Algorithms and Complexity

Exercise: “Lazy Lemmings”

Danilo Catalan Canales

Design:

Since we must find the optimal number of trials in the worst case we need to start with dividing the worst case scenarios in to sub problems:

(N = k, l = 1)
In the case that we only have 1 lemming we need to see through all k boards from the first until we reach the k’ board that's unsafe.

(N = k, l > 1)
When choosing to jump from board “x < k” we have two cases to address:

- The lemming gets hurt, and we can discard boards higher than x and start checking lower boards.
- The lemming is unhurt and we can safely discard the boards lower than x and start checking the higher boards.

So we’re looking at a formula where we want the **min** attempts in 1 trial + selecting the max(amount_of_boards_lower_than_x , amount_of_boards_higher_than_x) for all x of N

Naive Recursive approach:

jump(N,l):

```
1. if N is 0 or 1: return N
2. if l is 1: return the amount of N,
3. var minTrials = inf
4. for k in range(1, N){
    a. result = max(jump(N-k,l), jump(k-1,l-1))
    b. if result < minTrials: minTrials = result
   }
5. return 1+minTrials
```

Here we
1: since there is only 1 case when N =1 and 0 when N = 0 This is also true for point 2, when the number of lemmings is 1
3: We need a reference to some big number, so that we can change it after computing the max(....) value
4. Here we start choosing the k value, noting that we start from 1 to N since 0 is redundant.
a: In this line we check which one of the two cases is the worst off. So the left side is when the board is unsafe and the right side is the one which is safe. We do a recursion until we reach our base case which is when the input of the jump recursion is N = 0,1 or l =1
b: the if statement checks that we're in fact choosing the min number of trials, and if there is a smaller trial count between the worst case

Proof of Correctness:

1.
At the initialization phase N is the amount of boards we want to check and l is the amount of lemmings at "our disposal". N and l are chosen by the user. For simplicity we choose N=3, l = 2 as our input values.

We declare that a minTrial variable is as high as possible to later discard it when compared to the first value we get from the max function comparison. This is done for every recursion.

2.

```
jump(3,2)
for k in range(3) (k=1)
        result = max(jump(2,2), jump(0, 1))
        …
```

second expression will give 0, because of the first condition(if N = 0: then return N)
first expression will enter a recursive call and enter a new iteration and we will get:

```
jump(2,2)
for k in range(2) (k=1)
        result = max(jump(1,2), jump(1, 1)) result = 1
        minTrials= infiniti > result =1:
                minTrials = result
…
```

both expressions will grant 1 and so the result will equal 1

```
jump(2,2)
for k in range(2) (k=2)
        result = max(jump(0,2), jump(1, 1)) result = 1
        minTrials= 1 > result =1:
                no change
return 1+minTrials
```

we return to the first iteration where jump(2,2) returned 2

```
jump(3,2)
for k in range(3) (k=1)
    result = max(jump(2,2), jump(0, 1)) (result = 2)
        minTrials= infiniti > result = 2:
            minTrials = result
    …


jump(3,2)
for k in range(3) (k=2)
    result = max(jump(1,2), jump(1, 1)) (result = 1)
        minTrials= 2> result = 1:
            minTrials = result
    …


jump(3,2)
for k in range(3) (k=3)
    result = max(jump(0,2), jump(2, 1)) (result = 2)
        minTrials= 1> result = 2:
            no change
```

when we finally have reached final iteration we return the value of our minimum trials for the worst cases + 1

```
return 1+ minTrials = 2
```

This concludes the proof of correctness.

Time complexity:

Just as the naive recursive fibonacci solution, as we keep re-calling the function we obtain an exponential time complexity. The time is $N(t(N-1) + t(N-1)+O(a)) <= N(2 * T(N))$, a being constant.
This is the same as $O(N * 2^N)$.

2nd Design:

We can follow the same structure as the previous schematic and saving whenever we have a result of minimum trials for k boards.

Recursive with memory approach:

<u>jump(N,I):</u>

```
1. if memory[I][N] != infinity:
    a. return memory[I][N]
2. if N is 0 or 1: return N
3. if I is 1: return the amount of N
4. var minTrials = inf
5. for k in range(1, N){
    a. result = max(jump(N-k,I), jump(k-1,I-1))
    b. if result < minTrials:
        i. minTrials = result
   }
6. memo[I][N] = minTrials
7. return minTrials
```

Just as the previous algorithm but with the exception that if the memory isn't some huge number, return it as a representative for that recursion with those N and I inputs. As might have been recognized, it is very similar to the fibonacci memory algorithm, but with different constraints to fulfill.

Proof of Correctness:

1.
Initializing we have N and I inputs which are chosen by the user. Our memory slot is filled with the number **infinity** so that when compared in the iteration we discard it and replace it. Every I and N has a reserved slot in the memory with the specific memory[I][N] keys.
We also define a minTrials variable with a high as possible number. Which will be compared in the iteration.

2.
Just as with the previous algorithm we go through the different conditions and also check in the memory slots with the specific N and I inputs. If there is a value that's not **infinity**, skip the rest of the code and return the already saved data from the memory for N and I.
If not, go through the iteration and check the max for the x_lower and x_above cases.
When max has been chosen we give that value to the results variable. We check if it's optimal by comparing it with the current minTrails. If "results" is less than minTrails, then replace the current minTrails with the results and if not keep minTrails as is and do the next iteration.

3.
When the iteration has finished. We save the optimal trials in the memory with the specific N and I keys for the memory slot and return the 1+minTrails. This would be the minimum number of trials for worst case for N and I.

