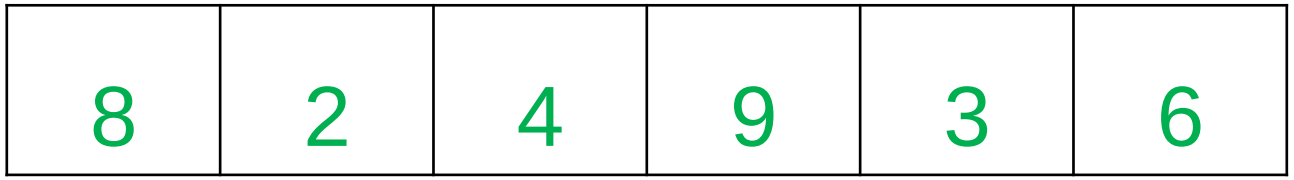
Sorting

Part 1

- You are giving a list `lst` of `n` numbers. If you are given an index `i` for `0 < i < n`.
- We find out which of the two numbers `lst[i-1]` and `lst[i]` is bigger.
- And, we will swap the bigger one to the right, such that `lst[i] > lst[i-1]`. If they are equal, just let it be.

BubbleSort

Example:



BubbleSort

Example:

8	2	4	9	3	6
2	8	4	9	3	6

BubbleSort

Example:

8	2	4	9	3	6
2	**8**	**4**	9	3	6
2	**4**	**8**	9	3	6

BubbleSort

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	**8**	**9**	3	6
2	4	**8**	**9**	3	6

BubbleSort

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	**9**	**3**	6
2	4	8	**3**	**9**	6

BubbleSort

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	4	8	3	**9**	**6**
2	4	8	3	**6**	**9**

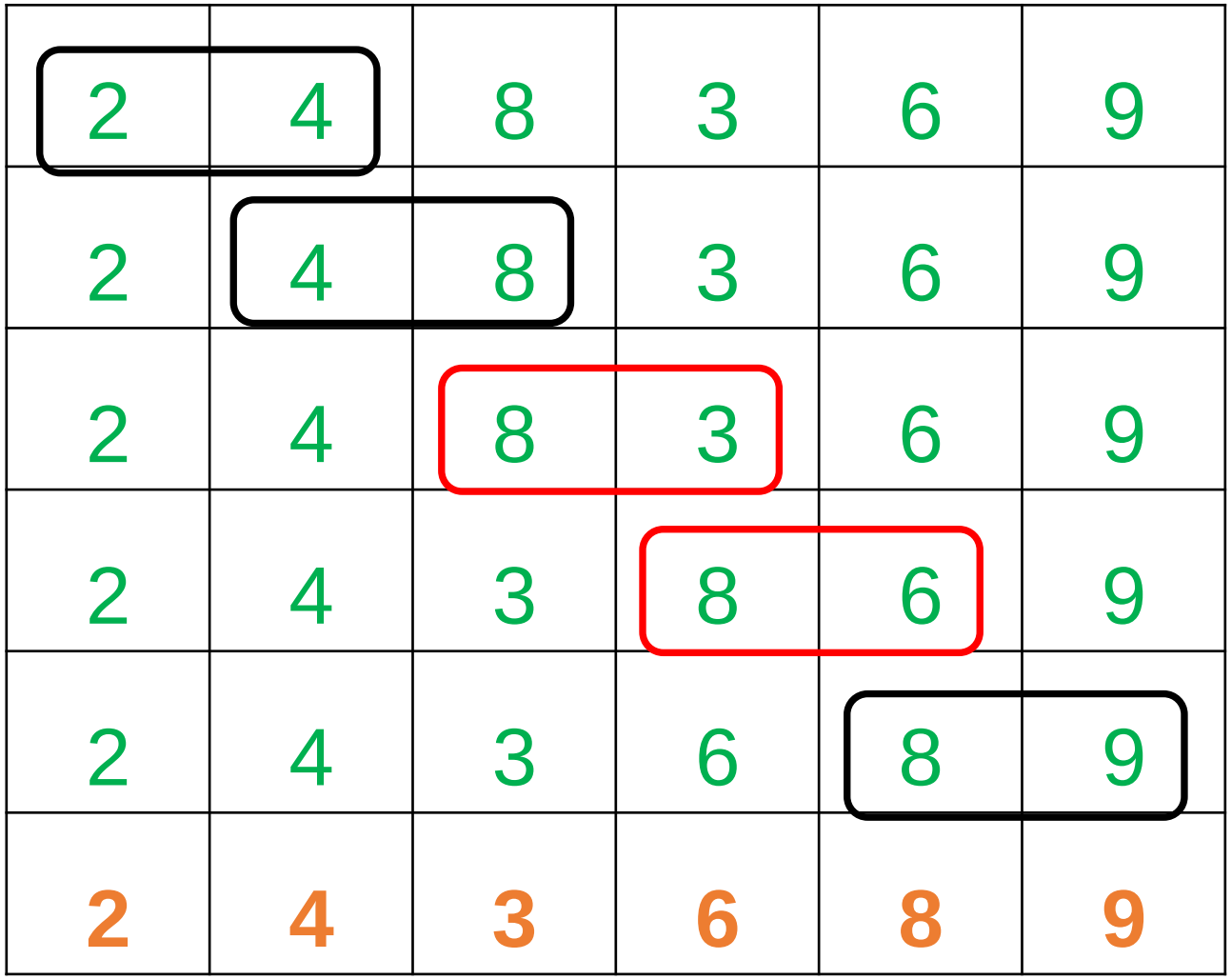
BubbleSort

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	4	8	3	9	6
2	**4**	**8**	**3**	**6**	**9**

