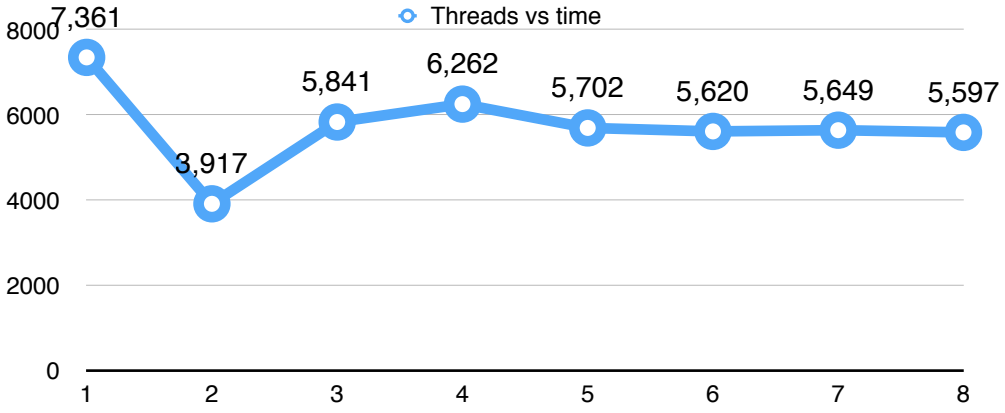
Exercise 2 Matan Gidnian 200846905

Part 1 - Question B



The number of thread that gave the best time: **2**

Number of cores I have on my CPU: **2**

The amount of threads is same to the amount of cores that I have on the CPU, taking to consideration the given task is not that complicated or time consuming. It's best to have a thread per core rather then making the CPU create more threads than actual physical cores which make the process slower and requires more time & power.

Part 1 - Question C

Creating this HUGE amount of threads will slower the running time as this is very inefficient and creates a lot of load on the CPU for very quick and small tasks.

Part 3

Question A

For start, the number is even, as if it were odd we had to add 1 somewhere in the process, which is not possible due to the equation.

x > 3 : Contrary assume that x < 4, taking this to consideration means that the last command to execute was of 0 or 2, therefore the instruction before was -2 or 0. -2 is not a possible value as we started with x = 0 and only added numbers on top of it. which means that it was 0, , meaning no adding was done, therefore contradiction the assumption this was the last command.

x < 69: Contrary assume that x > 68, X was incremented 2 times (as there are 2 threads) more then 17 times for each thread, we know that each thread was executing exactly 17 times, thus contradiction.

For any other number 'n' of the form n * 2, for 3 < x < 69, x = n by induction:

We proved x > 3, assuming that x is n * 2, and can also be n (as n < 69), then we know that at least one store instruction override other same instruction, or we would get x = 64, but as we know, there is a possible order that can avoid that, therefor x might be n. as needed.

Question B

1. **c = 5**

Thread 1 finished first then thread 2 executes.

2. **c = 14**

Thread 2 finished first then thread 1 executes.

3. **c = 6**

Thread 1 runs first into the IF, returned false does into the else, Thread 2 runs, Thread 1 finishes.

4. **c = 15**

Thread 1 runs first into the IF, returned false, thread 2 runs and sets b = 10, thread 1 continues and sets c = 10 + 5, thread 2 finished

5. **c = -4**

Thread 1 runs to line 4 and loads b as 0, thread 2 awakes and sets a = -4, thread 1 finished resulting in c = 0 - 4.

Question C

1) Because of the nested use of fork(), we will receive $\sum_{i=1}^{75} 2^i$ rows, in the following order:
x = 0, will be printed 2^1 times
x = 1, will be printed 2^2 times
x = 2, will be printed 2^3 times
…
x = 74, will be printed 2^75 times
And in generally "x = num, will be printed 2^(x+1) times", as the fork is nested, and every new fork is creating other fork thus multiplying the amount of printed rows and doubling with each iteration. Although we know what will be printed, the order of the lines is not fixed.
2) Here, 4 threads are created, and the possible options are: 777,557, 577, 377, 357.
Thread No1: will print 3 or 5 or 7.
Thread No2: will print 5 or 7.
Thread No3: will print 7.
Thread No4: will not enter the if statement as the addition & thread creation is done before printing.