MaximizePayout

Asher Kirshtein

March 15, 2022

1 Algorithm Explained

My Algorithm for solving the issue is as follows: I simply want to make sure that the largest base matches up with the largest possible exponent.

How do I do that? Simply, I sort both of the sets we are given(can be sorted either way) but I do it from largest to smallest. I then iterate over the lists and take the $A_i^{B_i}$ and add all of them up and return it. It maximises the payout because if you take a number to an exponent the numbers increase at an exponential rate so therefore you want to increase at the exponential rate by the largest possible factors. That's why you want to have the highest number to the highest exponent.

2 Proof of Correctness

Let's try a proof by induction:

Right now we are trying to prove that giving the highest base the highest power will give you the highest result. So:

$k^{k+1} + (k+1)^k \leq (k+1)^{k+1} + k^k$

Base Case P(1):
$1^2 + 2^1 \leq 2^2 + 1^1$
$1 + 2 \leq 4 + 1$
$3 \leq 5$
Therefore our base case holds true

Let's assume this holds true for all k now let's prove this for $k+1$:
$k^{k+1} + (k+1)^k \leq (k+1)^{k+1} + k^k$
If you take the $\sqrt[k]{P(k)}$
$k^2 + k + 1 \leq (k+1)^2 + k$
Subtract by k on both sides
$k^2 + 1 \leq (k+1)^2$

Expand it/reverse foil
$k^2 + 1 \leq k^2 + 2k + 1$
$k^2 \leq k^2 + 2k$
$0 \leq 2k$
Since we are only dealing with positive numbers this is true
QED

3 Run-time

Run-time: O($NlogN$) (Assuming Math.Pow is O(1))

The first cost we need to pay is the sorting of both of the lists. Which is $NlogN$ for each of the lists. We then have a cost of iterating.(Since both lists are the same size it is 1 iteration for both). So the cost of that iteration is just O(N).(within the iteration we do Math.Pow but we are assuming it is O(1)). Technically it is O(N * 2NlogN) but today has a vowel so constants don't matter so it is linearithmic or O(NlogN).

Proof of optimality(greedy stays ahead):

In the optimal we still also need to sort in order to access our values in the array as fast as possible so our greedy algorithm stays equal so far.

Then we do n different equations so it continues to stay with the optimal algorithm

Since our algorithm is at least as good as any other solutions algorithm it must be optimal QED