Week 7b

Searching and Sorting

Sorting

Problem

- Given a list `lst` of `n` numbers and an index `i` where `0 < i < n`, we can compare the two numbers `lst[i-1]` and `lst[i]`.
  - If the two numbers are different, the bigger one should be swapped to the right such that `lst[i] > lst[i-1]` after the swap.
- Write a function `bubble(lst)` which uses a simple loop to perform the above task from `i = 1` to `i = n-1`.
  - `n-1` iterations: maximum `n-1` swaps

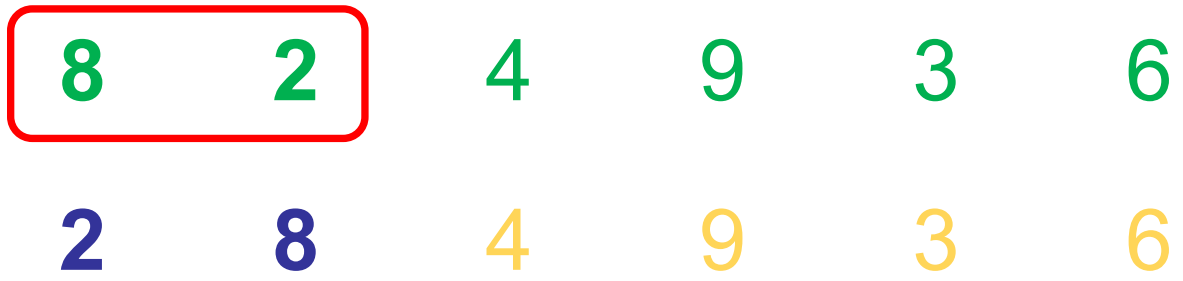
Original `lst`

Example:

8 2 4 9 3 6

Round 1, `i = 1`

Example:

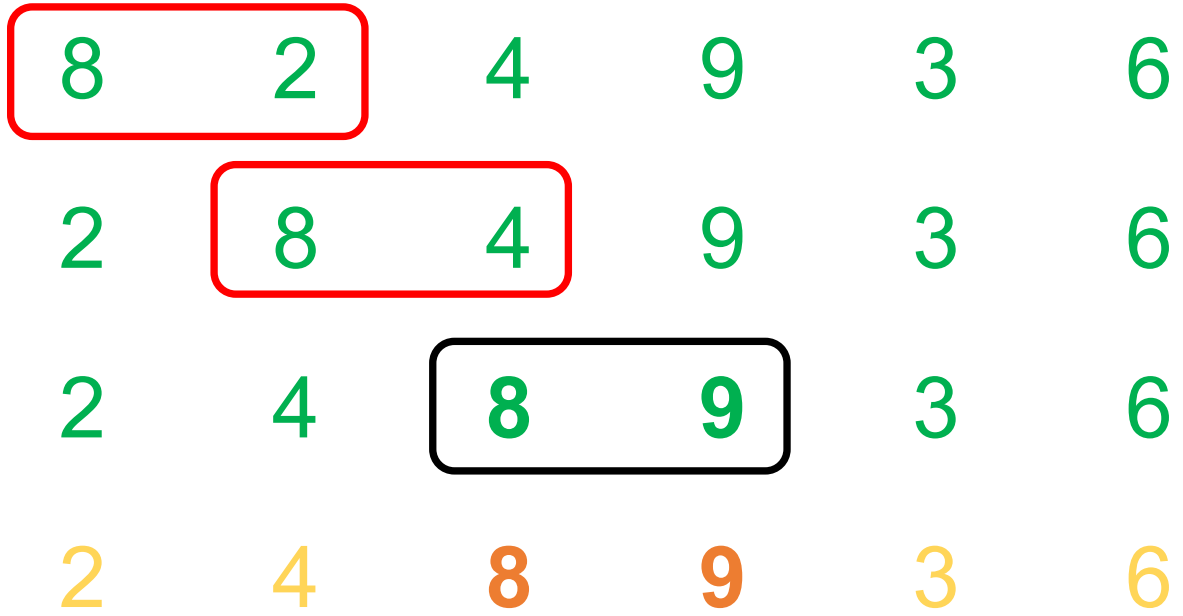


Round 1, `i = 2`

Example:

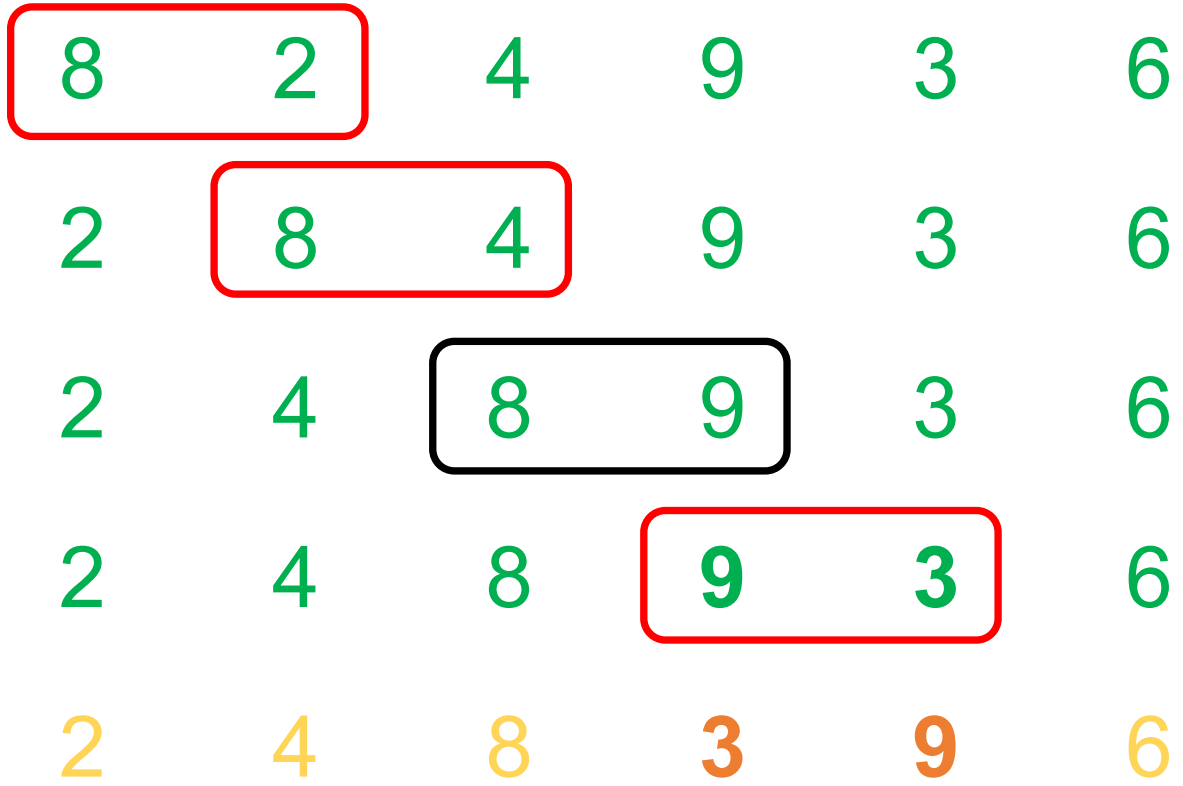
Round 1, `i = 3`

Example:



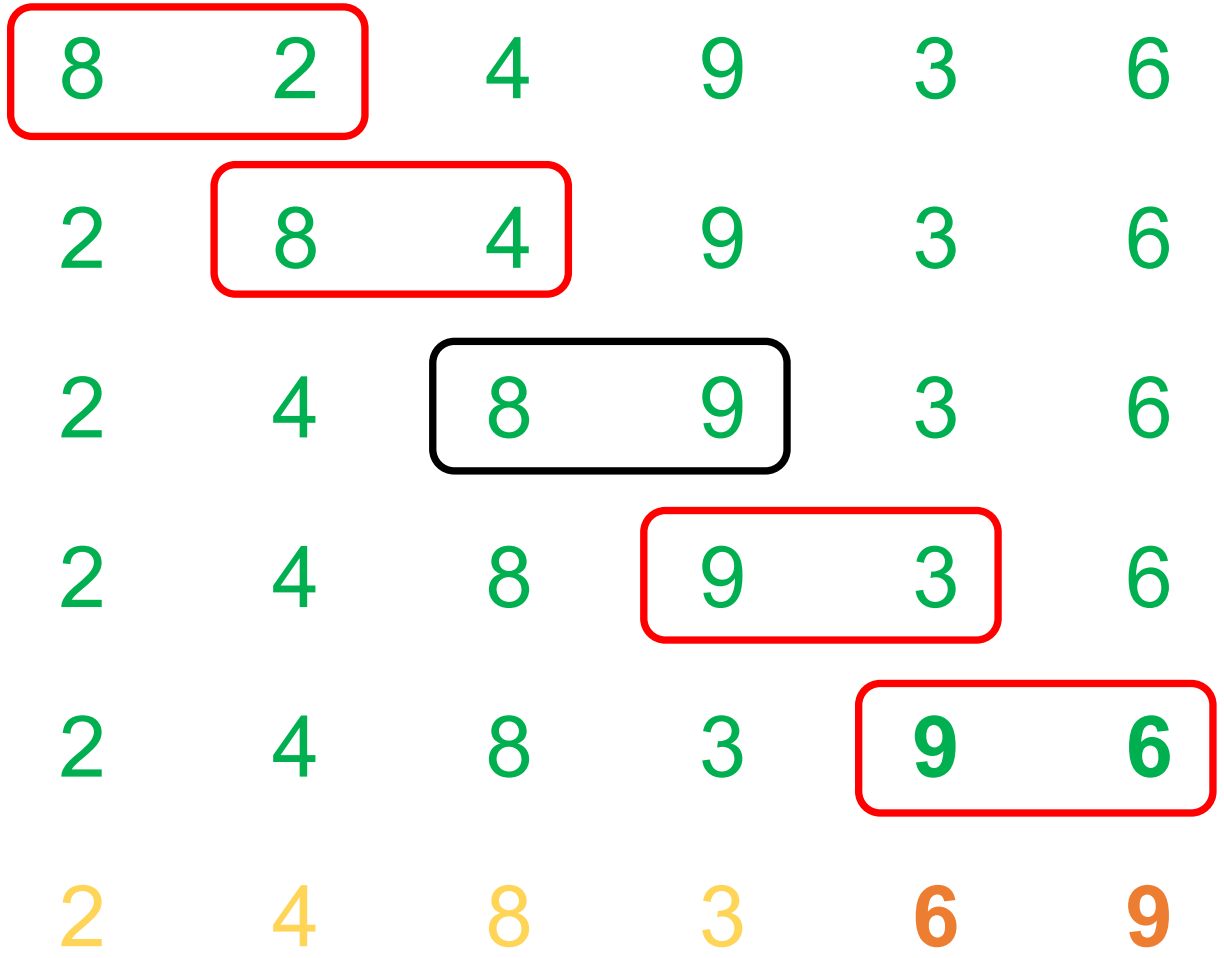
Round 1, `i = 4`

Example:



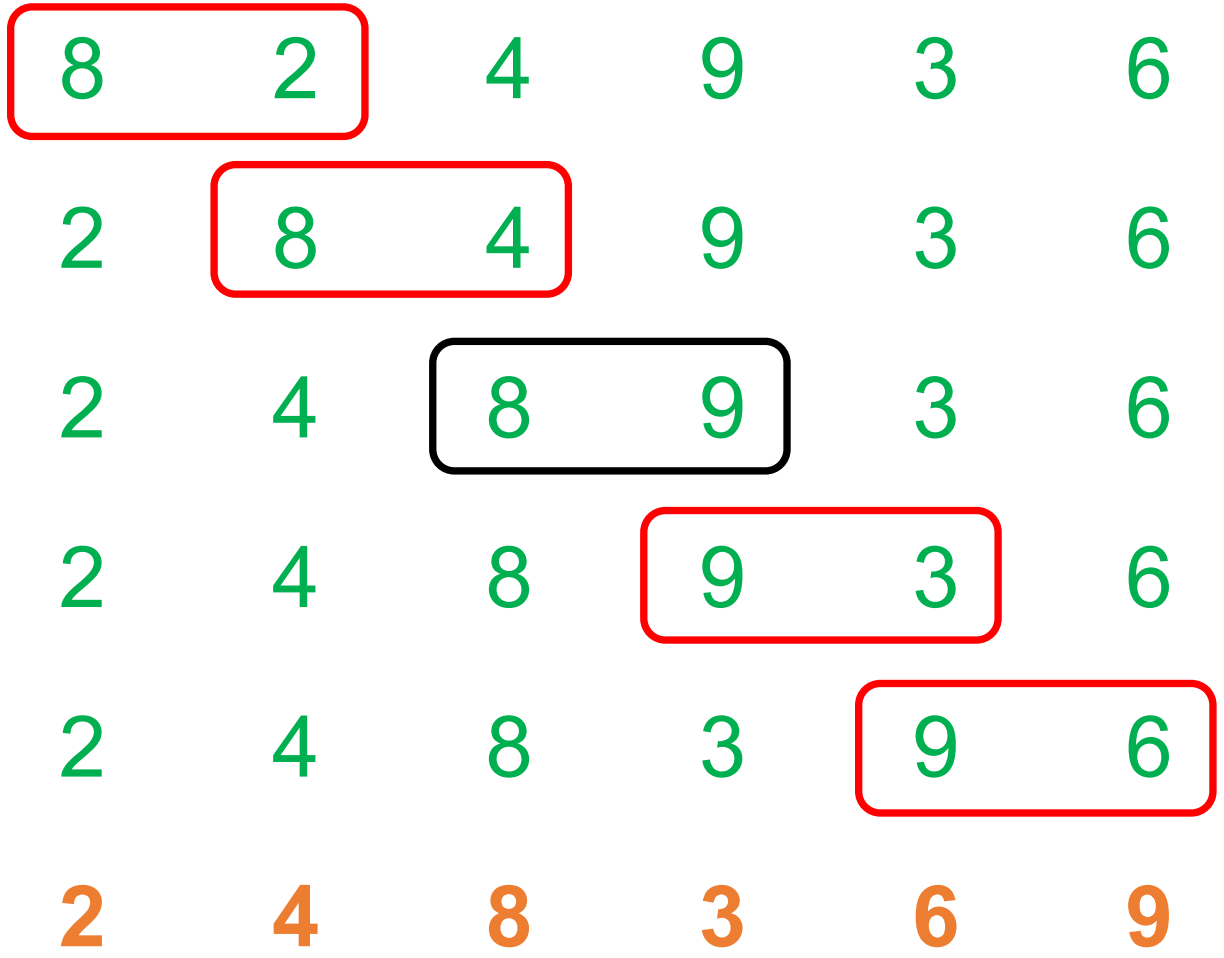
Round 1, `i = 5`

Example:



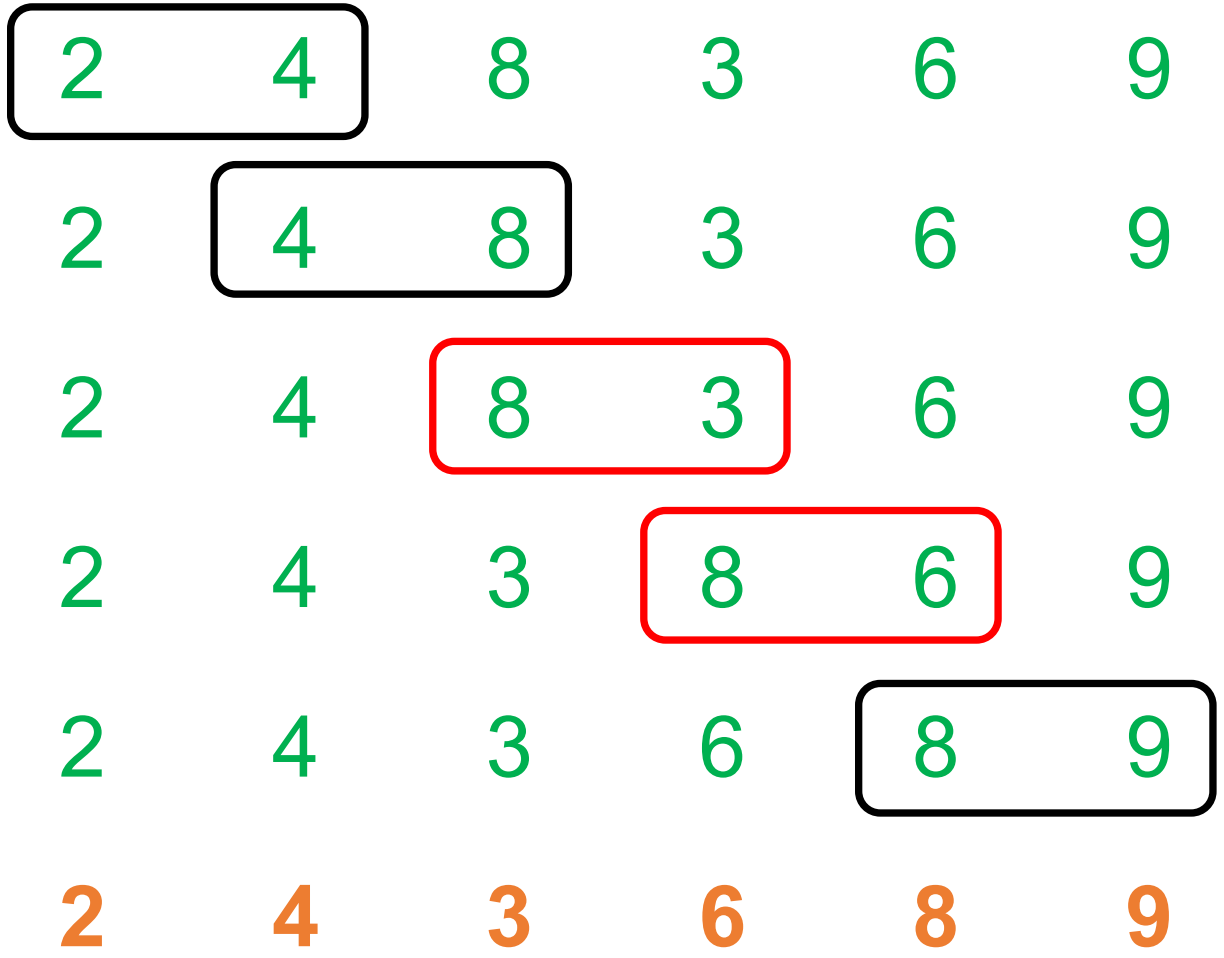
Round 1

Example:



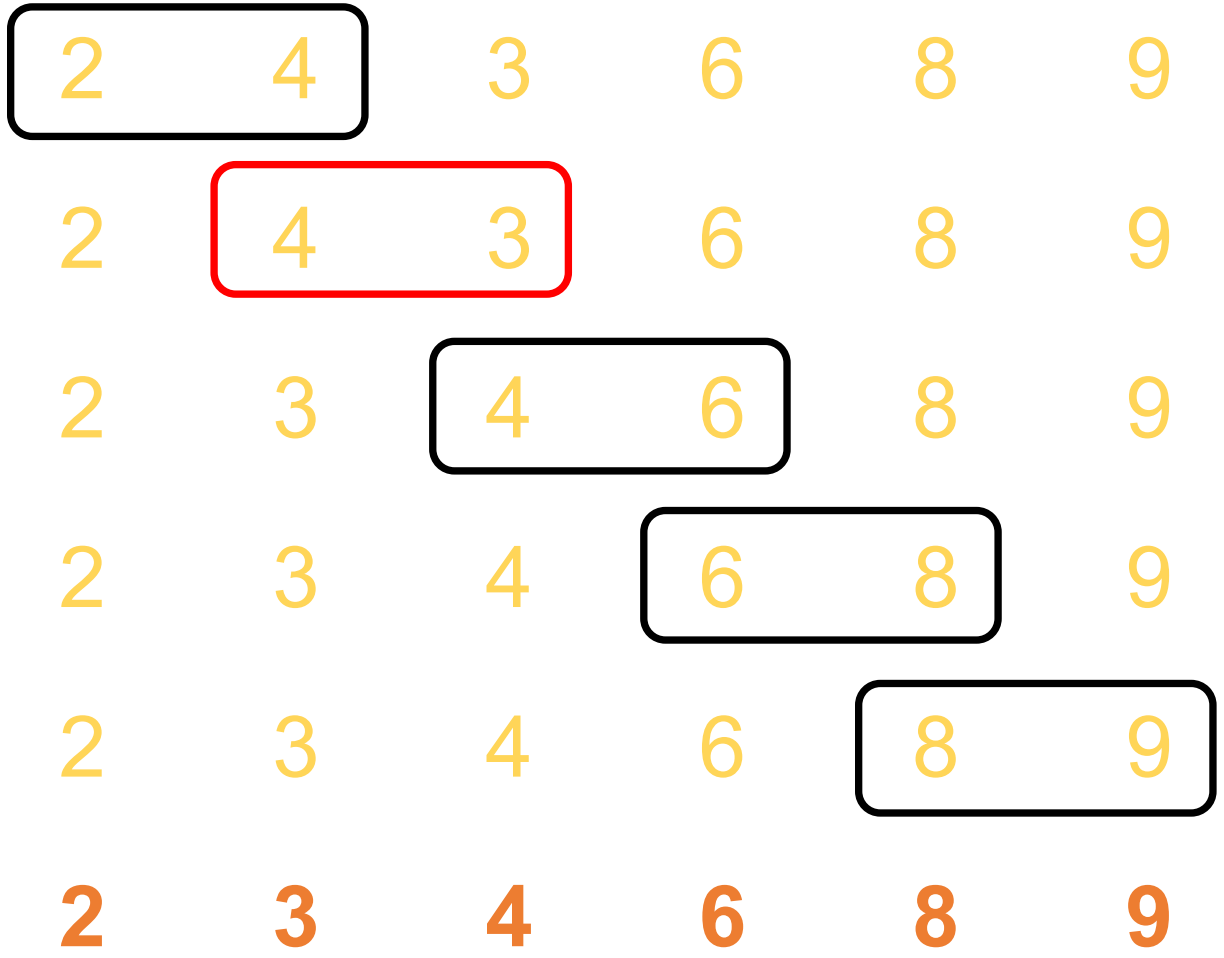
Round 2

Example:



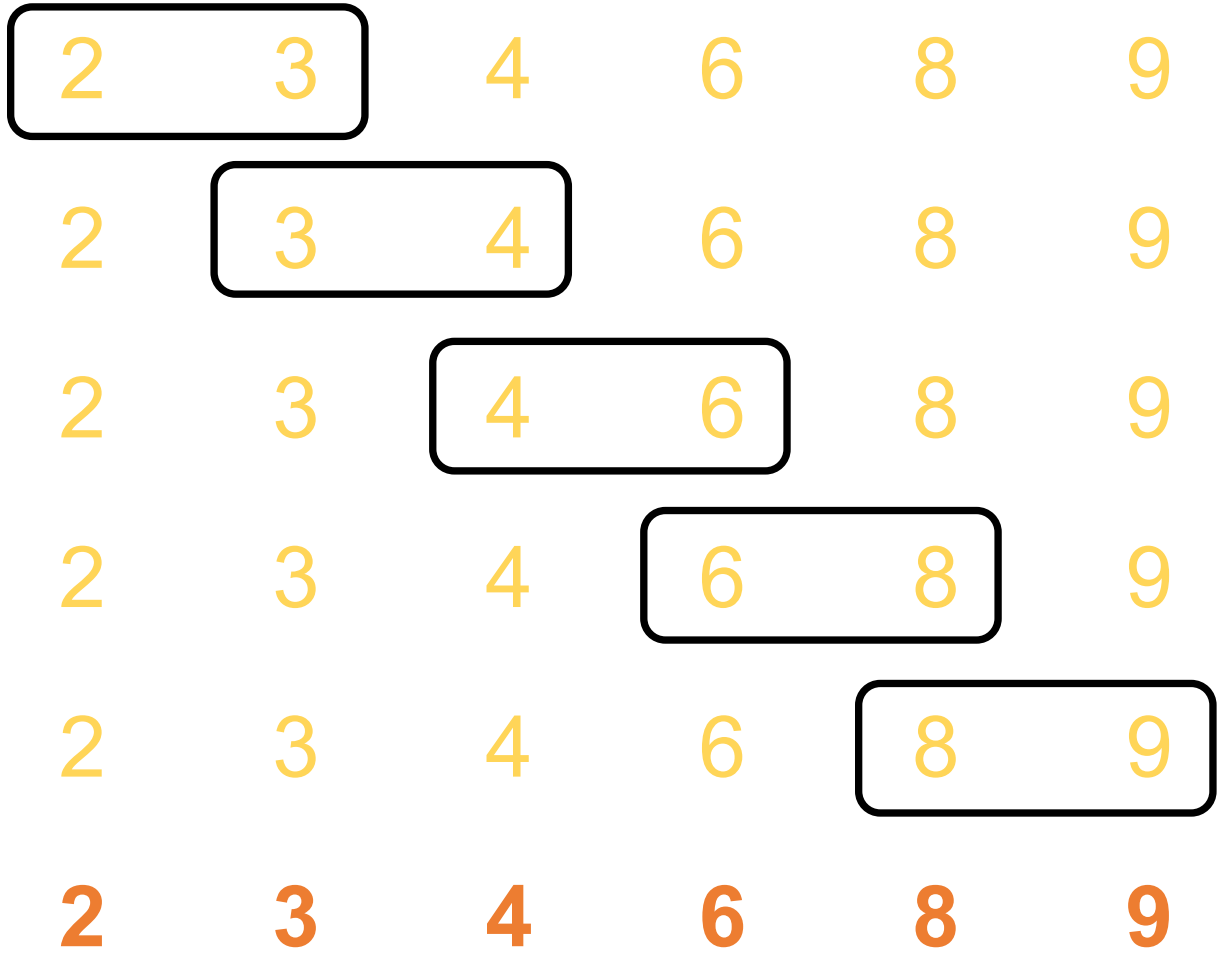
Round 3

Example:



Round 4

Example:



Bubble

```
# sorts lst itself
def bubble(lst):
  n = len(lst)
  for i in range(1, n):
     if lst[i-1] > lst[i]:
         lst[i-1], lst[i] = lst[i], lst[i-1]
```

Bubble

```
# returns a "bubbled" version of lst
def bubble(lst):
  n = len(lst)
  lst1 = lst.copy()
  for i in range(1, n):
     if lst1[i-1] > lst1[i]:
         lst1[i-1], lst1[i] = lst1[i], lst1[i-1]
  return lst1
```

Bubble

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

- After one round of bubble we have:

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

- And after 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]
```

Bubble

1. What does one round of bubble do to the list?
   - Places the largest element in its final (correct) position
2. What happens to the list as it goes through more rounds of bubble?
   - Places the 2nd largest element in its final position, then the 3rd, 4th, …
   - The `i`th pass places the `i`th largest element in its final position
3. How many rounds of bubble do we need to sort the whole list?
   - `n` rounds?

Bubble Sort

- Write a function `bubbleSort(lst)` which returns a list that is sorted from the elements of `lst`.

```
>>> 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]
```

- You should be able to change `n` to a larger number
  - And the sorting will still work

Bubble Sort

```
# returns a sorted version of lst
def bubbleSort(lst):
  n = len(lst)
  for _ in range(n):
     lst = bubble(lst)
  return lst
```

Final Thoughts

- Do you really need to…
  - …apply `bubble()` for `n` times…
    - How about `n-1` times?
    - Or even fewer?

Bubble Sort (Optimized)

```
# returns a sorted version of lst
def bubbleSortOptimized(lst):
  n = len(lst)
  lst1 = lst.copy()
  for _ in range(n - 1):
     swapped = False
     for i in range(1, n):
        if lst1[i-1] > lst1[i]:
           lst1[i-1], lst1[i] = lst1[i], lst1[i-1]
           swapped = True
     if not swapped: break
  return lst1
```

Final Thoughts

- Do you really need to…
  - …apply `bubble()` for `n` times…
    - How about `n-1` times?
    - Or even fewer?
  - …and to the entire list?

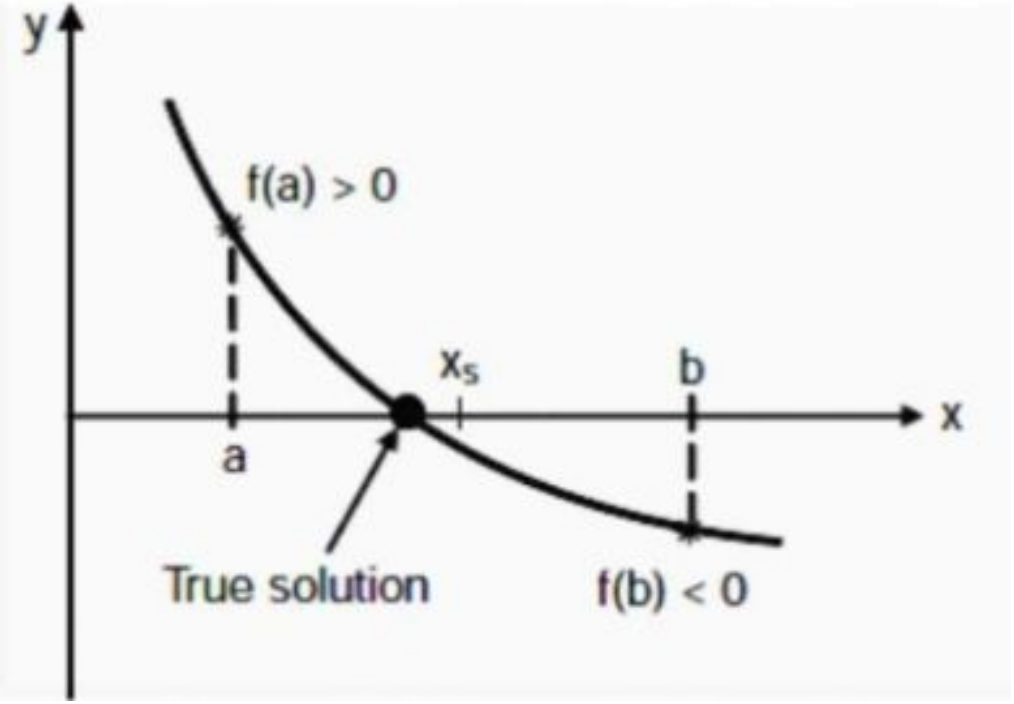
Bubble Sort (Optimized)

