Extra Credit Poker Probability Assignment
Aurelia B. Miller - Summer 2026

1. Derive formulas for the probabilities of all the hands at that website if there are s suites and r ranks, and a player gets a 5-card hand.

Royal Flush: nCr(s, 1)/nCr(rs, 5)

Since there is only one possible set of ranks for a royal flush (the top 5 ranks), there isn't a decision to make about the ranks. Instead, the only different possibilities are what suit the royal flush is in, though this hand is still impossible if r < 5 (which is true for all straights/straight flushes). Having accounted for all possible hands that are a royal flush, we divide that by the total number of hands possible: any 5 cards chosen out of the total amount, rs.

Straight Flush: ((nCr(r - 3, 1) - 1) * nCr(s, 1))/nCr(rs, 5)

Here, a rank where the straight will start *does* need to be chosen, and since we have 5-card hands, the highest straight can only start at r - 4 (r - 3, r - 3, r - 2, r - 1, r). However, an ace can also be the start of an A-5 straight, so we choose any one rank out of r - 3 to be the start of the straight (then subtract one starting rank since we exclude royal flush since it's a better hand) and since the other cards have to be the following ranks, move on to the suit. Since this is a straight *flush*, we choose just 1 suit out of s to account for all possible straight flush hands, then, again, divide by the total possible hands.

Four of a Kind: (nCr(r, 1) * nCr(s, 4) * nCr(r - 1, 1) * nCr(s, 1))/nCr(rs, 5)

Four of a kind just chooses one rank and then chooses four different suits to have that rank in. It's impossible to get four of a kind with fewer than four suits, which this formula does show. Then the kicker: we need to have a different rank (if there's, for example, 5 suits, 5 of a kind would be better than 4 of a kind, and for fewer than 5 suits it has to be another rank to be a possible hand) and just choose any one suit, then divide all possible four-of-a-kind hands by all possible 5-card hands.

Full House: (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 2))/nCr(rs, 5)

For a full house, we choose one rank that will have 3 suits chosen to go with it for the 3 same-rank cards, then do the same with 2 suits for the other part, while also subtracting 1 from the possible ranks (because getting the same rank as the first group of 3 would make a better hand or be impossible). Again, divide by all possible hands.

Flush: (nCr(r, 5) * nCr(s, 1))/nCr(rs, 5) - (nCr(r - 3, 1) * nCr(s, 1))/nCr(rs, 5)

For a flush, we choose 5 different ranks (because you can't have the same rank twice within the same suit) and one suit then divide by all possible cards–but then subtract all possible straight flushes, since that's a different (better) hand.

Straight: (nCr(r - 3, 1) * (nCr(s, 1)^5))/nCr(rs, 5) - (nCr(r - 3, 1) * nCr(s, 1))/nCr(rs, 5)

For a straight, we again choose from r - 3 starting values (from "0" to r - 4) and choose one suit out of s 5 times (so to the 5th power) and again divide by all possible cards, and then,

again, subtract all possible straight flushes since that's a better hand (which we must do because we accounted for all possible suit combinations, including straight flushes, and we include royal flushes in the subtraction because by accounting for all ranks and suits those were technically included).

Three of a Kind: (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 2) * nCr(s, 1)^2)/nCr(rs, 5)

For three of a kind we choose one rank and three suits, then for the kickers we choose two ranks out of the remaining options as well as 1 suit out of s suits twice (for each kicker), and finally divide by all possible hands.

Two Pair: (nCr(r, 2) * nCr(s, 2) * nCr(s, 2) * nCr(r - 2, 1) * nCr(s, 1))/nCr(rs, 5)

For two pair, we choose two ranks at the start, then choose two suits twice (for each pair), then pick some other rank and 1 suit for the kicker. Then, of course, divide by all possible hands.

Pair: (nCr(r, 1) * nCr(s, 2) * nCr(r - 1, 3) * (nCr(s, 1)^3))/nCr(rs, 5)

For pair, we pick just one rank and two suits for the pair and then 3 different ranks for the kickers (because if any were the same it would be a better hand) and choose one suit 3 times (to the 3rd power). Then of course divide by all possible hands.

High Card: 1 - (nCr(r, 1) * nCr(s, 2) * nCr(r - 1, 3) * (nCr(s, 1)^3))/nCr(rs, 5) - (nCr(r, 2) * nCr(s, 2) * nCr(s, 2) * nCr(r - 2, 1) * (s, 1))/nCr(rs, 5) - (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 1) * nCr(r - 2, 1) * nCr(s, 1))/nCr(rs, 5) - (nCr(r - 3, 1) * (nCr(s, 1)^5))/nCr(rs, 5) - ((nCr(r, 5) * nCr(s, 1))/nCr(rs, 5) - (nCr(r - 3, 1) * nCr(s, 1))/nCr(rs, 5)) - (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 2))/nCr(rs, 5) - (nCr(r, 1) * nCr(s, 4) * nCr(r - 1, 1) * nCr(s, 1))/nCr(rs, 5)

Since literally any rank or suit *can* be a high card, there's an 100% chance of getting something that could be a high card if no other hands work, but since literally every other hand is better, we subtract all the other probabilities from 1.

2. Consider straight flush, four-of-a-kind, and full house. There are 6 ways to order these types of hands. For which of these 6 ways is there an r, s such that if you have r ranks and s suites, then you get that ordering of probabilities?

6 ways: ***Straight Flush < Four of a Kind < Full House***, ***Straight Flush < Full House < Four of a Kind***, ***Four of a Kind < Straight Flush < Full House***, ***Four of a Kind < Full House < Straight Flush***, ***Full House < Straight Flush < Four of a Kind***, ***Full House < Four of a Kind < Straight Flush***

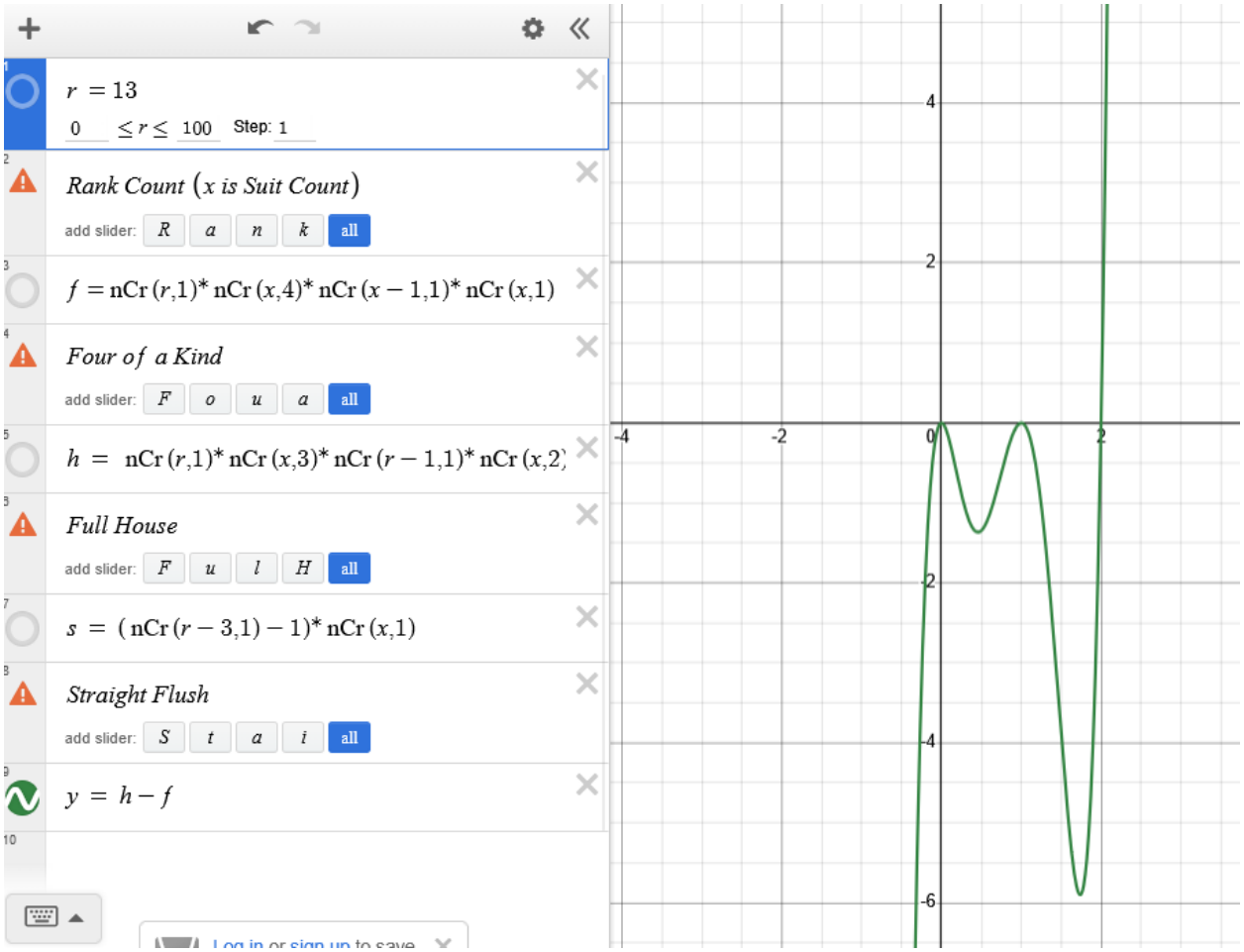
The denominator is the same for all of these, at all possible hands/nCr(rs, 5). Because of this, I can just focus on the numerator/number of ways to actually get each hand in the following calculations. Following this, the relevant formulas are:

Straight Flush: (nCr(r - 3, 1) - 1) * nCr(s, 1)

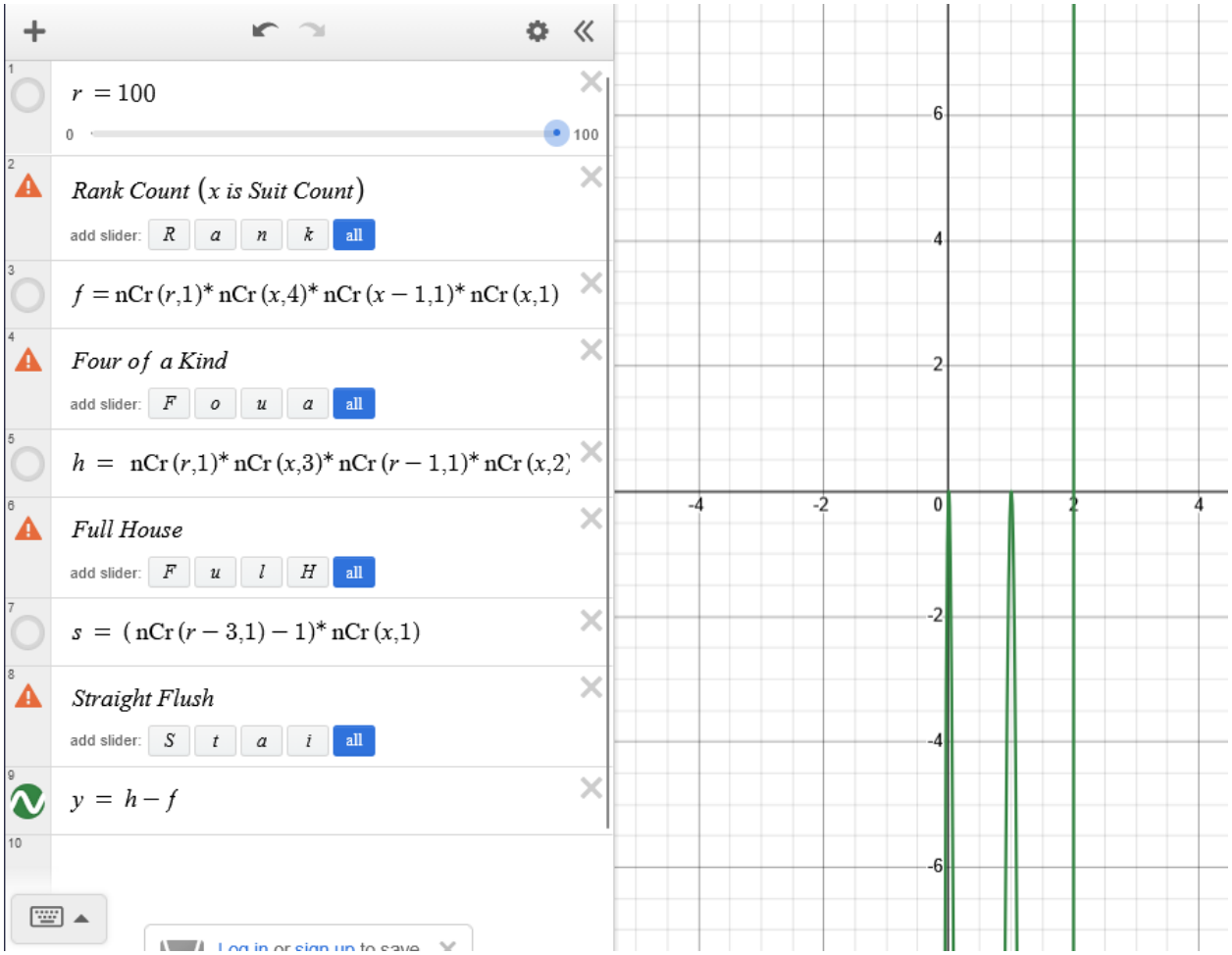
Four of a Kind: nCr(r, 1) * nCr(s, 4) * nCr(r - 1, 1) * nCr(s, 1)

Full House: nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 2)

Four of a Kind vs Full House:

