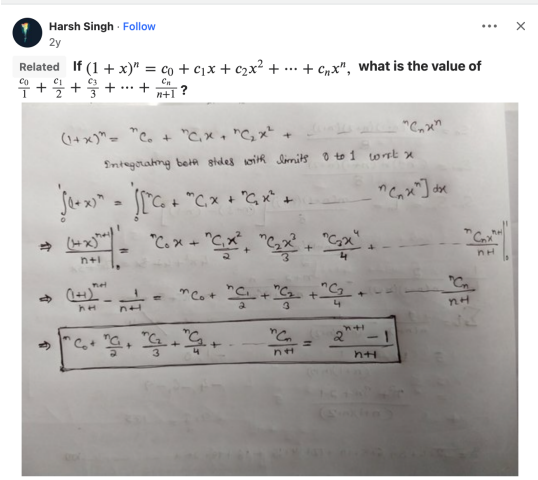
Coin Flip Research

GAME PLAN

- [ ] Do degree-analysis on Pascal's Triangle.
- [ ] Create a bound for P, A, C, J each representing paths: passed, all that arrives there, first time arriving, not arrived yet.
- [ ] Why do different degrees add up to exponential?
- [ ] IMPORTANT!! Each number in row x in the Pascal's Triangle is between 1 to $\frac{2^x}{x}$. How does this relate to ours?
- [ ] How to prove that u/v is an exponential? First, by percentage (heuristics). Second, by Pascal's Triangle bounds.
- [ ] Model the diagonals using polynomials, and use algorithm to model the vertical by some adding algorithm. Do this for Pascal, then do this for Seth's triangle.
- [ ] Optimal triangle setups and such.
- [ ] Read up on rigorous processes of modeling.
- [ ] Make generalization of Pascal's Triangle $\frac{1}{n!}\prod_{i=0}^{n}(x-i)$, dependent on diagonal n, layer x.
- [ ] Catalan numbers and parenthesis expressions' relations to this?
- [ ] Lucas' theorem: https://www.quora.com/How-should-I-prove-that-if-n-2-k-1-k-in-mathbb-N-then-every-entry-in-row-n-of-Pascals-triangle-is-odd-for-example-if-k-2-then-n-3-and-row-3-is-1-3-3-1-Both-1-and-3-are-odd-so-the-statement-is-true-for-k-2.
- [ ] Figure out how to get polynomials for Pascal's Triangle's diagonals, and why they sum up horizontally to be 2^n.
- [ ] Relate that to the Seth Triangle.
- [ ] Understand this.



Publication

Waterloo Journal of Integer Sequences: https://cs.uwaterloo.ca/journals/JIS/.

Colgate INTEGERS: http://math.colgate.edu/~integers/. Too difficult.

American Journal of Undergraduate Research: http://www.ajuronline.org/submissions/.

- There could be some water-ish stuff in the one above, especially in mathematics.

The Mathematical Intelligencer.

- $P(x,y) = P(x-1,y+1) + P(x-1,y-1)$.

- $P(x, 0) = 2^x$.
- $P(x, x) = 1$.
- $P(x, x-1) = 2$.

- [x] ~~Find a local file storing way to create storage for R function results.~~
- [x] ~~Figure out time complexity.~~
- [x] ~~Find a way to conquer the 60 barrier.~~
  - [x] ~~Find relationship between $P(x, y)$ and $P(x+1, y)$.~~

▼ Problem Description

- Standing at 0 on the number line. When a coin is flipped, heads mean right and tails mean left. Make a function $P(x, y)$ returning the number of result combinations of x coin flips that arrives at position y at any instance. Note that evidently, the ordering within these x coin flip matters.

▼ Intuition 1 —

- Split the condition-satisfying permutations into groups. For instance, y (minimum number) heads total, $y+1$ heads total, etc. They do not overlap.
- Recursion. Have another function $R(x, y, z)$, with z denoting the number of heads achieved.
- If z are all in the first $z + (z - y) = 2z - y$ coin slots, the permutation counts.

▼ Formula Design

- Have $P(x, y)$ as the big function, and $R(x, y, z)$ as the recursive function.
- Definitions.
  - $P(x, y) = \sum_{i=0}^{x-y} R[x, y, y+i]$.
  - $R(x, y, z) = \sum_{i=0}^{\max(0, \min(z-y, x-(2z-y)))} R[(\min(x, 2z-y), y, z-i) \times \binom{\max(1, x-\min(x, 2z-y))}{i}]$.
  - For $R(x, y, z)$, if $y = z$, then $R = 1$.
  - For $R(x, y, z)$, if $2z - y \geq x$, then $R = \binom{x}{z}$.

