Uzair Akram

MA3

Problem BreakDown:
Given: k Lines, 2 Darts
Get: the highest line where player can hit the bulls-eye
Optimize the number of trials; find: a minimum trials for each k

Programming Terms:
Input: k Lines, 2 Darts.
Output: farthest distance, you can hit bullseye
Optimize: number of trials

Brute Force

```
bullseye(k, d):
    miss = false
    i=0
    while i < k:
        miss = throw(i)
        if miss:
            return i
        i++
    return i

```

The brute force algorithm iteratively finds the threshold at which the player misses the bullseye.

The above algorithm iteratively finds the

```python
import math

def throw(n):
    if n > 846357:
        return 1
    else:
        return 0

# s = start line, k = final line, d = number of darts
def bullseye(s, k, d):
    miss = 0
    i = s

    if d < 2:
        while i < k:
            miss = throw(i)
            if miss == 1:
                print(i-1)
                break
            i = i+1
    else:
        x = s + int(math.sqrt(k))
        miss = throw(x)


        if miss == 1:
            bullseye(s, x-1, d-1)
        else:
            bullseye(x, k, d)


k = 1000000
d = 2

print('The maximum line number from which the player hits bullseye:')
bullseye(0, k, d)
```

The output:

```
(base) Uzairs-MacBook-Pro:MA3 uzairakram$ python bullseye.py
The maximum line number from which the player hits bullseye:
846357
(base) Uzairs-MacBook-Pro:MA3 uzairakram$
```

This algorithm above can solve the problem more efficiently using recursive programming technique. The algorithm shoots the dart from a test line *x* (determined number of lines, this

segments the problem into subproblems). The algorithm runs recursively updating the either the line *s* or line *k* as it tests lines, we assume that:

1. If the player is able to hit the bullseye at Line x, then he/she can also hit the bullseye at any line numbered less than x.
2. If the player misses the bullseye at Line x, then he/she will also miss the bullseye at line numbered greater than x.

We make use of the condition: *If you miss, you cannot reuse your dart. However, if you hit the bullseye you can reuse the dart.*

The algorithm considers 2 cases where the dart hits the bullseye from the test line *x* and when the dart misses the bullseye. If the dart is hit we continue to recursively use dynamic programming to find the failure line from where the dart misses. If the dart misses the algorithm runs iteratively to find the threshold line starting at the base line and iterating to the failure line.

The algorithm uses a similar technique used for solving the EggDropping problem.