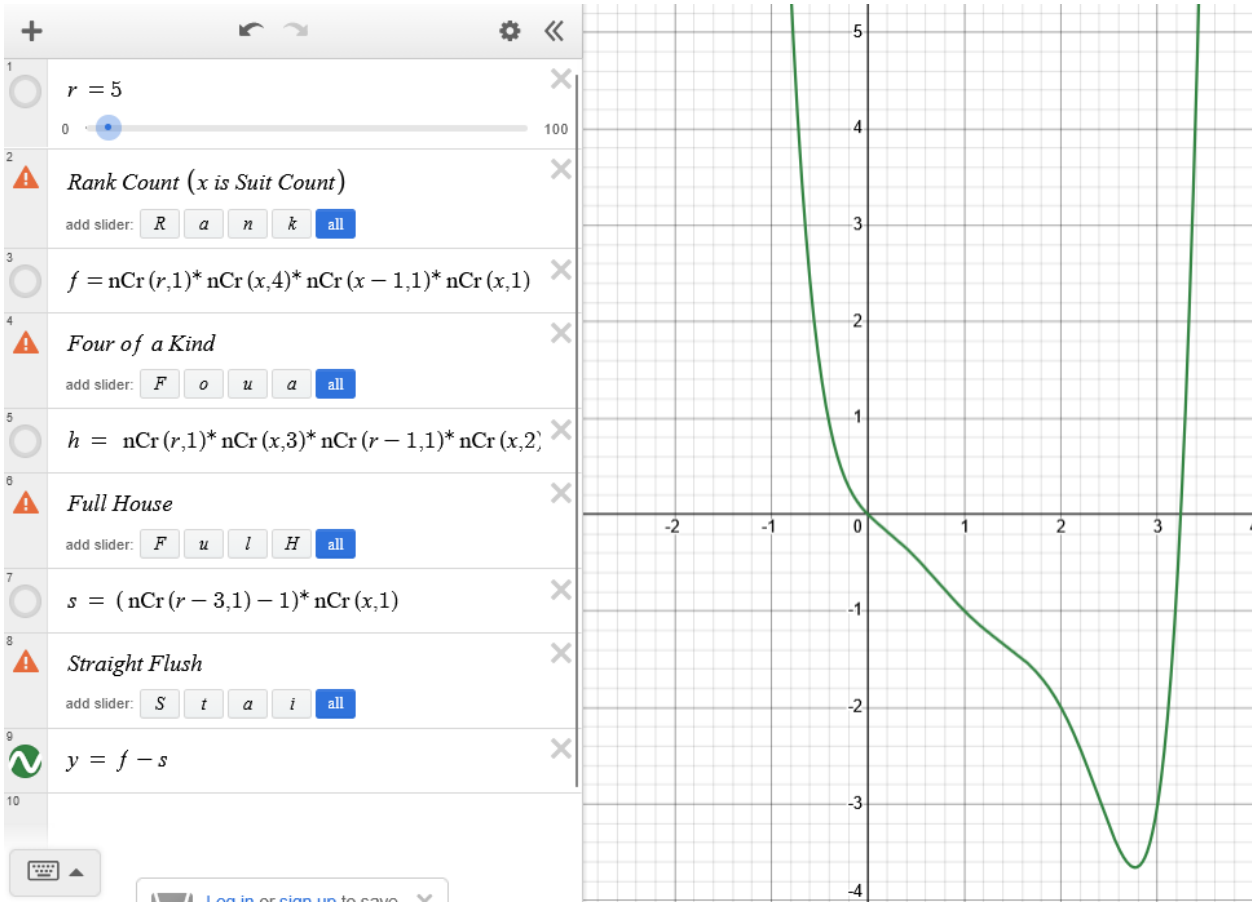


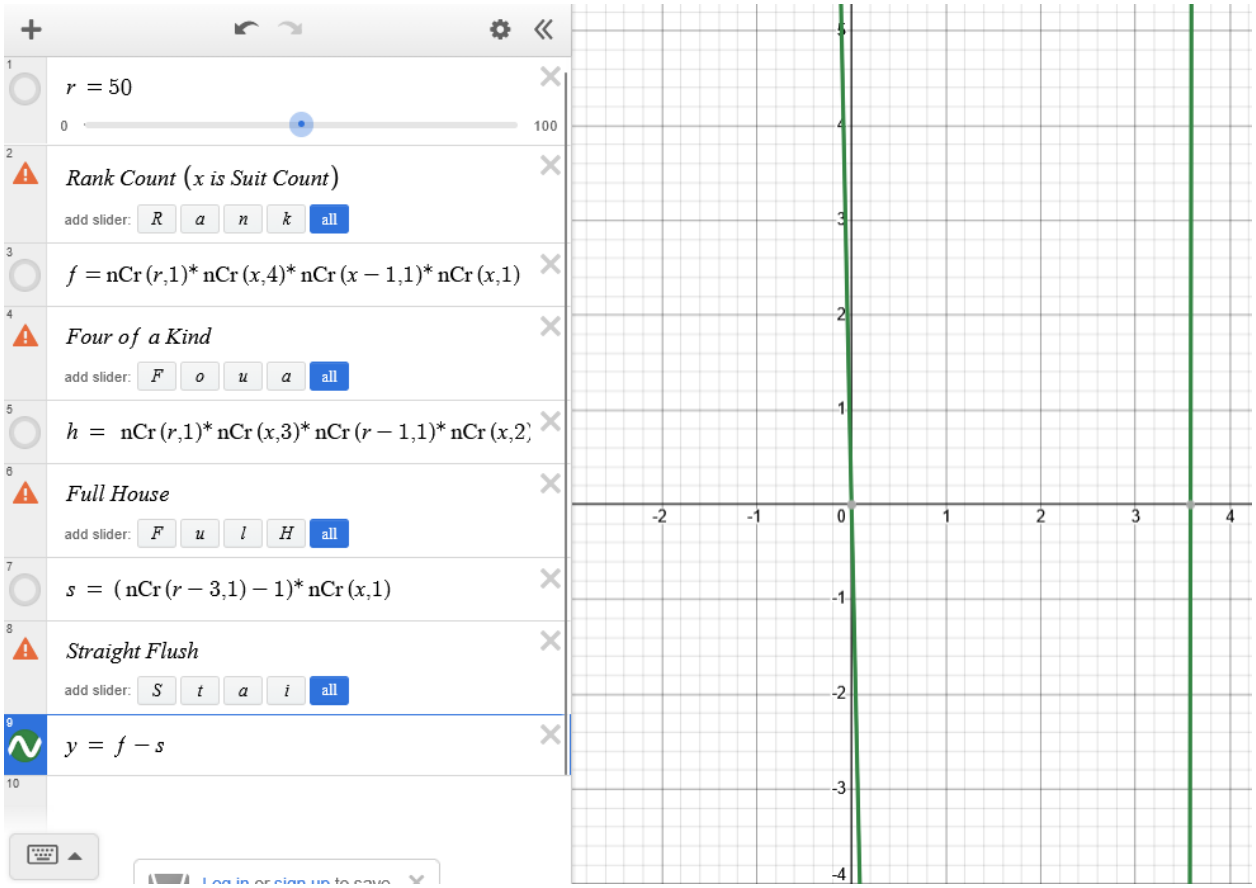
Full house will always be more common than four of a kind unless you have fewer suits than you need to make either of them (degenerate cases), so always. (This also happens with other rank counts, as seen below.)



Therefore, Straight Flush < Full House < Four of a Kind, Full House < Straight Flush < Four of a Kind, and Full House < Four of a Kind < Straight Flush are not possible orders.
Also, using this method, I will ignore all negative and/or non-integer x-values (number of suits).

Straight Flush vs Four of a Kind

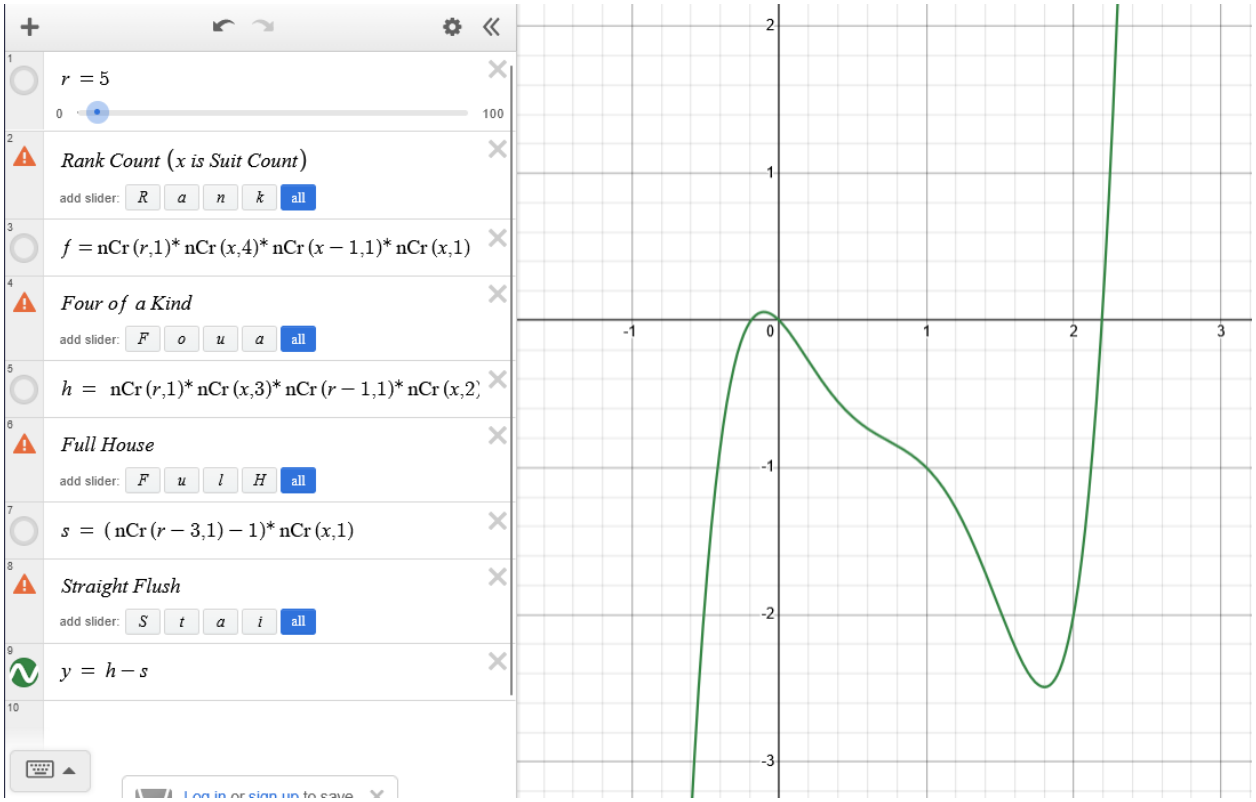

r = 5
0
100
Rank Count (x is Suit Count)
add slider: R a n k all
f = nCr (r,1)* nCr (x,4)* nCr (x − 1,1)* nCr (x,1)
Four of a Kind
add slider: F o u a all
h = nCr (r,1)* nCr (x,3)* nCr (r − 1,1)* nCr (x,2)
Full House
add slider: F u l H all
s = (nCr (r − 3,1) − 1)* nCr (x,1)
Straight Flush
add slider: S t a i all
y = f − s
5
4
3
2
1
-2
-1
0
1
2
3
-1
-2
-3
-4



No matter how much the ranks change, the difference is always positive and increases by x = 4, so when there are enough suits to actually make a Four of a Kind rather than being a degenerate case. Because of this, we know that Straight Flushes are always rarer than Four of a Kind (when both are possible) .
Therefore, Four of a Kind < Straight Flush < Full House and Four of a Kind < Full House < Straight Flush are not possible orders.

So the only possible order is the one we know now, Straight Flush < Four of a Kind < Full House

3. In the last part I said there were 6 orderings. That is, if we don't include ties. If ties are allowed then there are 13 orderings. For which of these 13 ways is there an r, s such that if you have r ranks and suites, then you get that ordering of probabilities.
As seen earlier, Four of a Kind and Straight Flush cannot tie (the only times the difference is zero are when it is literally impossible to make four of a kind because there's not enough suits, or when there is a non-integer number of suits) and the same for Four of a Kind and Full House (again, only crosses zero when there's too few suits to make either of them). In regards to Straight Flush and Full House, since Four of a Kind is always strictly rarer than Full House and Straight Flush is always strictly rarer than Four of a Kind, we can infer that Straight Flush must always be rarer than Full House. Still, we can double-check to be sure:



Again, the difference only is negative or zero when there's just not enough suits to make a Full House or the number of suits is not a whole number.
Therefore, considering ties doesn't change anything, there is still only one possible order: Straight Flush < Four of a Kind < Full House.