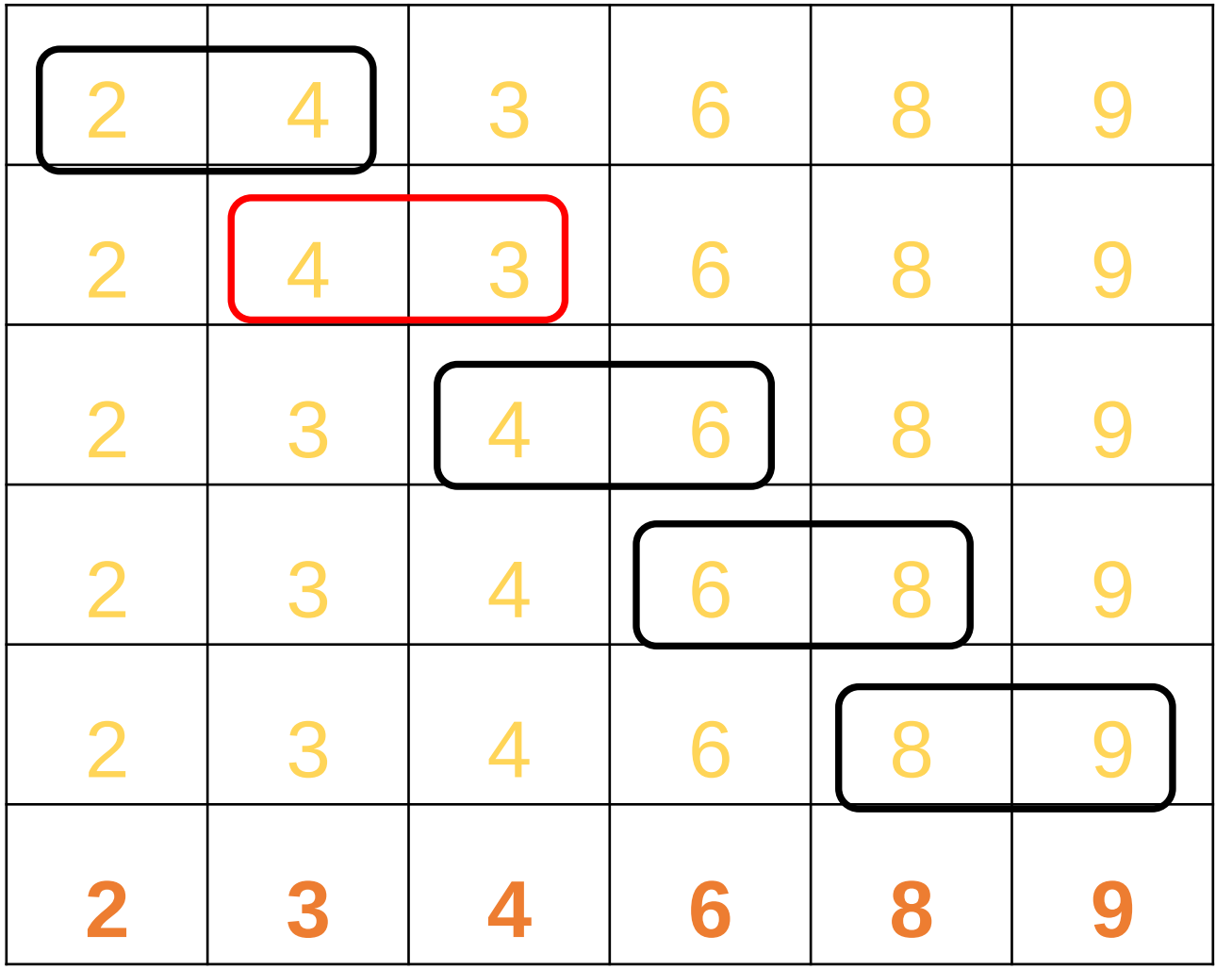
Another n-1 round

Example:



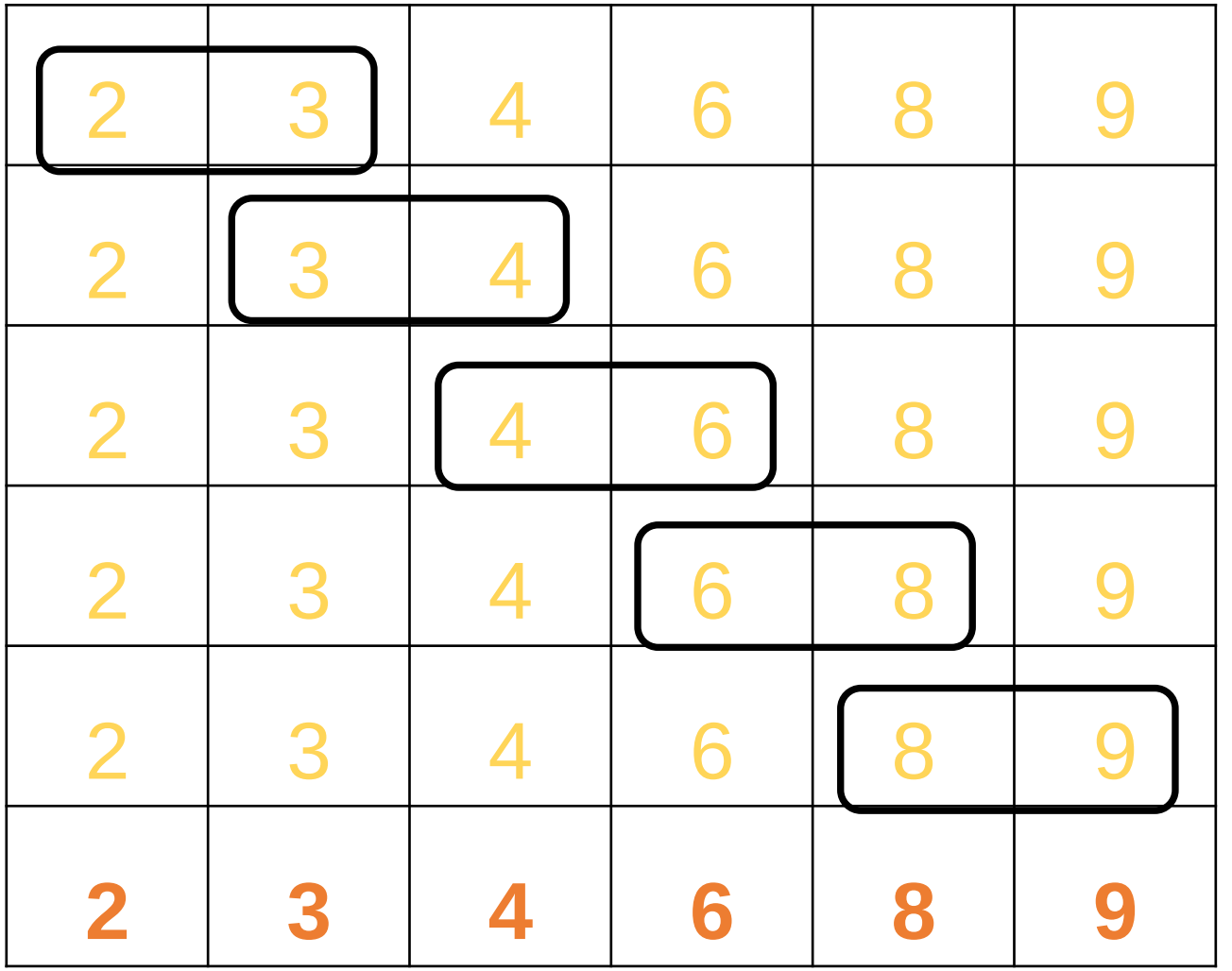
Another n-1 round

Example:



Another n-1 round

Example:



Part 1

- For one round of n-1 bubble we have

```
>>> L = [4,5,6,7,1,2,3,9,8]
>>> L1 = bubble(L)
>>> L1
[4, 5, 6, 1, 2, 3, 7, 8, 9]
```

- And a few rounds more

```
>>> L2 = bubble(L1)
>>> L2
[4, 5, 1, 2, 3, 6, 7, 8, 9]
>>> L3 = bubble(L2)
>>> L3
[4, 1, 2, 3, 5, 6, 7, 8, 9]
```

- How many rounds do we need to sort the whole list?

Bubble Sort

- Write a function `bubbleSort(lst)` to return a list that is sorted. Here is some sample output, in which, you should be able to change n to a larger number and the sorting still work.

```
>>> from random import randint
>>> n = 20
>>> L = [randint(0,10000) for i in range(n)]
>>> print(L)
[8753, 4935, 9379, 7034, 515, 854, 7747, 3661, 9932, 1590, 8123, 3924, 9565, 469
9, 6735, 1109, 9955, 1600, 2481, 9363]
>>> print(bubbleSort(L))
[515, 854, 1109, 1590, 1600, 2481, 3661, 3924, 4699, 4935, 6735, 7034, 7747, 812
3, 8753, 9363, 9379, 9565, 9932, 9955]
```

Final thoughts

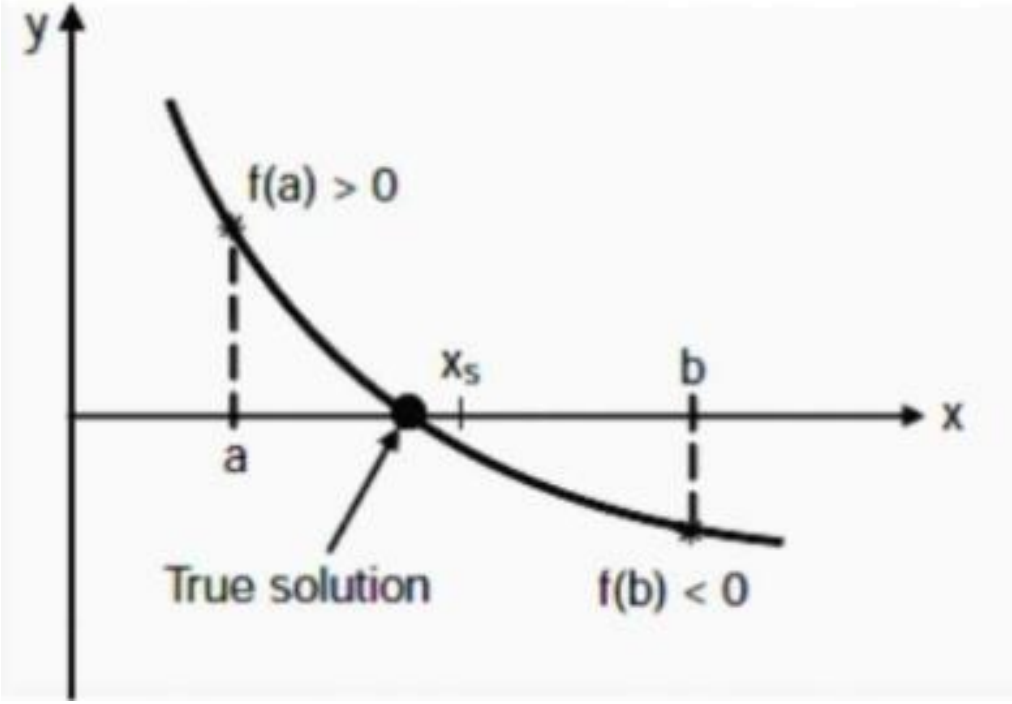
- Do you really need to:
  - Apply so many times?
    - When can we end?
  - For the whole list?