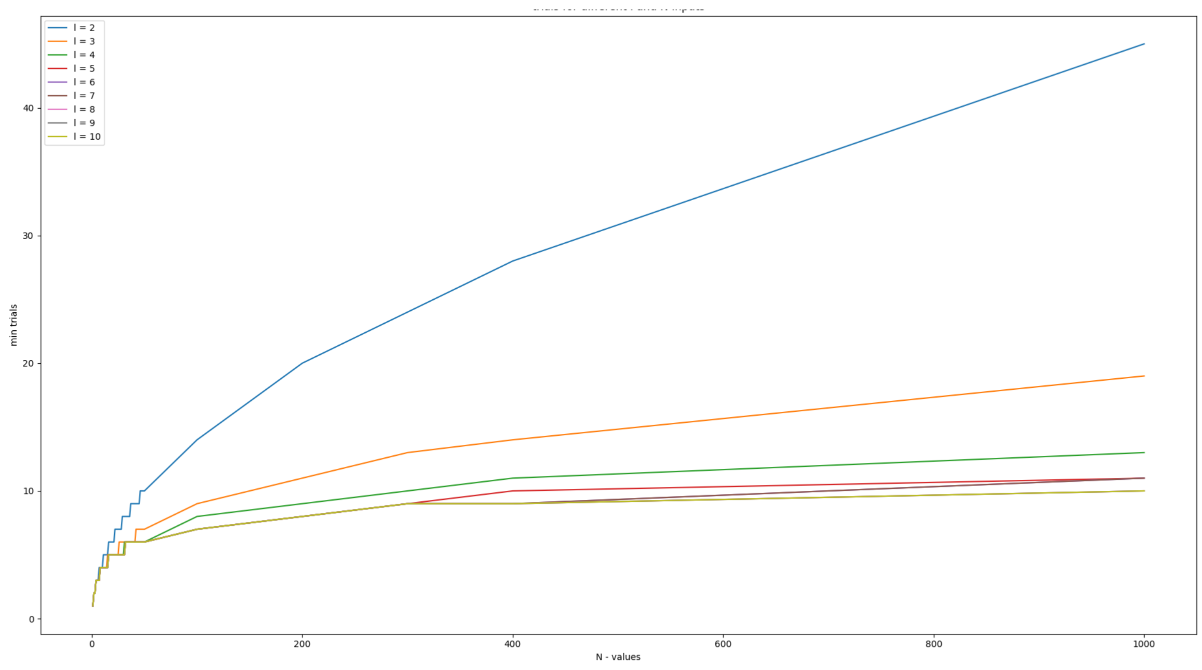
This concludes the proof of correctness.

Time complexity

The time complexity given for the function is:

N - the for loop will at most run for N times.
and will run N^2 recursions as maximum

Which is O (N * N ^2) => O(N^3)



skipped I = 1 because it would only give back the same N values.

I = 2,

[1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 14, 20, 24, 28, 32, 45]

I=3

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 9, 11, 13, 14, 15, 19]

I = 4

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 8, 9, 10, 11, 11, 13]

I = 5

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 9, 10, 10, 11]

I = 6

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 9, 9, 10, 11]

I = 7

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 9, 9, 9, 11]

I = 8

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 9, 9, 9, 10]

I = 9

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 9, 9, 9, 10]

I = 10

[1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 9, 9, 9, 10]

Exemplify:

1 : inf | 2 : 2 | 3 : 2 | 4 : 3 | 5 : 3 | 6 : 3 | 7 : 4 | 8 : 4 | 9 : 4 | 10 : 4 | 11 : 5 | 12 : 5 | 13 : 5 |

14: 5 | 15 : 5 | 16 : 6 | 17 : 6 | 18 : 6 | 19 : 6 | 20 : 6 | 21 : 6 | 22 : 7 | 23 : 7 | 24 : 7 | 25 : 7 | 26 : 7|

27 : 7 | 28 : 7 | 29 : 8 | 30 : 8 | 31 : 8 | 32 : 8 | 33 : 8 | 34 : 8 | 35 : 8 | 36 : 8 | 37 : 9 | 38 : 9 |

39 : 9 40 : 9 | 41 : 9 | 42 : 9 | 43 : 9 | 44 : 9 | 45 : 9 | 46 : 10 | 47 : 10 | 48 : 10 | 49 : 10 | 50 : 10 |

51 : 10 | 52 : 10 | 53 : 10 | 54 : 10 | 55 : 10 | 56 : 11 | 57 : 11 | 58 : 11 | 59 : 11 | 60 : 11 | 61 : 11 |

62 : 11 | 63 : 11 | 64 : 11 | 65 : 11 | 66 : 11 | 67 : 12 | 68 : 12 | 69 : 12 | 70 : 12 | 71 : 12 | 72 : 12 |

73 : 12 | 74 : 12 | 75 : 12 | 76 : 12 | 77 : 12 | 78 : 12 | 79 : 13 | 80 : 13 | 81 : 13 | 82 : 13 |

83 : 13 | 84 : 13 | 85 : 13 | 86 : 13 | 87 : 13 | 88 : 13 | 89 : 13 | 90 : 13 | 91 : 13 | 92 : 14 |

93 : 14 | 94 : 14 | 95 : 14 | 96 : 14 | 97 : 14 | 98 : 14 | 99 : 14 | 100 : 14 | 101 : inf | 102 : inf |

By following this table we can have an example like:

6 -> 15 -> 20

10-> 46 or 10 ->55

3-> 10 -> 28

These are some of the combinations on getting a min of 14 trials