4. CHECK your work by using simulations.

Here's a quick simulation that checks if there are any alternative orders starting from 5 ranks and 4 suits (so that Straight Flushes and Four of a Kinds are possible) up to 10,000 ranks and 10,000 suits. It found none.

main.py — Share — Run — Output

```
from math import comb

wrong = False

for r in range (5, 10000):
    for s in range(4, 10000):
        fh = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) * comb(s, 2)
        fk = comb(r, 1) * comb(s, 4) * comb(r - 1, 1) * comb(s, 1)
        sf = (comb(r - 4, 1) - 1) * comb(s, 1)
        if not(sf < fk and fk < fh):
            print("Discrepancy!")
            wrong = True
            print("ranks: " + r)
            print("suits: " + s)

if wrong == False:
    print("Simulation Agrees. ")
```

```
Simulation Agrees.

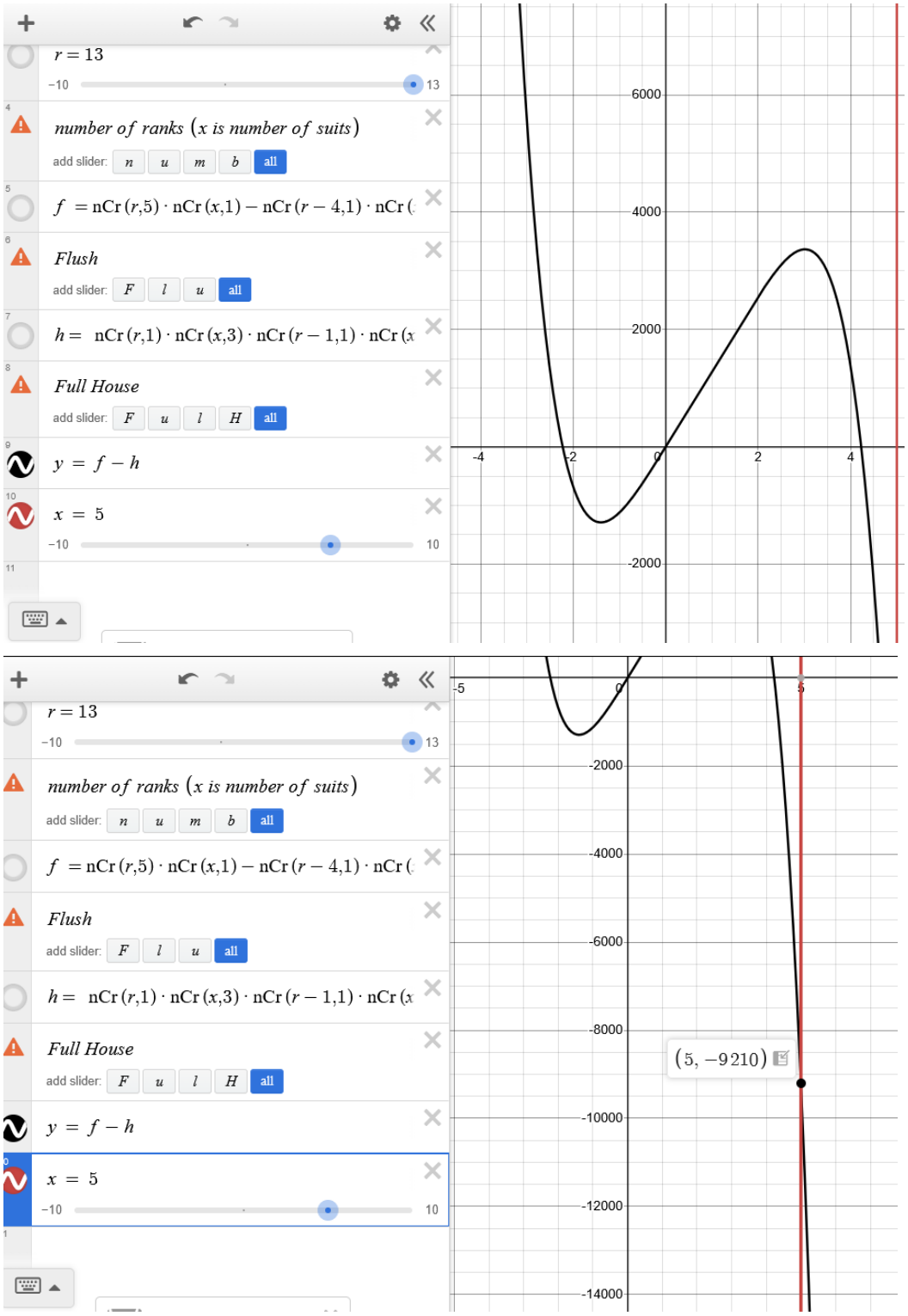
=== Code Execution Successful ===
```

5. Note that the three hands Straight Flush, 4-of-a-kind, Full House have no obvious order. So its plausible that any order could happen (this may not be true, but it could be true). By contrast, if I had chosen the three hands: 1-pair, 2-pair, 3-of-a-kind then clearly there is only one ordering. There are other combinations whose ordering is not obvious. I will list some. There may be others. For each set of hands, where any ordering is plausible, do what we did above for Straight Flush, Four-of-a-kind, Full house. Here are some to look at:

- **Full House, Flush, Straight**
- **Flush, Straight, 3-of-a kind**
- **No pair, one-pair**

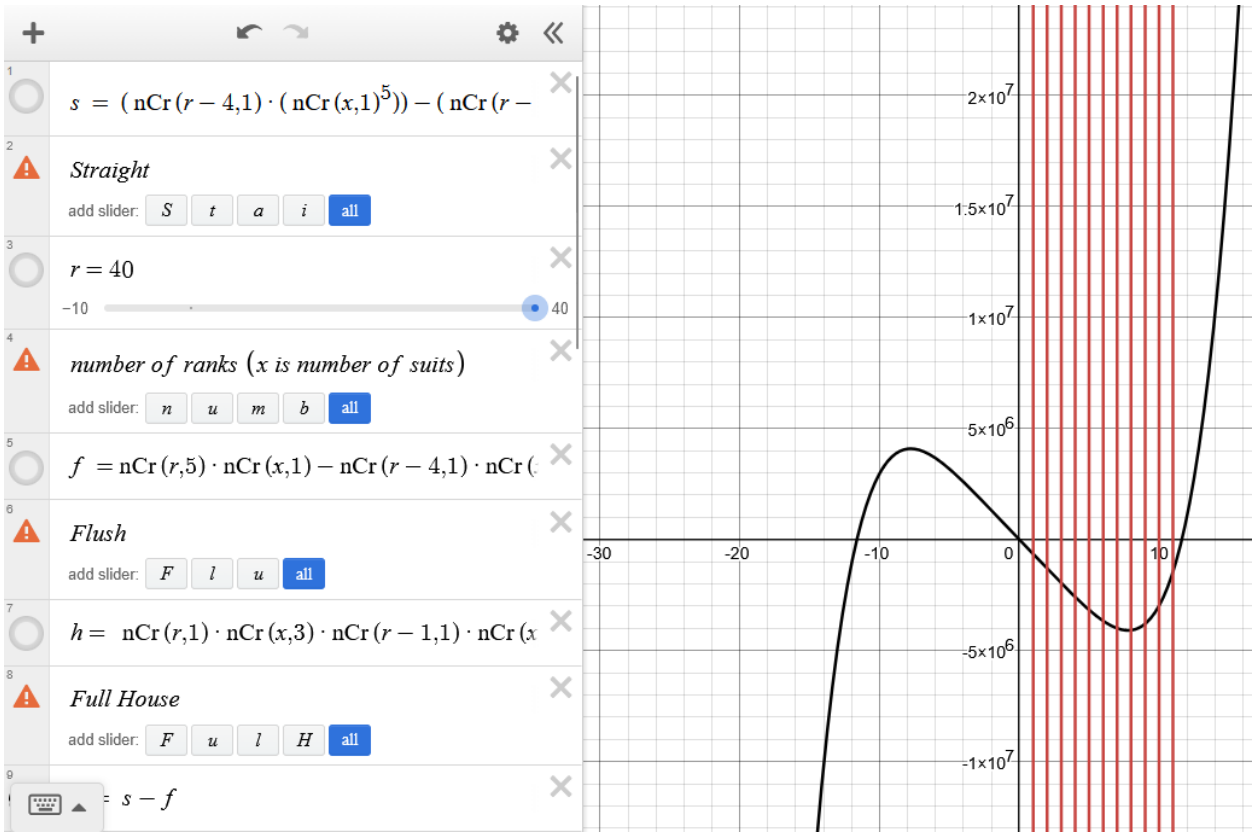
The denominator is the same for all of these, at all possible hands/nCr(rs, 5). Because of this, I can just focus on the numerator/number of ways to actually get each hand in the following calculations. Following this, the relevant formulas for the first group here are:
Straight: (nCr(r - 3, 1) * (nCr(s, 1)^5)) - (nCr(r - 3, 1)) * nCr(s, 1)
Flush: (nCr(r, 5) * nCr(s, 1)) - (nCr(r - 3, 1) * nCr(s, 1))
Full House: nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 2)
Currently: **Full House < Flush < Straight**
Possibilities: **Flush < Straight < Full House**, **Flush < Full House < Straight**, **Full House < Straight < Flush**, **Straight < Full House < Flush**, **Straight < Flush < Full House**

Note: for the graphs attached to justify claims and show reasoning, x represents the number of suits, the number of ranks is a variable that I can manually change the value of, and in most cases y represents the difference between the number of possibilities for each hand. Relevant formulas, especially if not explicitly listed in the text of this document, are visible on the left-hand side of the graphs.
Flush vs Full House

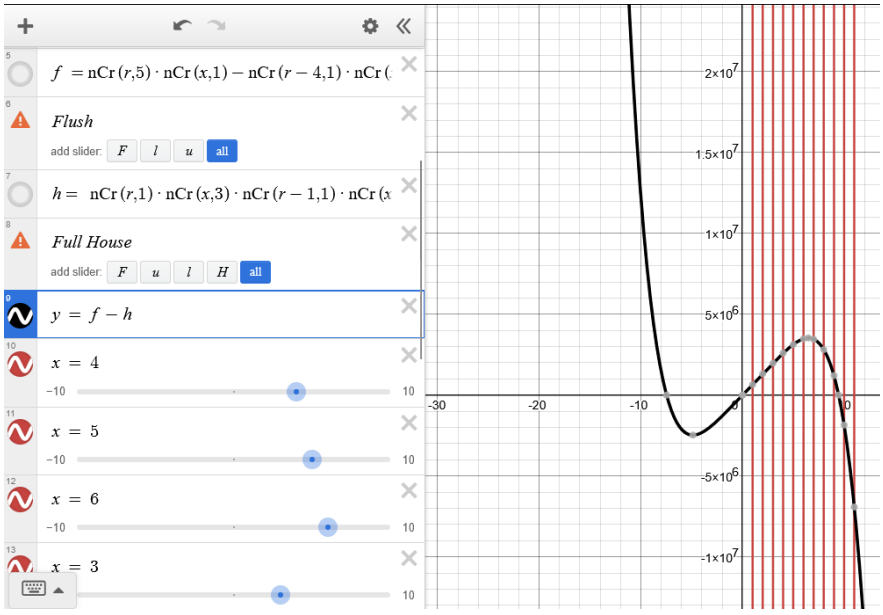


Even after adding just one suit and not changing the number of ranks, there are 9,210 more ways to make a full house than to make a flush (shown by the point (5, -9,210) being on the line; 5 suits causes the difference between flush possibilities and full house possibilities to be -9,210 with ranks remaining static at 13). Clearly, these orders can change, and we know for certain that there is at least one new possible order: **Flush < Full House < Straight**.