Searching

Bisection method

Bisection Method

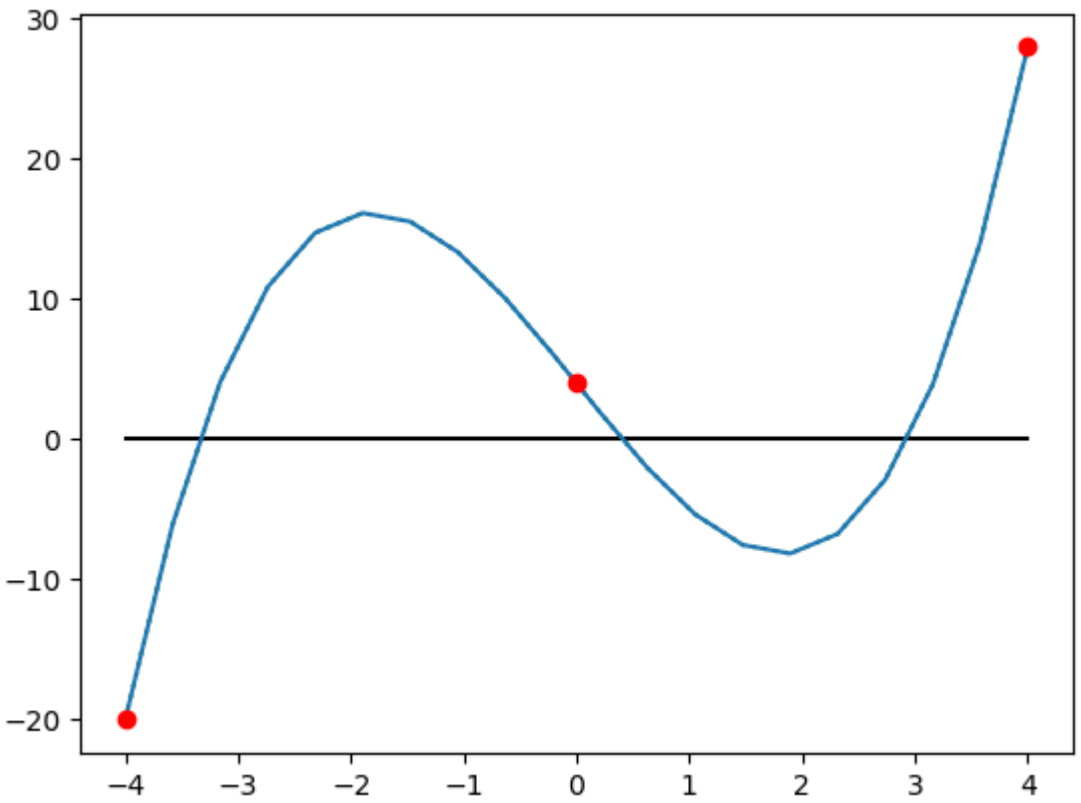
- The ***bisection method*** in mathematics is a root-finding method that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. Given a function f(x), you want to solve for x when f(x) = 0.

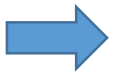


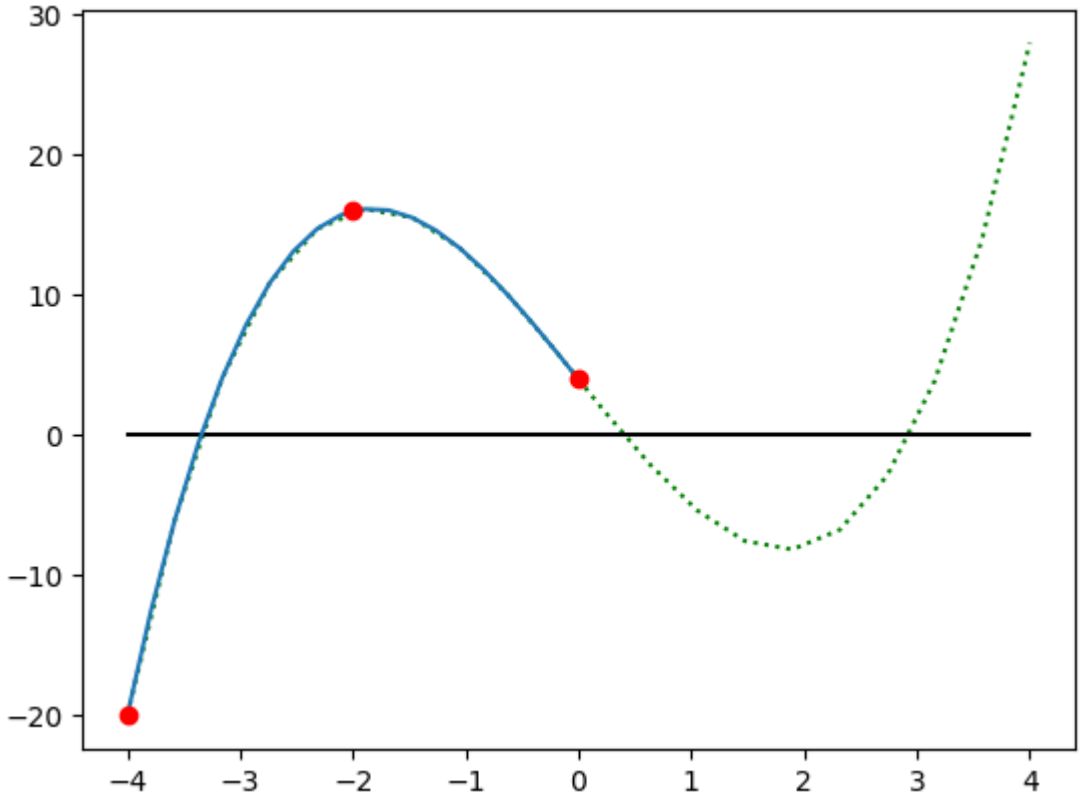
Algorithm

- In this question, we assume that the function f is continuous. And you are given two numbers a and b such that f(a) > 0 and f(b) < 0. So you repeat the following
  - Compute $x_s = (a + b) /2$
  - If $f(x_s) < 0$ then the solution lies on the of x_s. So, $b = x_s$.
    - Otherwise, $a = x_s$.
  - Repeat these until the absolute value of $f(x_s)$ is smaller than a constant ‘error’

“a” “mid” and “b”