```
# returns a sorted version of lst
def bubbleSortOptimized(lst):
  n = len(lst)
  lst1 = lst.copy()
  for i in range(n - 1, 0, -1):
     swapped = False
     for j in range(1, i):
        if lst1[j-1] > lst1[j]:
           lst1[j-1], lst1[j] = lst1[j], lst1[j-1]
           swapped = True
     if not swapped: break
  return lst1
```

Searching

Bisection Method

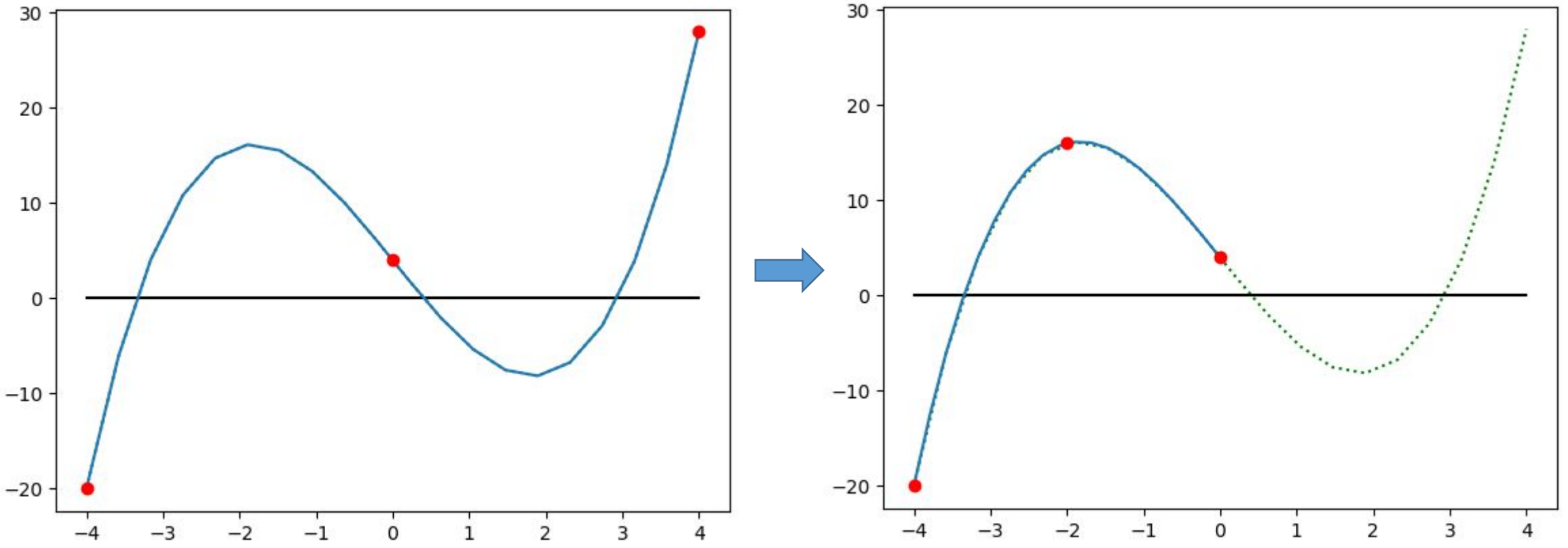
- The ***bisection method*** in mathematics is a root-finding method that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. Given a function $f(x)$, you want to solve for x when $f(x) = 0$.



Bisection Method

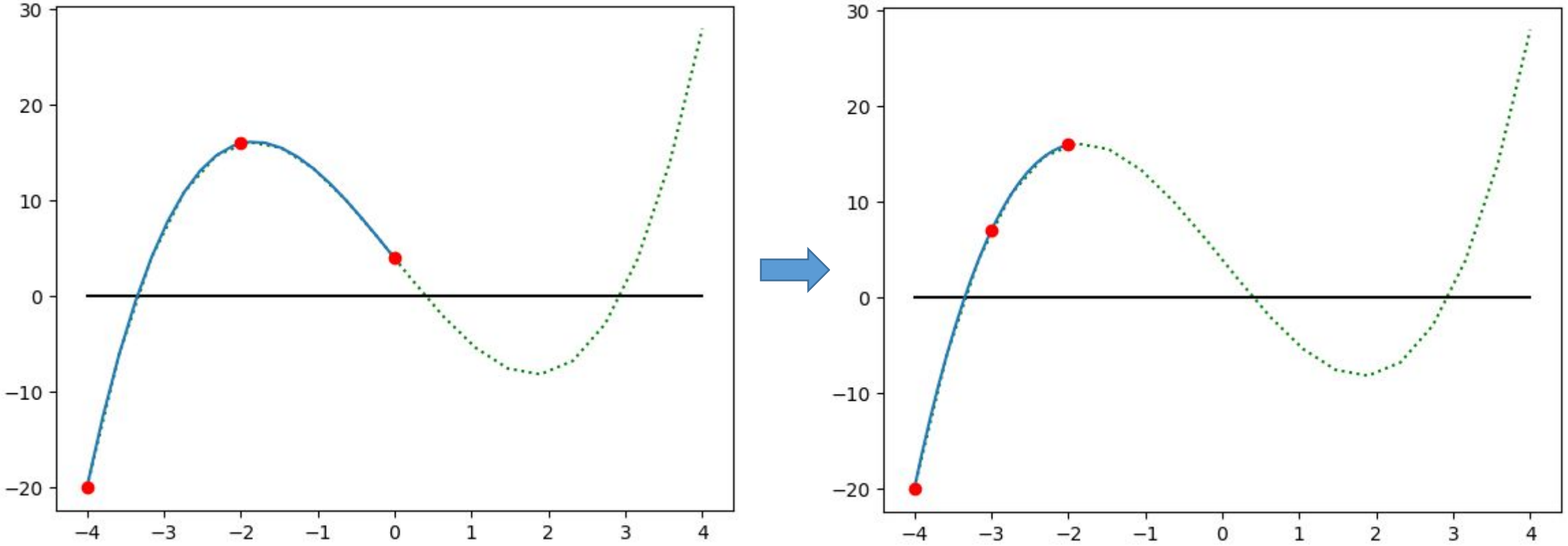
- In this question, we assume that the function f is continuous.
- Given numbers a and b such that $f(a) > 0$ and $f(b) < 0$, repeat:
  - Compute $x_s = (a + b)/2$.
  - If $f(x_s) < 0$, then the solution lies on the left of x_s. So, $b = x_s$.
    - Otherwise, $a = x_s$.
- We stop when $|f(x_s)|$ is smaller than a constant "error".