Flush vs Straight



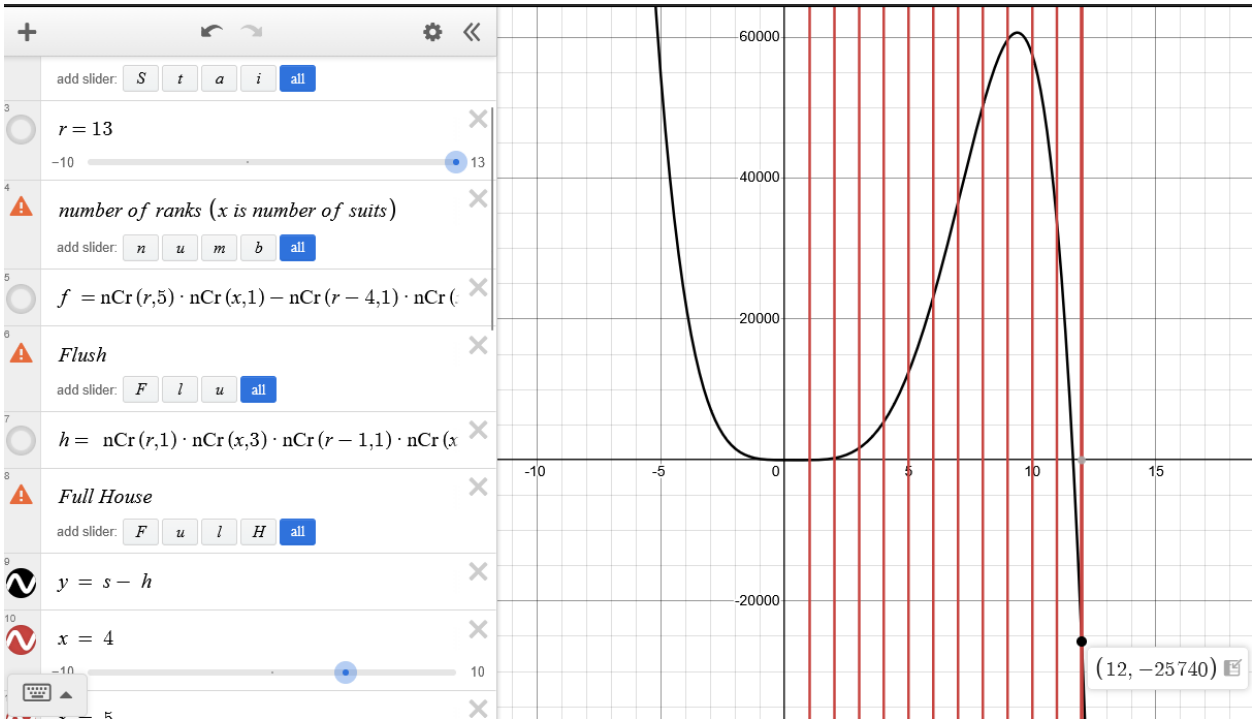
(the red lines are acceptable values for suit numbers(natural numbers); 1, 2, 3, through 11) Having a higher number of ranks makes flushes far more likely than straights, which makes sense: there's more possible cards in each suit that can be combined into flushes, and while the overall odds of straights will increase, it is a much less direct increase. I'm not sure exactly where the border is–how many ranks you need to change the order like this–but I do know that switching flushes and straights in probability/order is possible.



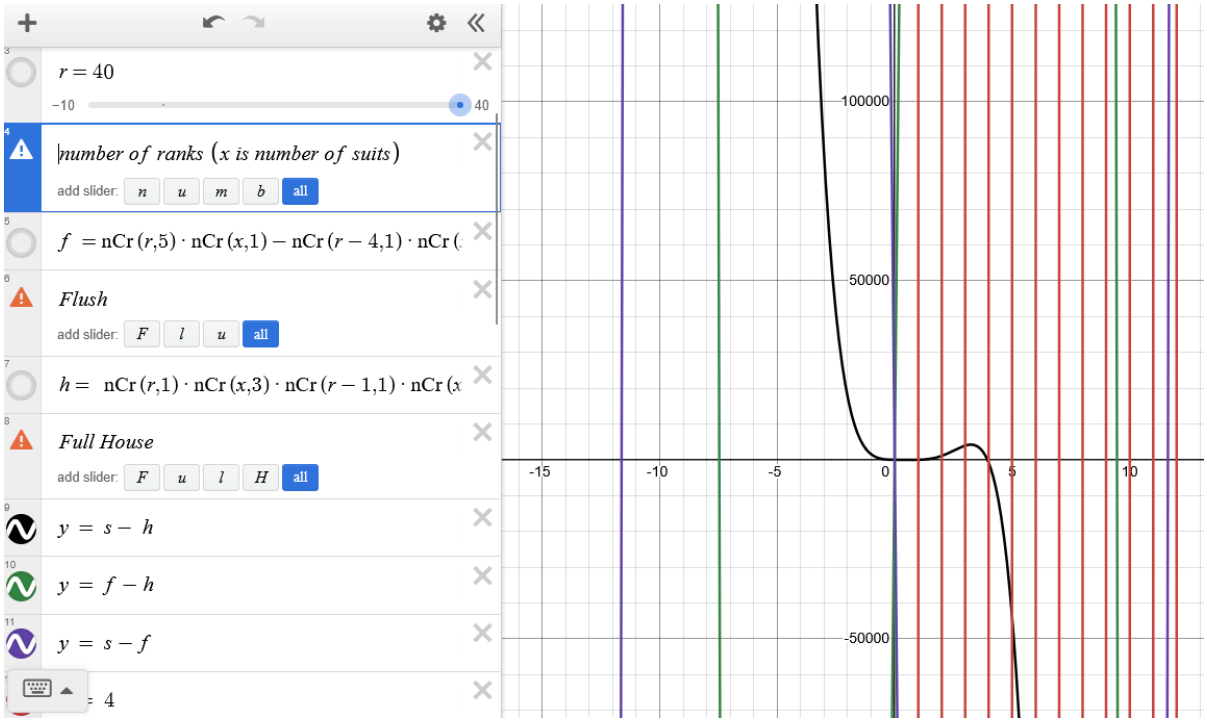
(Also, it is possible for straights and flushes to switch order under the same parameters as full houses and flushes switch order.)

Therefore, we know that **Full House < Straight < Flush** and **Straight < Flush < Full House** are both possible orders.

