum-plus1)
      (display "  game-length: ") (display game-length)
      (tournament-loop gnum-plus1
                       (+ sum game-length)
                       (+ sum-of-sq (square game-length)))))
  (tournament-loop 0 0 0))
(define searcher implacable-searcher)
(define hider random-location-hider)
(tournament 10 100)
    ;;  game no: 1  game-length: 12
    ;;  game no: 2  game-length: 66
    ;;  game no: 3  game-length: 18
    ;;  game no: 4  game-length: 96
    ;;  game no: 5  game-length: 93
    ;;  game no: 6  game-length: 43
    ;;  game no: 7  game-length: 70
    ;;  game no: 8  game-length: 13
    ;;  game no: 9  game-length: 44
    ;;  game no: 10  game-length: 20
The MEAN game length was 47.5, with DEVIATION 30.627601930285046
;Value: "Tournament is over."
```