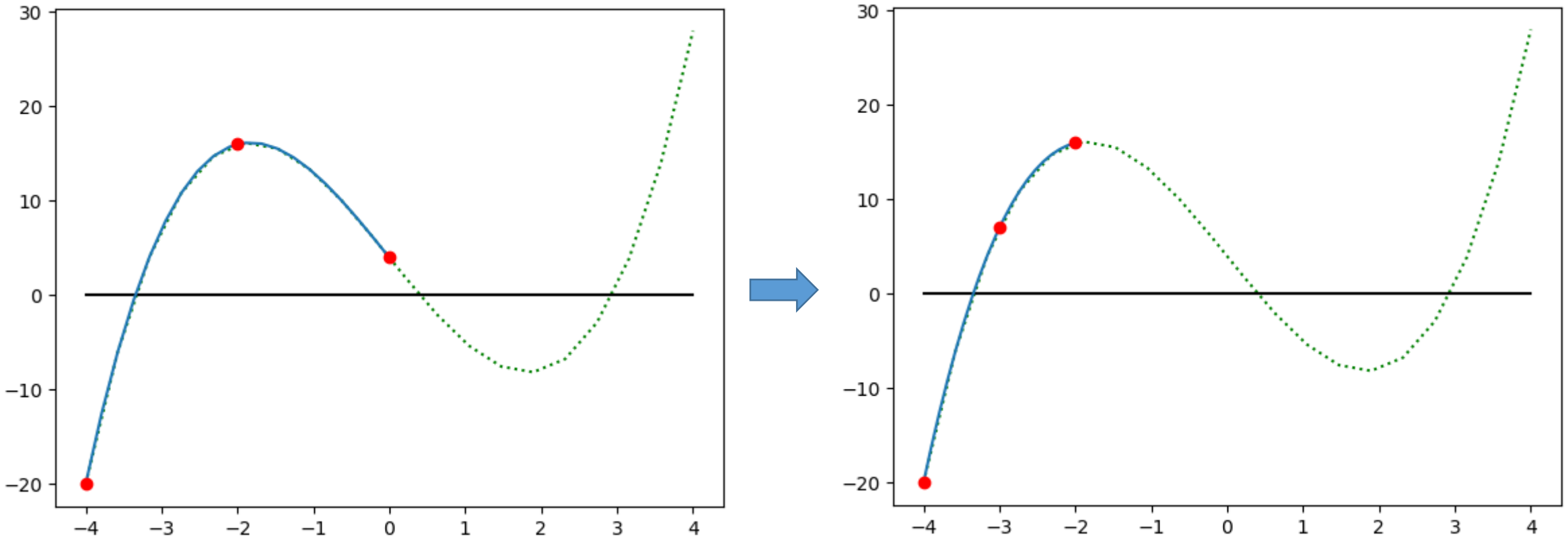






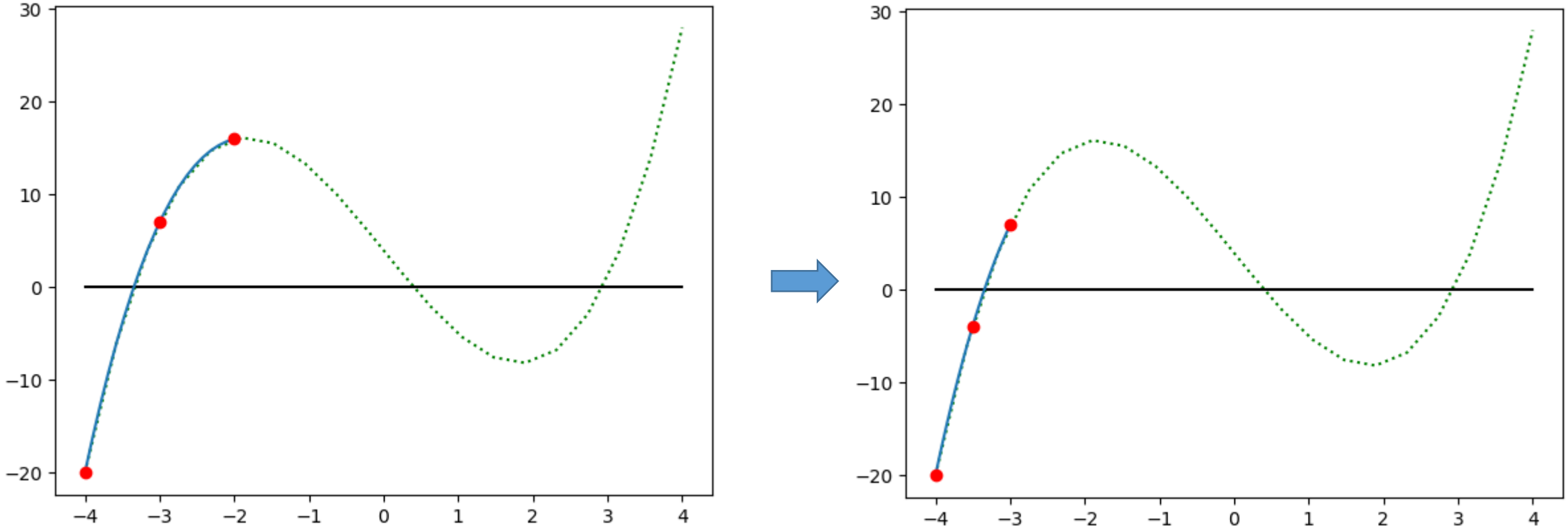
- Replace `b` by `mid`

“a” “mid” and “b”