Full House vs Straight

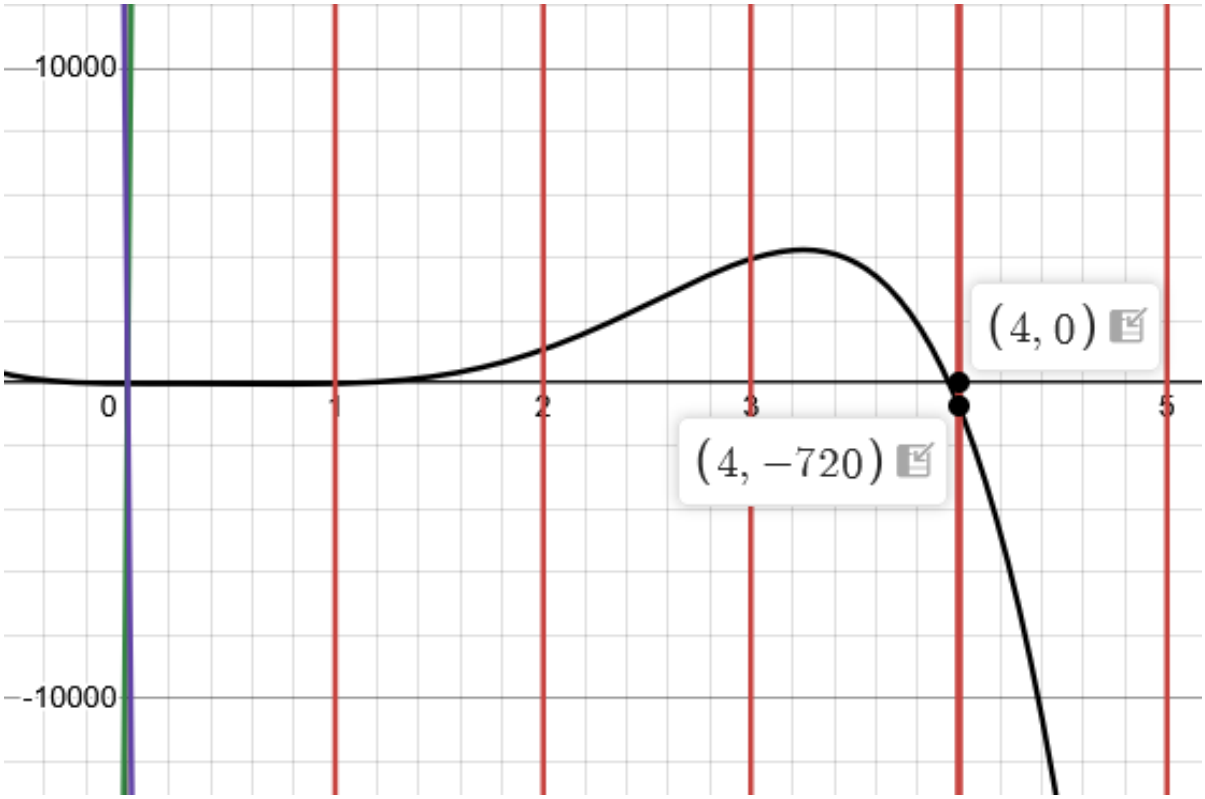


We can see straights become rarer than full houses with just the normal 13 ranks when the number of suits is 12, so we know they *can* switch–but what about the flush in between them? Can straights and full houses switch rarity when flush probabilities are also different?



Well, here I plotted the difference between each pair out of these 3 hands when the number of ranks is 40, which we know can make flushes more likely. Straights become rarer than full houses much faster.

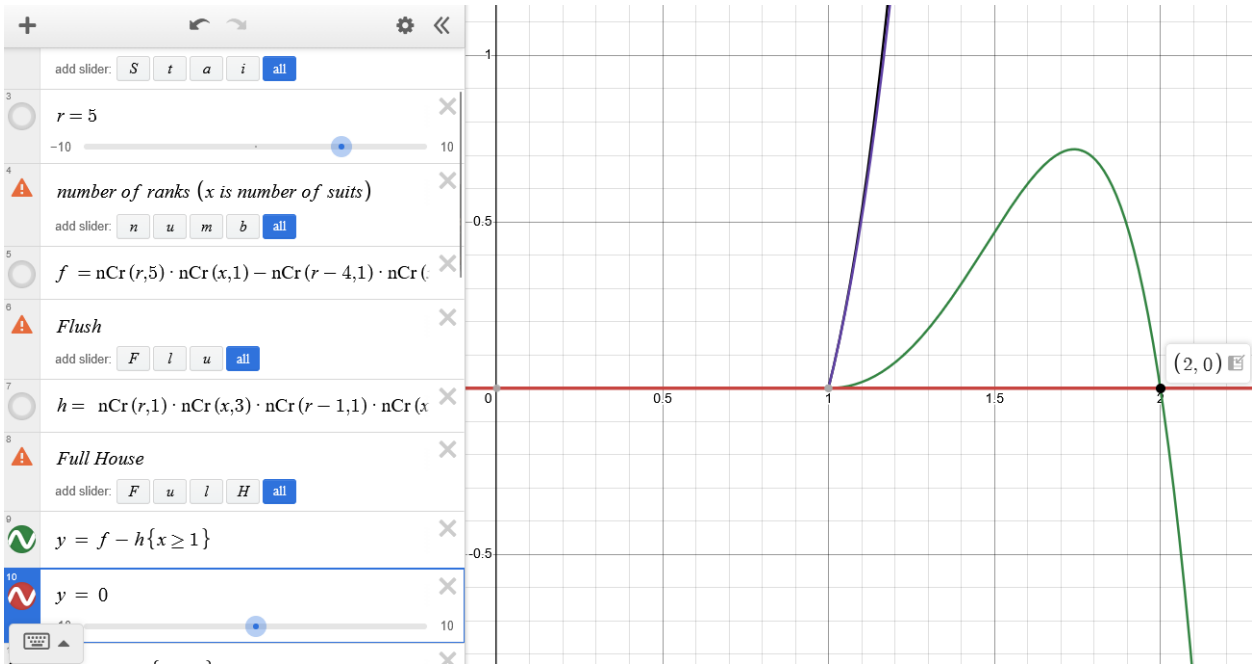
And the scaling is weird for the other differences, but:





We can see sign changes in all three differences with a constant number of ranks and in the same ranges of suit values. Therefore, we know that these three hand types are interchangeable in terms of probability when rank and suit numbers are changed, and the remaining two orders are also possible, leaving a total of 6 possible orders for these!

…Now to consider ties.



Flushes and Full Houses can tie in probability when there are 5 ranks and 2 suits. After determining this, I realized how complicated it would be to check all possible tie conditions and if they can happen at the same time, so I made a program that runs through the possibilities and lets me know of any ties it finds.



```
from math import comb

for r in range (5, 100):
    for s in range(3, 100):
        straight_prob = comb(r - 3, 1) * (comb(s, 1)**5) - (comb(r - 3, 1) * comb(s, 1))
        flush_prob = (comb(r, 5) * comb(s, 1)) - (comb(r - 3, 1) * comb(s, 1))
        full_house_prob = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) * comb(s, 2)

        if straight_prob - flush_prob == 0 or straight_prob - full_house_prob == 0 or full_house_prob - flush_prob == 0:
            print("Tie")
```

```
=== Code Execution Successful ===
```

[This code will be transcribed at the end of this section for convenience.]
I tested the program for anything we didn't expect according to the existing order, so I know it can find a lot of possibilities, but it found none here. This program did make me realize a flaw with the tie I previously found, however: neither a flush nor a full house is possible with these numbers. A flush can't happen–well, it can, but it can only ever be a straight flush (and a royal flush at that) when there are only 5 ranks, which is a better hand. A full house also can't happen; 3 suits are needed in order to get three cards of the same rank for the full house. Based on this, I conclude that there are no ties possible for this set of hands.

The formulas for the next group are:
Three of a Kind: (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 1) * nCr(r - 2, 1) * nCr(s, 1))
Straight: (nCr(r - 3, 1) * (nCr(s, 1)^5)) - (nCr(r - 3, 1)) * nCr(s, 1)
Flush: (nCr(r, 5) * nCr(s, 1)) - (nCr(r - 3, 1) * nCr(s, 1))
I think I've proved enough relationships using math to prove that I understand how to do this mathematically, and since it will be more efficient to have a program check the remaining possibilities for me, I decided to do that.

```
                    orderlist.append("3-of-a-kind")
                if probslist[i] == straight_prob:
                    orderlist.append("straight")
                if probslist[i] == flush_prob:
                    orderlist.append("flush")
            if not orderlist in possibilities:
                possibilities.append(orderlist)
                threeks.append(three_kind_prob)
                straights.append(straight_prob)
                flushes.append(flush_prob)
                suitcounts.append(s)
                rankcounts.append(r)
        else:
            print("Tie")
            print("Straight Number: " + str(straight_prob) +
                "\nFlush Number: " + str(flush_prob) + "\nThree
                of a Kind Number: " + str(three_kind_prob))
            print(probslist)
            print("rank count: " + str(r) + "\nsuit count: " +
                str(s))
            print("")

print(possibilities)
print(suitcounts)
print(rankcounts)
```

```
[['flush', '3-of-a-kind', 'straight'], ['flush', 'straight', '3-of-a
    -kind'], ['3-of-a-kind', 'flush', 'straight'], ['3-of-a-kind',
    'straight', 'flush'], ['straight', 'flush', '3-of-a-kind'],
    ['straight', '3-of-a-kind', 'flush']]
[2, 5, 2, 2, 3, 3]
[5, 5, 6, 9, 12, 17]

=== Code Execution Successful ===
```

(full code will be attached at the bottom of this section)
The program put all unique (non-tying) orders that it found into a list, which contains 6 orders: **Flush < Straight < Three of a Kind**, which we are used to, **Three of a Kind < Flush < Straight**, **Three of a Kind < Straight < Flush**, **Straight < Flush< Three of a Kind**, **Straight < Three of a Kind < Flush**, and **Flush < Three of a Kind < Straight**.

Finally, no-pair/high card and one-pair.
One Pair: nCr(r, 1) * nCr(s, 2) * nCr(r - 1, 3) * (nCr(s, 1)^3)
No Pair: nCr(rs, 5) - (nCr(r, 1) * nCr(s, 2) * nCr(r - 1, 3) * (nCr(s, 1)^3)) - (nCr(r, 2) * nCr(s, 2) * nCr(s, 2) * nCr(r - 2, 1) * nCr(s, 1)) - (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 1) * nCr(r - 2, 1) * nCr(s, 1)) - (nCr(r - 3, 1) * (nCr(s, 1)^5)) - ((nCr(r, 5) * nCr(s, 1)) - (nCr(r - 3, 1) * nCr(s, 1))) - (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 2)) - (nCr(r, 1) * nCr(s, 4) * nCr(r - 1, 1) * nCr(s, 1))

```
        1)
        fh = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) * comb(s,
            2)
        f = comb(r, 5) * comb(s, 1)
        st = (comb(r - 3, 1) * comb(s, 1)**5) - (comb(r - 3, 1)
            * comb(s, 1))
        tk = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) * comb(s,
            1) * comb(r - 2, 1) * comb(s, 1)
        tp = comb(r, 2) * comb(s, 2) * comb(s, 2) * comb(r - 2,
            1) * comb(s, 1)

        one_pair_prob = comb(r, 1) * comb(s, 2) * comb(r - 1, 3
            ) * (comb(s, 1)**3)
        no_pair_prob = comb(r*s, 5) - sf - fk - fh - f - st -
            tk - tp - one_pair_prob

        if one_pair_prob == no_pair_prob and not found1:
            print("tie")
            found1 = True
        elif one_pair_prob < no_pair_prob and not found2:
            print("one pair < no pair")
            found2 = True
        elif one_pair_prob > no_pair_prob and not found3:
            print("no pair < one pair")
            found3 = True

```

```
no pair < one pair
one pair < no pair

=== Code Execution Successful ===
```

(full code will be attached at the bottom of this section)
The code did not find any ties, but both **No Pair < One Pair** and **One Pair < No Pair** were possible–which is a little funny, that in some cases a high card could be a better hand than a pair.