▼ Implementation

▼ Brute force code.

```
// Brute force calculate the permutations of P(x, y).
// Time complexity: O(2^n). Exponential.
class BruteP {

    // Brute force calculates the number of permutations of x coins reaching y distance.
    public static int bruteP(int x, int y) {
        /*
         Brute force, O(2^x) time.

         Function:
         - Simulates all the possible permutations of x coins through an x-sized boolean array.
         - Uses reached() to examine each array; count the total number of arrays that reach the target.

         Note: x = 20 is pushing the limits.

        */
        int howManyReach = 0;
        BigInteger bi = BigInteger.ZERO;
        BigDecimal rows = BigDecimal.valueOf(Math.pow(2, x));
        while (bi.compareTo(rows.toBigInteger()) < 0) {
            StringBuilder bin = new StringBuilder(bi.toString(2));//Integer.toBinaryString(i);
            while (bin.length() < x)
                bin.insert(0, "0");
            char[] chars = bin.toString().toCharArray();
            boolean[] boolArray = new boolean[x];
            for (int j = 0; j < chars.length; j++) {
                boolArray[j] = chars[j] == '0';
            }
            howManyReach += reached(boolArray, y);
            bi = bi.add(BigInteger.ONE);
```

```
        }
        return howManyReach;
    }

    // Input an array and see if it reaches the target.
    public static int reached(boolean[] list, int y) {

        int k = 0;
        for (boolean b : list) {
            if (b) {
                k++;
            } else {
                k--;
            }
            if (k == y) {
                return 1;
            }
        }
        return 0;
    }

}
```

▼ P1 code.

```
// Recursively defined P, splitting it according to the number of heads.
// Time complexity: honestly I don't know. Need to figure this out.
class P1 {

    // Smartly calculates the number of permutations of x coins reaching y distance.
    public static long P(int x, int y) {
        long totalP = 0;
        for (int i=0; i<=x-y; i++) {
            totalP += R(x, y, y+i);
        }
        return totalP;
    }

    // Recursively calculates the number of permutations of x coins reaching y distance, with z coins facing up.
    public static long R(int x, int y, int z) {
        if (y == z) {return 1;}
        if (2*z - y >= x) {return choose(x, z);}

        long totalR = 0;

        int k1;
        int k2;
        int k3;
        int k4 = 2*z-y;
        k1 = Math.max(0, Math.min(z-y, x-k4));
        k2 = Math.min(x, k4);
        k3 = Math.max(1, x-k2);

        for (int i=0; i<=k1; i++) {
            totalR += R(k2, y, z - i) * choose(k3, i);
        }
        return totalR;
    }

    // The "choose()" function is essential to combinatorics.
    static final Map<Long, Map<Long, Long>> map = new HashMap<>();
    public static long choose(long total, long choose) {
        if(total < choose)
            return 0;
        if(choose == 0 || choose == total)
            return 1;

        if (!(map.containsKey(total) && map.get(total).containsKey(choose))){
            map.put(total, new HashMap<>());
            map.get(total).put(choose, choose(total-1,choose-1)+choose(total-1,choose));
        }
        return map.get(total).get(choose);
    }

}
```

▼ P2 code.

```
// Much improved dynamic programming approach. Still using a recursively defined sequence C, but much faster.
// Time complexity: O(n^2 * O(choose)). Since we use HashMap, O(choose) is likely O(1), but I'm unsure. Need to ponder.
class P2 {

    // Smartly calculates the number of permutations of x coins reaching y distance.
    public static BigInteger P(int x, int y) {
        BigInteger[] C = new BigInteger[((x-y)/2)+1];
        BigInteger p = new BigInteger("0");
        BigInteger two = new BigInteger("2");

        for (int i=0; i<C.length; i++) {
            BigInteger overlap = new BigInteger("0");
            for (int j=0; j<i; j++) {
                overlap = overlap.add(C[i-j-1].multiply(choose((long) 2*(j+1), (long) j+1)));
            }
            C[i] = choose(y+((long) 2*i), (y+(long) i)).subtract(overlap);
            p = p.add(C[i].multiply(two.pow(x-y-(2*i))));
        }
        return p;
    }

    // The "choose()" function is essential to combinatorics.
    static final Map<Long, Map<Long, BigInteger>> map = new HashMap<>();
    public static BigInteger choose(long total, long choose) {
        if(total < choose)
            return new BigInteger("0");
        if(choose == 0 || choose == total)
            return new BigInteger("1");

        if (!(map.containsKey(total) && map.get(total).containsKey(choose))){
            map.put(total, new HashMap<>());
            map.get(total).put(choose, choose(total-1,choose-1).add(choose(total-1,choose)));
        }
        return map.get(total).get(choose);
    }

}
```