You know that you need to find the minimum amount of time to complete p pizzas, with n chefs.

What if you check for all possible durations that can be taken by the chefs to complete the order? Further, how would you check this? Think in the direction of binary search.

If you take a range low to high, where low is 0 and high is the maximum possible time it can take to complete the order(or a very high value like 10^9 that can't be exceeded), you know that your solution (in minutes) will lie in between this range. Thus it effectively becomes a searching problem. Now, instead of linearly searching for all possible durations, wouldn't it be more efficient to use binary search?

Now, think about how you will update mid. There will need to be some modification, as you are not exactly searching for a value. So, you need to have a function 'check()' with arguments as number of chefs, number of pizzas, ranks of chefs and mid (minutes). This boolean function should check if for particular values of arguments, a solution is possible.

This will lead to 2 cases:

1) **A solution is possible.** If this is the case, then you know that in 'mid' minutes, the order can be completed by n chefs i.e it is a potential answer. But are you sure this is the minimum possible value? No. But think about this, if it is possible to do the task in 'mid' minutes, it will always be possible to do the task in time greater than mid. Right? Since we need to minimize our time, we need to check if it is possible to do the task in time less than mid. Therefore, we will reduce the size of our search space, i.e high = mid. So, new search space is low to mid.

2) **A solution is not possible.** If this is the case, it means that there is no possible configuration with which the task can be completed in 'mid' minutes. We can conclude that if the task cannot be done in 'mid' minutes, it can never be done in time less than 'mid'. So, again we will update our search space as low = mid, i.e new search space becomes mid to high.

Now we need to write the function check(). How will you check with given values of p, n (with ranks) and minutes (mid), if it's possible to complete the order or not?

There is another thing you need to keep in mind. For the given n chefs, you need to find out how many pizzas each chef can make. So, with a logic similar to the reverse of above (here we are maximizing), you need to maximize the number of pizzas each chef can make.

Note that since all chefs are working simultaneously, the total time for all chefs is the same and would be equal to 'mid' passed by the previous binary search function as an argument. You can calculate the time taken by a chef with rank r to make x pizzas using the given function F(x).

For every chef with rank r, we need to find the maximum number of pizzas they can make in a given fixed time.

How do we maximize the number of pizzas for every chef? Consider a binary search approach again. Take a range from low to high, for the number of pizzas. Calculate ‘mid2’ for this and think about when you will update it. **Your condition is that F(x) should be less than or equal to ‘mid’ because time taken by any chef to make x pizzas should not exceed the total time available.**

> **Case 1:** So if a chef with rank r can make mid2 pizzas in mid time, they can definitely make pizzas lesser than mid2 in the same time, but can they make more? Since we need to maximize the number of pizzas, we will update our search space as low = mid, i.e new search space is mid to high.
>
> **Case 2:** If a chef with rank r cannot make mid2 pizzas in time less than or equal mid, then they will definitely not be able to make more than mid2 pizzas in the same time. But can they make less than mid2 pizzas in this time? To find this out, we reduce our search space as high = mid, i.e new search space is low to mid.

Now we will need to see if the sum of this maximum number for every chef is greater than or equal to p. If yes, then this is a possible solution and our check() function returns true to our first binary search function.

Note that your first binary search function is for minimizing time, while your second binary search function is for maximizing the number of pizzas.