Full code for Full House vs Flush vs Straight Tie:

```
from math import comb

for r in range (5, 100):
    for s in range(3, 100):
        straight_prob = comb(r - 3, 1) * (comb(s, 1)**5) - (comb(r - 3,
1) * comb(s, 1))
        flush_prob = (comb(r, 5) * comb(s, 1)) - (comb(r - 3, 1) *
comb(s, 1))
        full_house_prob = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) *
comb(s, 2)

        if straight_prob - flush_prob == 0 or straight_prob -
full_house_prob == 0 or full_house_prob - flush_prob == 0:
            print("Tie")
```

Full code for ThreeKind vs Straight vs Flush:

```
from math import comb

possibilities = []
threeks = []
straights = []
flushes = []
suitcounts = []
rankcounts = []

for r in range(5, 100):
    for s in range(2, 100):
        three_kind_prob = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) *
comb(s, 1) * comb(r - 2, 1) * comb(s, 1)
        straight_prob = ((comb(r - 3, 1) * comb(s, 1))**5) - (comb(r -
3, 1) * comb(s, 1))
        flush_prob = (comb(r, 5) * comb(s, 1)) - (comb(r - 3, 1) *
comb(s, 1))

        probslist = [three_kind_prob, straight_prob, flush_prob]
        probslist.sort()
        orderlist = []
        if not(three_kind_prob - straight_prob == 0 or three_kind_prob
- flush_prob == 0 or flush_prob - straight_prob == 0):
```

```
            for i in range(0, 3):
                if probslist[i] == three_kind_prob:
                    orderlist.append("3-of-a-kind")
                if probslist[i] == straight_prob:
                    orderlist.append("straight")
                if probslist[i] == flush_prob:
                    orderlist.append("flush")
            if not orderlist in possibilities:
                possibilities.append(orderlist)
                threeks.append(three_kind_prob)
                straights.append(straight_prob)
                flushes.append(flush_prob)
                suitcounts.append(s)
                rankcounts.append(r)
        else:
            print("Tie")
            print("Straight Number: " + str(straight_prob) + "\nFlush
Number: " + str(flush_prob) + "\nThree of a Kind Number: " +
str(three_kind_prob))
            print(probslist)
            print("rank count: " + str(r) + "\nsuit count: " + str(s))
            print("")

print(possibilities)
print(suitcounts)
print(rankcounts)
```

Full code for one pair vs no pair:

```
from math import comb

found1 = False
found2 = False
found3 = False

for r in range(5, 1000):
    for s in range(2, 1000):
        sf = comb(r - 3, 1) * comb(s, 1)
        fk = comb(r, 1) * comb(s, 4) * comb(r - 1, 1) * comb(s, 1)
        fh = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) * comb(s, 2)
        f = comb(r, 5) * comb(s, 1)
        st = (comb(r - 3, 1) * comb(s, 1)**5) - (comb(r - 3, 1) *
comb(s, 1))
        tk = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) * comb(s, 1) *
comb(r - 2, 1) * comb(s, 1)
```

```
        tp = comb(r, 2) * comb(s, 2) * comb(s, 2) * comb(r - 2, 1) *
comb(s, 1)

        one_pair_prob = comb(r, 1) * comb(s, 2) * comb(r - 1, 3) *
(comb(s, 1)**3)
        no_pair_prob = comb(r*s, 5) - sf - fk - fh - f - st - tk - tp -
one_pair_prob

        if one_pair_prob == no_pair_prob and not found1:
            print("tie")
            found1 = True
        elif one_pair_prob < no_pair_prob and not found2:
            print("one pair < no pair")
            found2 = True
        elif one_pair_prob > no_pair_prob and not found3:
            print("no pair < one pair")
            found3 = True
```

6. Everything above is for 5-card hands. Do the same thing for 6-card hands. You will need to consider other good hands like

- **Three pairs**
- **Two 3-of-a-kind**
- **A pair and 4-of-a-kind**
- **There may be others**

Pre-existing Hands:

Royal Flush: nCr(s, 1)/nCr(rs, 6)

All straights now must have 6 cards in order, though that isn't particularly relevant in this specific calculation. The same is true for flushes as well. But still, this formula is extremely similar to the royal flush formula in 5-card hands.

Straight Flush: ((nCr(r - 4, 1) - 1) * nCr(s, 1))/nCr(rs, 6)

Here is the first change from 6-card hands beyond the denominator: the highest straight starts at r - 4 now (r - 5, r - 4, r - 3, r - 2, r - 1, r and then add back one possibility for Ace-low straights). Other than these two changes, this formula is identical to the previous one.

Four of a Kind: (nCr(r, 1) * nCr(s, 4) * nCr(r - 1, 2) * $nCr(s, 1)^2$))/nCr(rs, 6)

This formula differs from the previous one in two main ways: the denominator, like all of these new formulas will, and the kicker(s). There is a new kicker to account for, and it must specifically be a different rank from the other kicker to distinguish this hand from the new hand of 4-of-a-kind + a pair.

Full House: (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 2) * nCr(r - 2, 1) * nCr(s, 1)/nCr(rs, 6)

The adjustment we make specifically for Full House is adding a kicker, which we have to do to avoid the new hand of two 3-of-a-kinds. There isn't much else to say here.

Flush: (nCr(r, 6) * nCr(s, 1))/nCr(rs, 6) - (nCr(r - 4, 1) * nCr(s, 1))/nCr(rs, 6)

Like I said earlier, we need *six* cards for a flush now, so must choose 6 different ranks to be in the same suit. We change the denominator–or, denominator*s*, since we have to remove the better hand of a straight flush.

Straight: (nCr(r - 4, 1) * $(nCr(s, 1)^6)$)/nCr(rs, 6) - (nCr(r - 4, 1) * nCr(s, 1))/nCr(rs, 6)

Similar to flush, we remove the odds of a straight flush, and similar to straight flush, we have to change r - 3 to r - 4. Additionally, we choose a suit for each card 6 individual times, changing an exponent.

Three of a Kind: (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 3) * $nCr(s, 1)^3$)/nCr(rs, 6)

We just add another kicker, which must be a different rank to avoid four of a kind or full house. I combined the suit selections into an exponent for convenience.

Two Pair: (nCr(r, 2) * nCr(s, 2) * nCr(s, 2) * nCr(r - 2, 2) $nCr(s, 1)^2$)/nCr(rs, 6)

We add another kicker and make sure it isn't the same rank as any other cards to avoid a three of a kind or the new, better hand of three pair.

Pair: (nCr(r, 1) * nCr(s, 2) * nCr(r - 1, 4) * $(nCr(s, 1)^4)$)/nCr(rs, 6)

One pair and four other cards of different ranks which we choose suits for individually. …I'll come back to high card later, since I essentially need to subtract every other probability from 1, and I need to address new possible hands.

New Hands:

Three pair: (nCr(r, 3) * $nCr(s, 2)^3$)/nCr(rs, 6)

Choose 3 ranks, choose 2 suits 3 times, and divide by all possible hands.

Two 3-of-a-kind: (nCr(r, 2) * $nCr(s, 3)^2$)/nCr(rs, 6)

Choose 2 ranks, choose 3 suits twice, and divide by all possible hands.

4-of-a-kind + pair: (nCr(r, 2) * nCr(s, 4) * nCr(s, 2))/nCr(rs, 6)

Choose 2 ranks, choose 4 suits for one of those ranks and 2 suits for the other, and divide by all possible hands.

While 5-of-a-kind and 6-of-a-kind are technically possible since we're flexible with the number of suits, we didn't consider 5-of-a-kind for 5-card hands so I'm not considering it or 6-of-a-kind here.