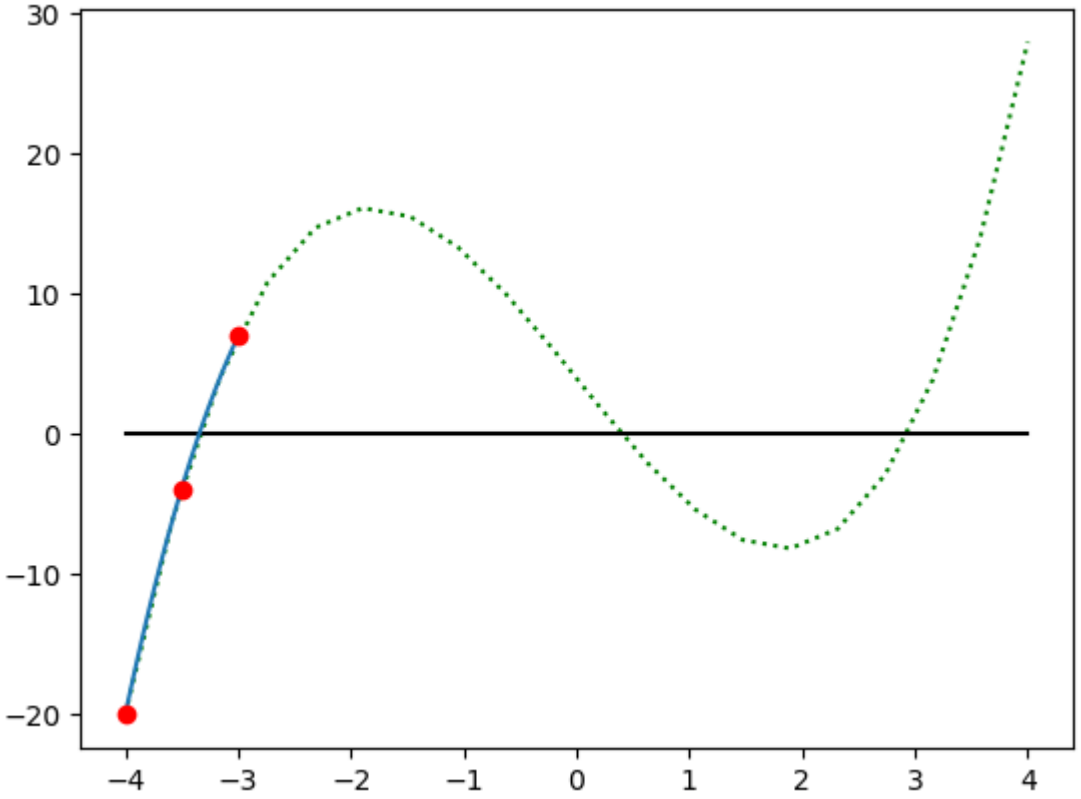
- Replace `b` by `mid`

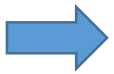
“a” “mid” and “b”

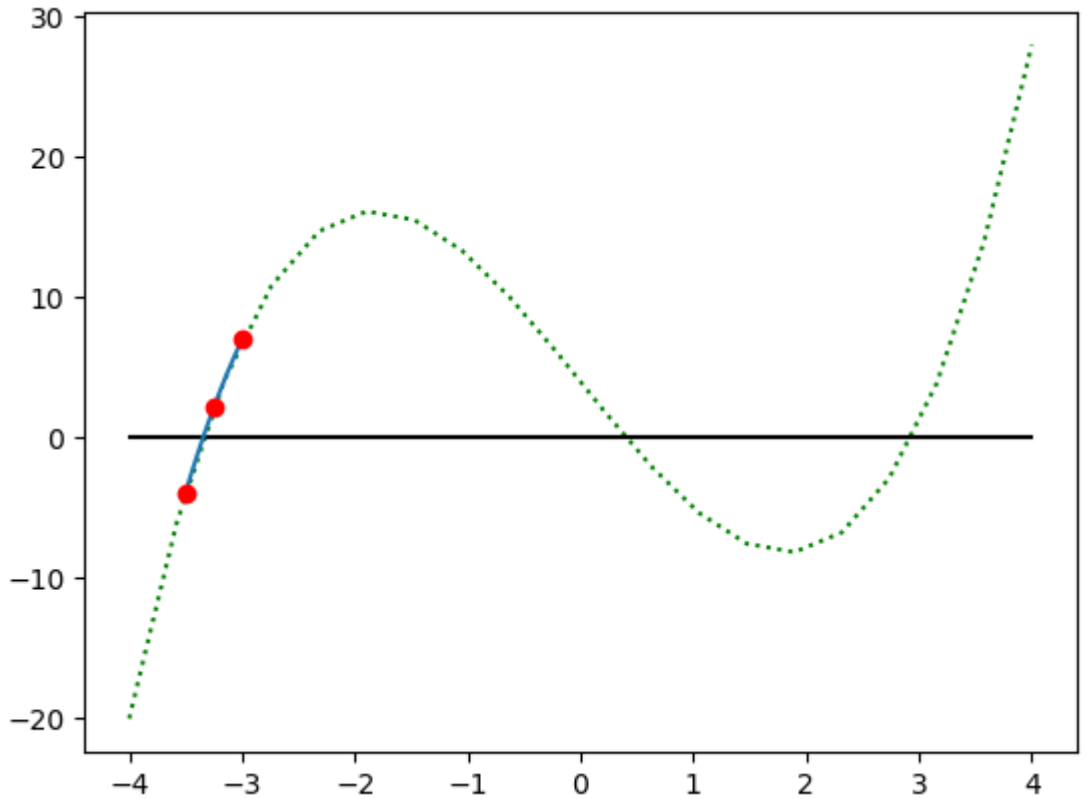


- Replace `b` by `mid`

“a” “mid” and “b”