- If $f(a) > 0$ and $f(b) < 0$, we just need to reverse the inequality (i.e. check for $f(x_s) > 0$ instead)
- What if $f(a)$ and $f(b)$ have the same sign?

`a`, `mid` and `b`



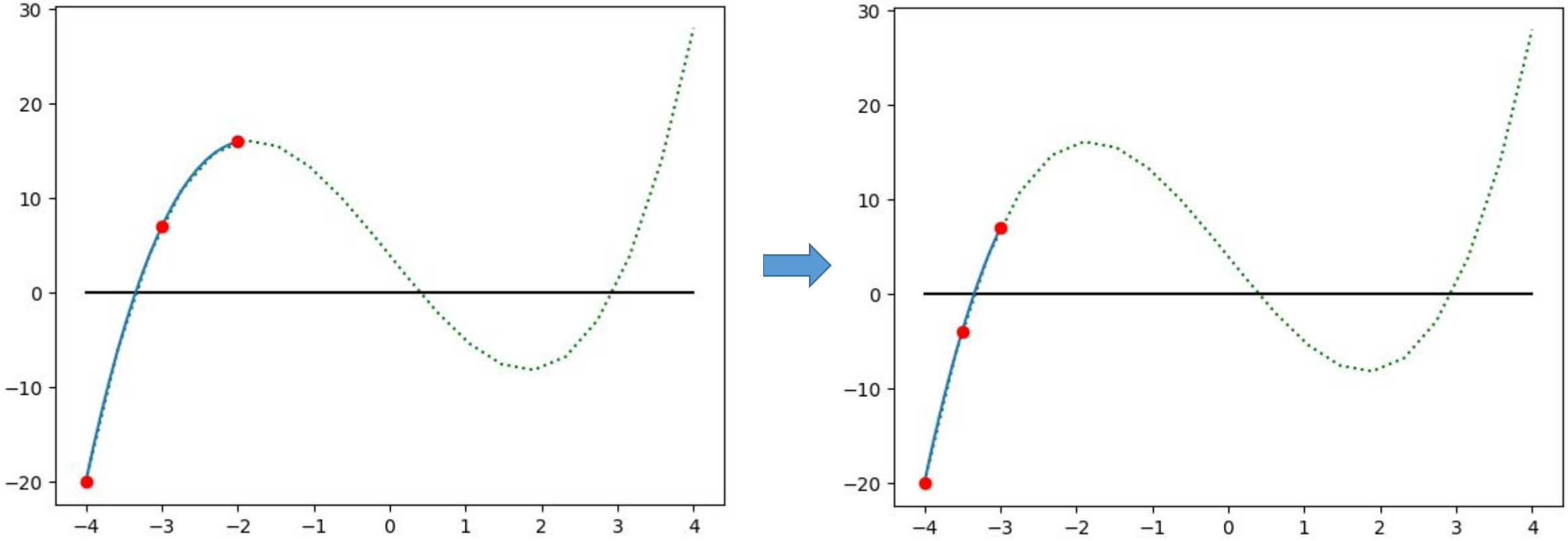
- Replace `b` by `mid`

`a`, `mid` **and** `b`



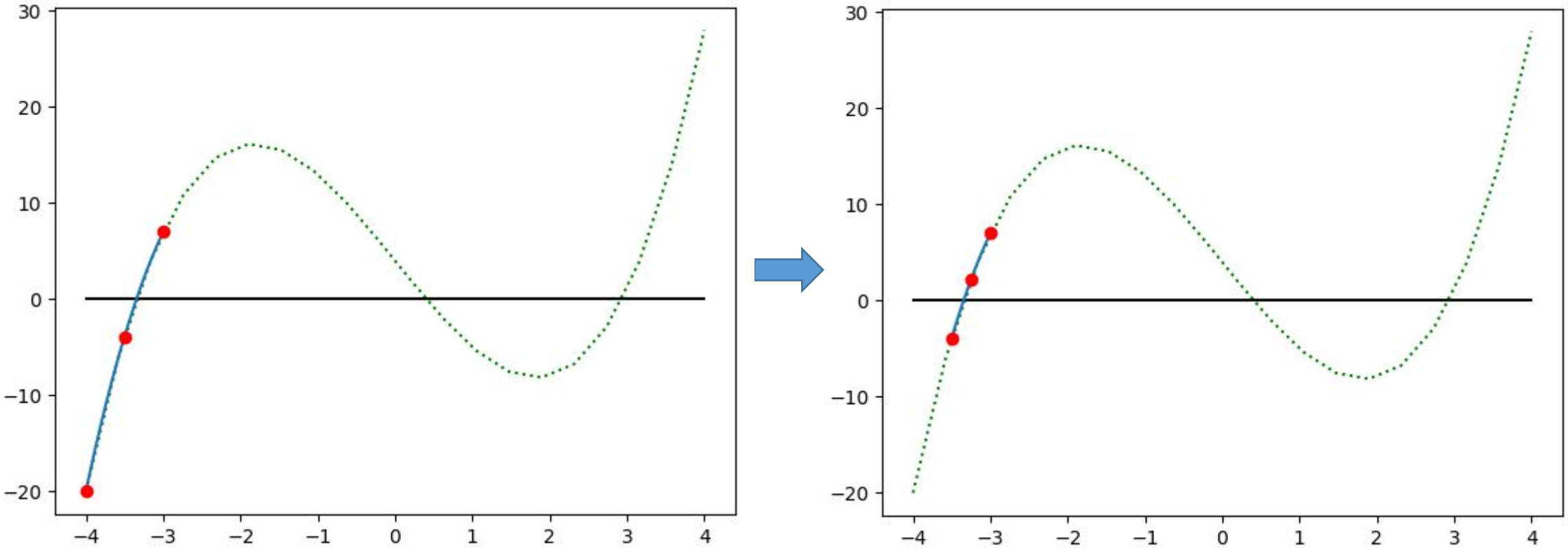
- **Replace** `b` **by** `mid`

`a`, `mid` and `b`



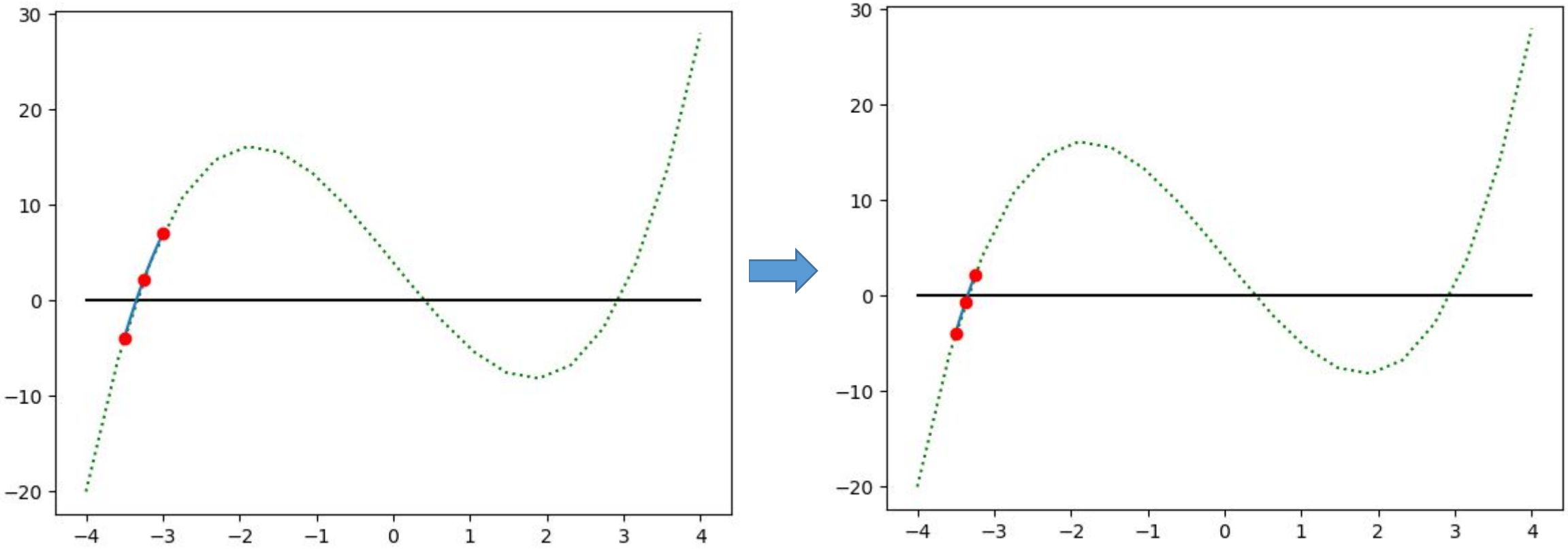
- Replace `b` by `mid`

`a`, `mid` and `b`



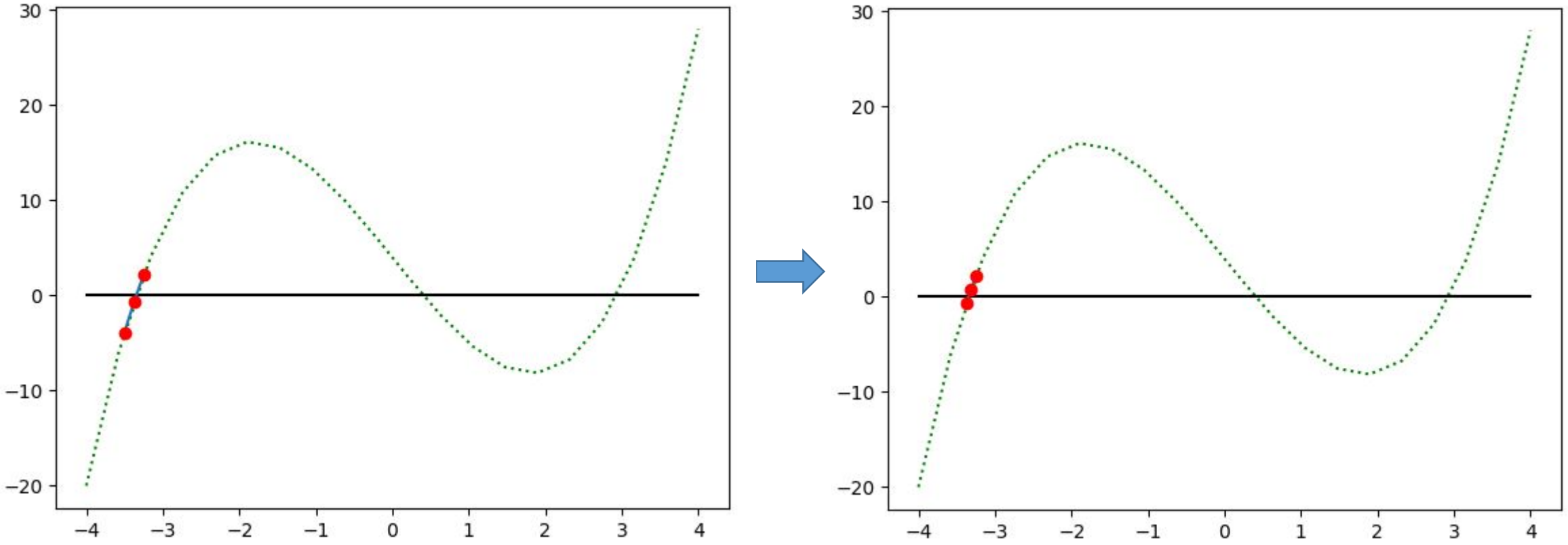
- Replace `a` by `mid`

`a`, `mid` and `b`



- Replace `b` by `mid`

`a`, `mid` and `b`



- After more iterations, `x` = -3.34596…

Bisection Method

- Write a function `bisection(start,end,f)` to solve for `x` in `f(x)=0` using the bisection method
- You can set “error” to a small number e.g. 0.00000001

```
def g(x):
    return (x/40)**5 - 8*(x/40)**3 + x/4 + 6

f = lambda x : x**3  - 10*x + 4
```

- `bisection(-4,4,f)` should output -3.346 (no reverse)
- `bisection(-100,80,g)` should give output 63.396

```
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
```

```
if f(mid) > 0:
    a = mid
else:
    b = mid
```

Advanced Sorting

Partition

- We are given a list `lst` of `n` unique numbers. (However, it is not difficult to solve the problem even if there are duplicate numbers.)
- We pick any number from `lst`, say `x`.
- For the rest of the numbers, we then separate them into two lists:
  - `lsta` which contains all the numbers smaller than `x`, and
  - `lstb` which contains the rest of the numbers
- Let's give a name to this functionality: 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 become sorted after the “magic”.
- Finally, we output `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]
```

Q_ _ _ _ Sort

```
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
```

- Write the function `partition(lst,x)` to finish this magic.
- What is this “magic”?
- And what is the magic that we have performed?