Pre-existing Hands But It's Just No Pair/High Card:

High card: 1 - (nCr(r, 1) * nCr(s, 4) * nCr(r - 1, 1) * nCr(s, 1) * nCr(r - 2, 1) * nCr(s, 1))/nCr(rs, 6) - (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 1) * nCr(s, 2) * nCr(r - 2, 1) * nCr(s, 1)/nCr(rs, 6) - (nCr(r, 6) * nCr(s, 1))/nCr(rs, 6) - ((nCr(r - 4, 1) * $(nCr(s, 1)^6)$)/nCr(rs, 6) - (nCr(r - 4, 1) * nCr(s, 1))/nCr(rs, 6))

- (nCr(r, 1) * nCr(s, 3) * nCr(r - 1, 3) * nCr(s, 1)^3)/nCr(rs, 6) - (nCr(r, 2) * nCr(s, 2) * nCr(s, 2) * nCr(r - 2, 2) nCr(s, 1)^2)/nCr(rs, 6) - (nCr(r, 1) * nCr(s, 2) * nCr(r - 1, 4) * (nCr(s, 1)^4))/nCr(rs, 6) - (nCr(r, 3) * nCr(s, 2)^3)/nCr(rs, 6) - (nCr(r, 2) * nCr(s, 3)^2)/nCr(rs, 6) - (nCr(r, 2) * nCr(s, 4) * nCr(s, 2))/nCr(rs, 6)

1 minus everything. That's all.

7. We won't be looking at 7-card hands as that leads to too many new hands to look at.
Okay.

8. (This is not about cards.) How many ways can 4 horses finish a horse race, allowing ties? 5 horses?
4 horses can finish a race without ties 4! = 24 ways. Addressing ties is a little trickier.
If all 4 horses tie, that's just 1 additional outcome. If 3 of the 4 horses tie, the remaining horse can place before all of them or behind all of them, so that's 2 more possibilities–but the remaining horse could be any individual of the 4, so multiply that by four for 8 new outcomes. If 2 of the horses tie, the remaining horses can be first and second, first and last, or third and last–but the order of the non-tying horses does matter, so each of those are two possibilities, one with horse A performing better and one with horse B performing better. Also, horses A and B could be horses C and D; it matters which horses tie and which don't. We're picking 2 horses out of 4–either to tie, where the order doesn't matter, or to not tie, where I've already accounted for a different order. Therefore, I can use a combination: 3 * 2 * nCr(4, 2) more outcomes if just two horses tie. And if two sets of horses individually tie (as opposed to all four horses tying–like 4-kind vs 2-pair), that's two more possibilities per set of two horses chosen out of the four, so 2* nCr(4, 2), but we then divide by 2 because of overcounting cases (horses A, B tying and coming first is the same as horses C, D tying for last). After grinding out the math and adding it all up, we find our answer: 75.

For 5 horses, we go through the same process; factorial + all possible different numbers of horses that can tie * all possible ways the horses can be organized into those numbers
No ties = 5! = 120
5 horses tie = +1
4 horses tie = +10 (5 possible non-tying horse combinations (nCr(5,1) to lay the framework used for other ties) times 2 (2! For the same framework reasons (treat tied horses like 1 horse)) non-tying position possibilities)
3 horses tie = +60 (nCr(5, 2) = 10 non-tying horse combinations times 3! = 6 non-tying position possibilities)
2 horses tie = +240 (nCr(5, 3) = 10 non-tying horse combinations times 4! = 24 non-tying)
2 horses tie and other 3 horses also tie = +20 (nCr(5, 2) = 10 combinations for how the horses are grouped times 2 possibilities per combination)
2 horses tie and 2 other horses tie and also there's another horse = +90 (nCr(5, 2)*nCr(3,2)/2 = 15 ways to pick out the tying groups (divide by two to not re-count the cases where tying group A is tying group B) times 3! = 6 ways to order tying group A, tying group B, and the individual horse)
Total possibilities = 541.

Horse-Race Side Project Extension: Write a program that will, given n, calculate the first n horse numbers (H(n) is the number of ways n horses can finish a race, and we allow ties).

Attached here is the requested program, written as a function horseRace(n) that returns an integer. This code contains an import statement for python's math module and a few print statements in the program's body rather than the function for the purpose of testing.

```
import math

def horseRace(n):
    ways = 0 # baseline for variable that will be returned

    for k in range(0, n + 1): # summation formula
        s = 0 # s is for Stirling number of the second kind
        for j in range(0, k + 1):
            s += (math.pow(-1, k - j) * (j**n))/(math.factorial(k -
j)*math.factorial(j))

        ways += math.factorial(k) * s

    ways = round(ways, 0) # i'm getting 75.0 and 541.0000000000005

    output = int(ways) # getting rid of the .0 at the end because it
annoys me

    return(output)


for i in range(0, 11):
    print(str(i) + " horses: " + str(horseRace(i)))
```

Appendix A: 5-card Hand Orderings

My initial plan for this appendix was to manually check if the conditions I found for, for example, flushes and straights switching orders could also allow pairs and no pairs to switch orders, and then compile the total possible combinations based just on that. I figured writing a program to check the conditions for me would be easier, and ended up finding it very straightforward to just write a program that finds all orderings that can actually happen by iterating through different amounts of suits and ranks, starting from 5 ranks and 4 suits to ensure that all hands (ex. Straights and 4-of-a-kinds) were always possible. I will attach this code at the bottom of the appendix. Here are the 25 orderings it found:

Flush, Straight Flush, Four of a Kind, Full House, No Pair, Three of a Kind, Straight, Two Pair, Pair

Flush, Straight Flush, Four of a Kind, Full House, No Pair, Straight, Three of a Kind, Two Pair, Pair
Flush, Straight Flush, Four of a Kind, No Pair, Full House, Straight, Three of a Kind, Two Pair, Pair
Straight Flush, Flush, Four of a Kind, Full House, Straight, Three of a Kind, No Pair, Two Pair, Pair
Straight Flush, Flush, Four of a Kind, Full House, Straight, No Pair, Three of a Kind, Two Pair, Pair
Straight Flush, Flush, Four of a Kind, Full House, Straight, Three of a Kind, Two Pair, No Pair, Pair
Straight Flush, Four of a Kind, Flush, Full House, Straight, Three of a Kind, Two Pair, No Pair, Pair
Straight Flush, Flush, Four of a Kind, Straight, Full House, Three of a Kind, Two Pair, No Pair, Pair
yuppp :p
Straight Flush, Four of a Kind, Full House, Flush, Straight, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Four of a Kind, Flush, Full House, Straight, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Flush, Four of a Kind, Full House, Straight, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Flush, Four of a Kind, Straight, Full House, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Four of a Kind, Full House, Straight, Flush, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Four of a Kind, Flush, Straight, Full House, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Flush, Straight, Four of a Kind, Full House, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Four of a Kind, Straight, Full House, Flush, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Four of a Kind, Straight, Flush, Full House, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Four of a Kind, Full House, Straight, Three of a Kind, Flush, Two Pair, Pair, No Pair
Straight Flush, Four of a Kind, Straight, Full House, Three of a Kind, Flush, Two Pair, Pair, No Pair
Straight Flush, Straight, Flush, Four of a Kind, Full House, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Straight, Four of a Kind, Flush, Full House, Three of a Kind, Two Pair, Pair, No Pair
Straight Flush, Straight, Four of a Kind, Full House, Flush, Three of a Kind, Two Pair, Pair, No Pair

Straight Flush, Four of a Kind, Straight, Full House, Three of a Kind, Two Pair, Flush, Pair, No Pair
Straight Flush, Straight, Four of a Kind, Full House, Three of a Kind, Flush, Two Pair, Pair, No Pair
Straight Flush, Straight, Four of a Kind, Full House, Three of a Kind, Two Pair, Flush, Pair, No Pair

Full code for all possible orderings:

```
from math import comb
import operator

orderings = []

for r in range(5, 1000):
    for s in range(4, 1000):
        sf = comb(r - 3, 1) * comb(s, 1)
        fk = comb(r, 1) * comb(s, 4) * comb(r - 1, 1) * comb(s, 1)
        fh = comb(r, 1) * comb(s, 3) * comb(r - 1, 1) * comb(s, 2)
        f = comb(r, 5) * comb(s, 1)
        st = (comb(r - 3, 1) * comb(s, 1)**5) - (comb(r - 4, 1) *
comb(s, 1))
        tk = comb(r, 1) * comb(s, 3) * comb(r - 1, 2) * (comb(s, 1)**2)
        tp = comb(r, 2) * comb(s, 2) * comb(s, 2) * comb(r - 2, 1) *
comb(s, 1)
        p = comb(r, 1) * comb(s, 2) * comb(r - 1, 3) * (comb(s, 1)**3)
        hc = comb(r*s, 5) - sf - fk - fh - f - s - tk - tp - p
        handProbs = {"Straight Flush":sf, "Four of a Kind":fk, "Full
House":fh, "Flush":f, "Straight":st, "Three of a Kind":tk, "Two
Pair":tp, "Pair":p, "No Pair":hc}

        for p in (0, len(handProbs)):
            hand = list(handProbs)[0]
            prob = list(handProbs.values())[0]
            names = ""
            removeList = []
            if list(handProbs.values()).count(prob) > 1:
                for i in range(0, len(list(handProbs)), 1):
                    if list(handProbs.values())[i] == prob:
                        names += list(handProbs)[i] + " and "
                        removeList.append(list(handProbs)[i])
                for l in removeList:
                    handProbs.pop(l)
                names = names[:len(names) - 5]
                names += " Tied"
                handProbs[names] = prob
```

```
        order = dict(sorted(handProbs.items(),
key=operator.itemgetter(1)))

        if not list(order.keys()) in orderings:
            orderings.append(list(order.keys()))

#print(len(orderings))
#print(orderings)
for o in orderings:
    printOut = o[0]
    for h in range(1, len(o)):
        printOut += ", " + o[h]
    print(printOut)
```