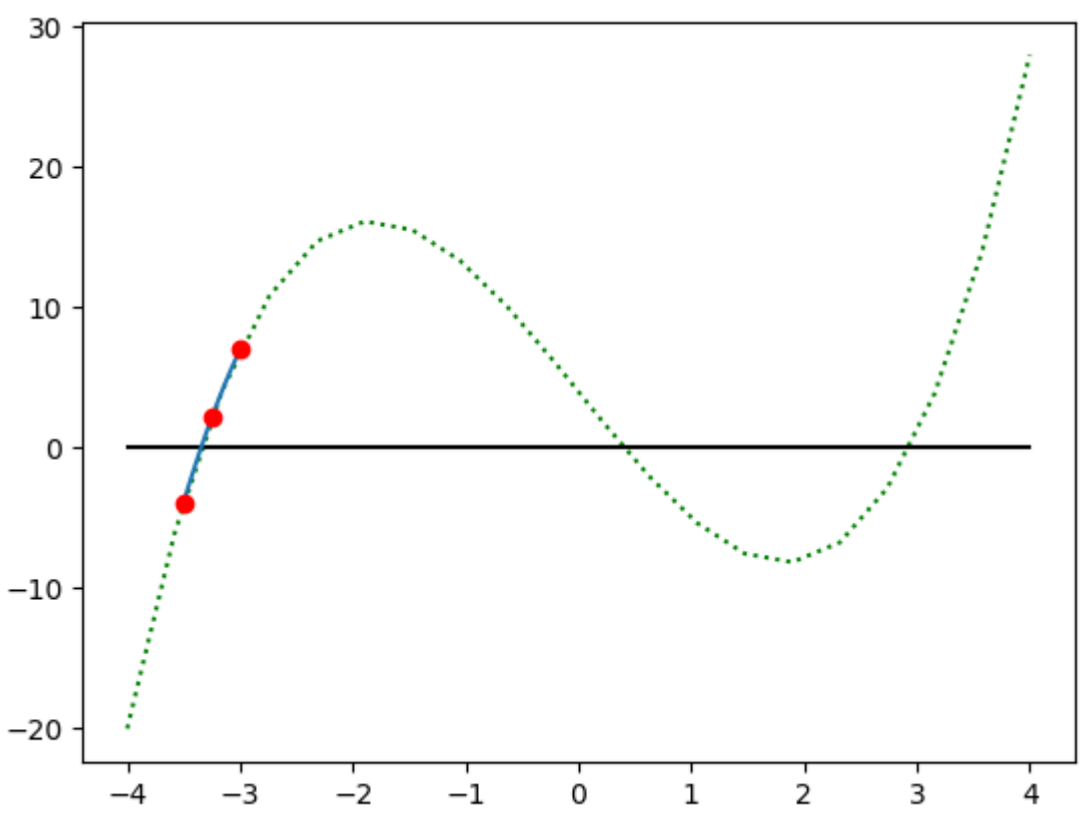


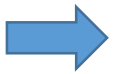


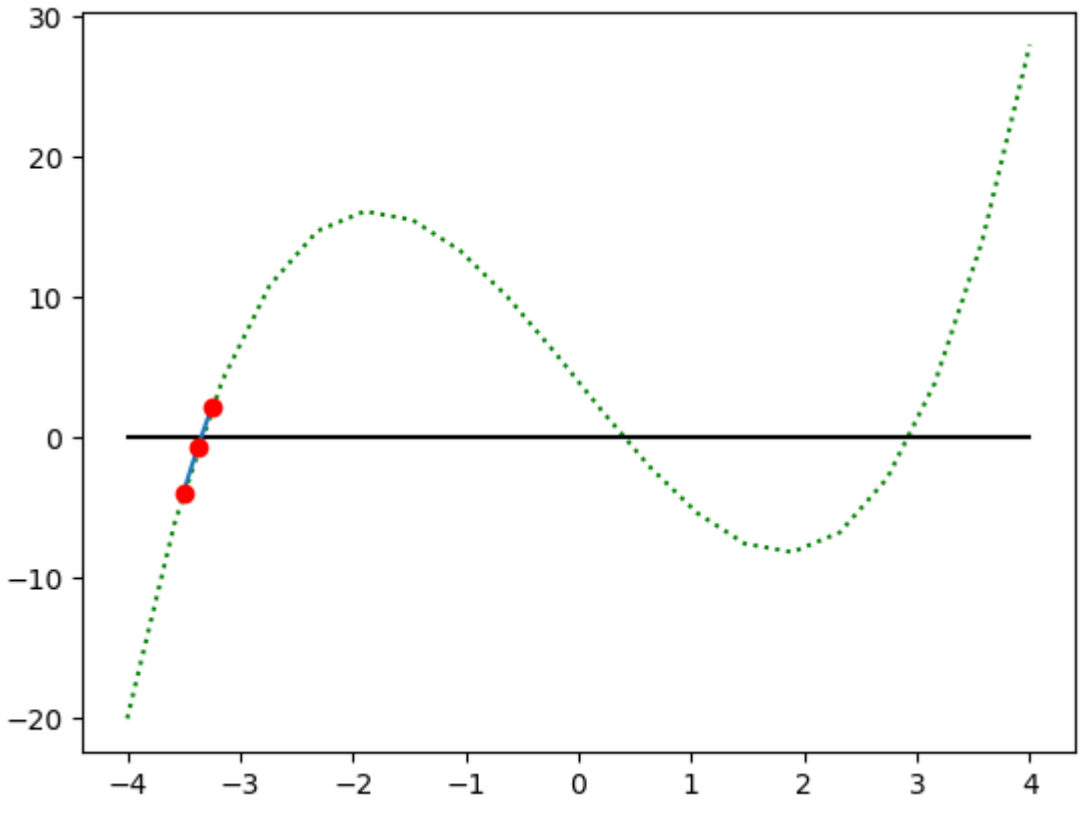


- Replace `a` by `mid`

“a” “mid” and “b”

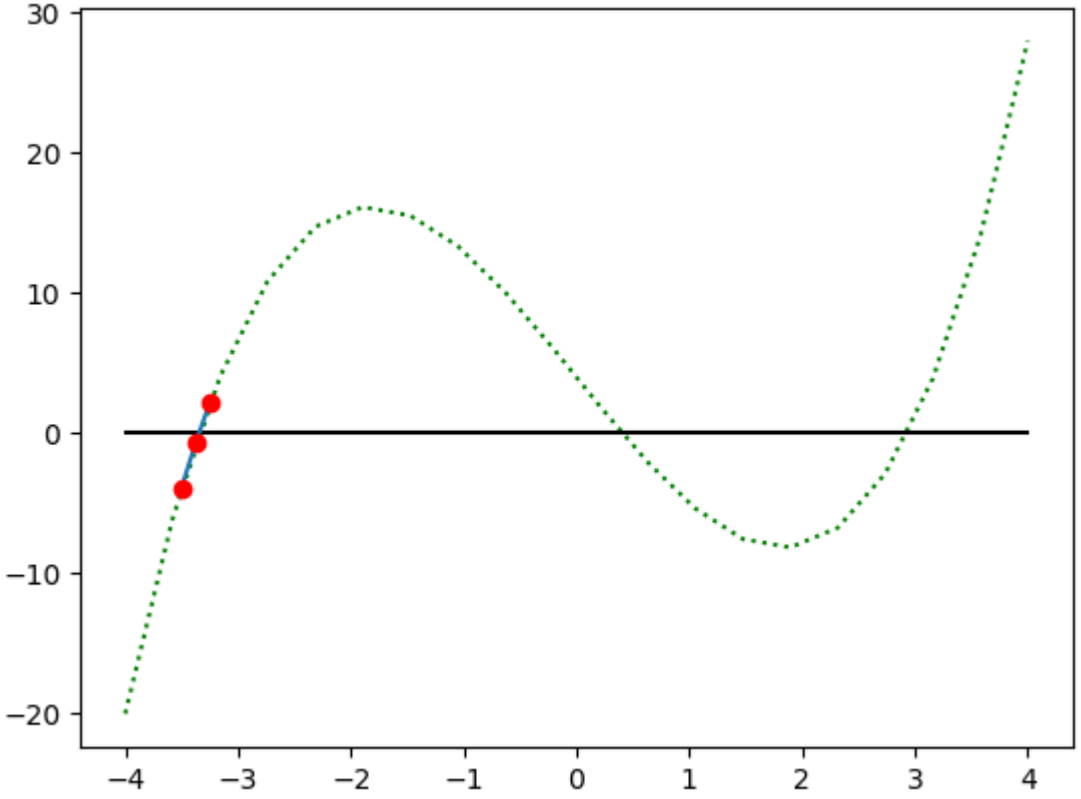


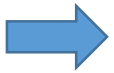


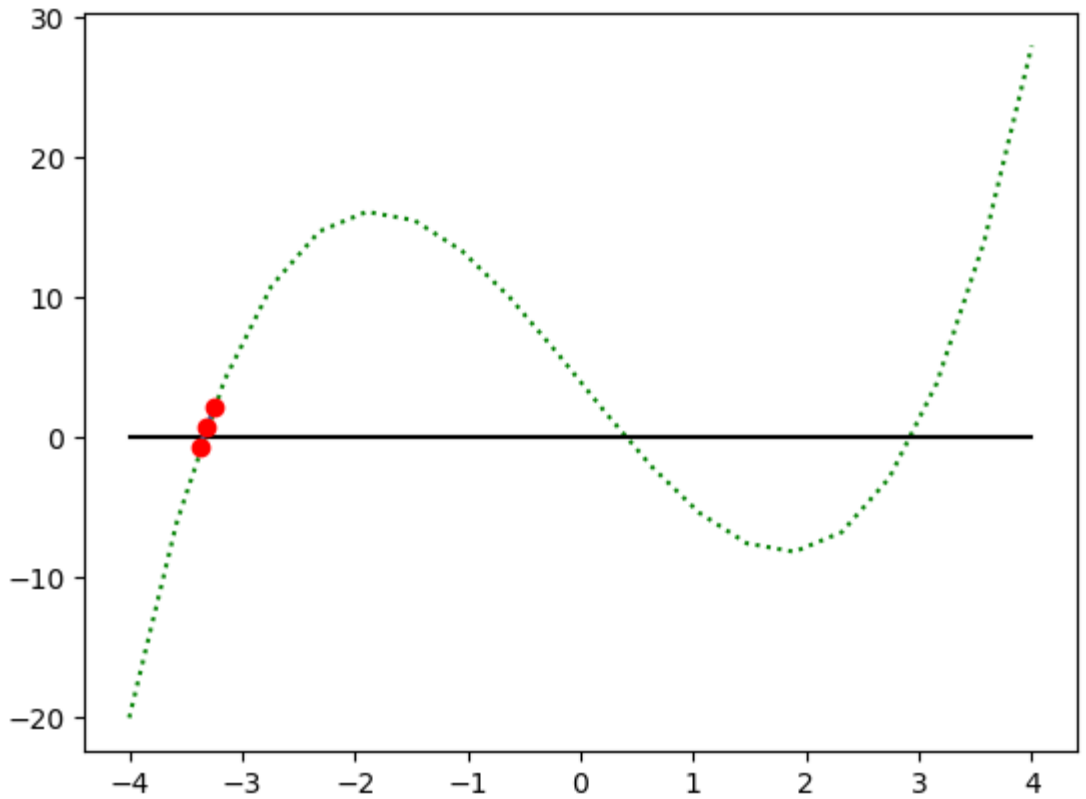


- Replace `b` by `mid`

“a” “mid” and “b”







- After more iterations, x = -3.34596…

Rest of the Tutorial

- Try to implement `bisection(start,end,f)` to solve the problem
- For the two given function `f` and `g`, you should be able to solve `f` without “reversing”
- Suggestion
  - For every iteration, print out:
    - a, mid , b
    - f(a), f(mid), f(b)
    - And the decision that which one among a or b that is to be replaced by mid

```
def bisection(start,end,f):
    #assuming f(start) < 0 and f(end) > 0
    if f(start) * f(end) > 0:
        print("The curve does not have different signs on both ends")
        return
    reverse = False
    if f(start) > 0:
        reverse = True

    a = start
    b = end
    mid = (start+end)/2
    while abs(f(mid)) > ERROR:
        # This part of code is only for visualization only
        # This part of code is only for debugging only
        if DEBUG:
            print( ' start = ' + str(a), ' mid = ' + str(mid) + ' end = ' + str(b))
            print( ' f(start) =' + str(f(a)) + ' f(mid) = ' + str(f(mid)) + ' f(end) = ' + str(f(b)))

        if not reverse:
            if f(mid) < 0:
                a = mid
            else:
                b = mid
        else:
            # How?
            pass
        mid = (a+b)/2
    return mid
```

Extra qSort

- You are given a list `lst` of n numbers. To simplify the problem, we assume we have no duplicate element in the list. (However, even so, it is not difficult to solve.)
- We will pick any element of the list, say `x`.
- And for the rest of the element in `lst`, we will separate them into two lists, one list `lsta` contains all the element smaller than `x`, and `lstb`, otherwise. Let's give a name to this functionality as "partition".

```
>>> lst = [5,4,1,2,3,9,7,6,0]
>>> partition(lst,4)
([1, 2, 3, 0], [5, 9, 7, 6])
```

Magic

- Then we apply some "*magic*" to `lsta` and `lstb` such that they are sorted after the *magic*.
- Finally, we output the list `lsta + [x] + lstb`.

```
>>> lst = [5,4,1,2,3,9,7,6,0]
>>> part = partition(lst,4)
>>> lsta = magic(part[0])
>>> lsta
[0, 1, 2, 3]
>>> lstb = magic(part[1])
>>> lstb
[5, 6, 7, 9]
>>> lsta + [4] + lstb
[0, 1, 2, 3, 4, 5, 6, 7, 9]
```

Magic

- Then we apply some "*magic*" to `lsta` and `lstb` such that they are sorted after the *magic*.
- Finally, we output the list `lsta + [x] + lstb`.

```
def magic(lst):
    if not lst:
        return lst
    part = partition(lst,lst[0])
    lsta = magic(part[0])
    lstb = magic(part[1])
    return lsta + [lst[0]] + lstb
```

- You can finish the function partition in order to finish this *magic*