In-Class Exercise Q1

```
def power1(x, n):
  if n == 0:
    return 1
  else:
    return x * power1(x, n-1)
```

a. How many multiplications will the function **power1** perform when **x** = 3 and **n** = 16? Answer: 16

b. What is the complexity of **power1**? Answer: O(n)

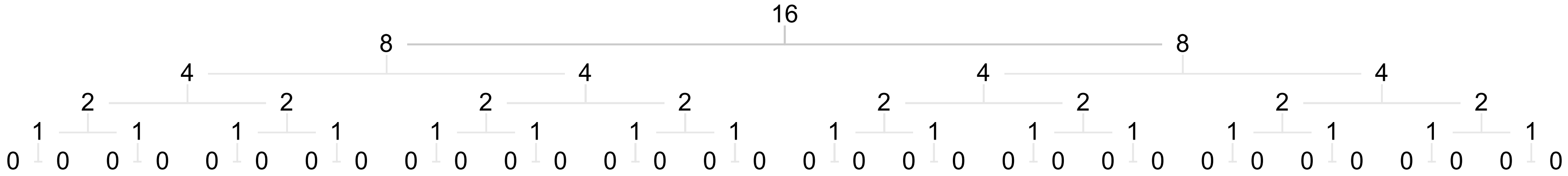
In-Class Exercise Q2

```
def power2(x, n):
  if n == 0:                          #when n is 0, no multiplication needed
    return 1
  elif n % 2 == 0:                    #when n is even, 1 multiplication needed
    return power2(x, n//2) * power2(x, n//2)
  else:                               #when n is odd, 2 multiplications needed
    return x * power2(x, n//2) * power2(x, n//2)
```

a. How many multiplications will the function **power2** perform when **x** = 3 and **n** = 16?

b. What is the complexity of **power2**?

power2(3, 16)


16
8
8
4
4
4
4
2
2
2
2
2
2
2
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

power2(3, 16)

	No of instances	No of multiplications
power2(3, 16)	1	1 x 1 = 1
power2(3, 8)	2	2 x 1 = 2
power2(3, 4)	4	4 x 1 = 4
power2(3, 2)	8	8 x 1 = 8
power2(3, 1)	16	16 x 2 = 32
power2(3, 0)	32	32 x 0 = 0
total		47

Note that there are 2 multiplications if n is odd, 1 multiplication if n is even, and 0 multiplication for n == 0.

Note that there are 2 recursive calls for any n > 0, and zero recursive call for n == 0.

Complexity of power2

- Complexity is based on the number of operations (multiplications).
- For a problem size n:
  - There will be up to (n - 1) instances of power2(x, >1).
    - Each instance has up to 2 multiplications.
  - There will be up to n instances of power2(x, 1).
    - Each instance has exactly 2 multiplications.
  - There will be up to 2n instances of power2(x, 0).
    - Each instance has 0 multiplication, so not counted.
- Maximum possible number of multiplications: 2(n-1) + 2n
- Therefore, complexity is O(n).

In-Class Exercise Q3

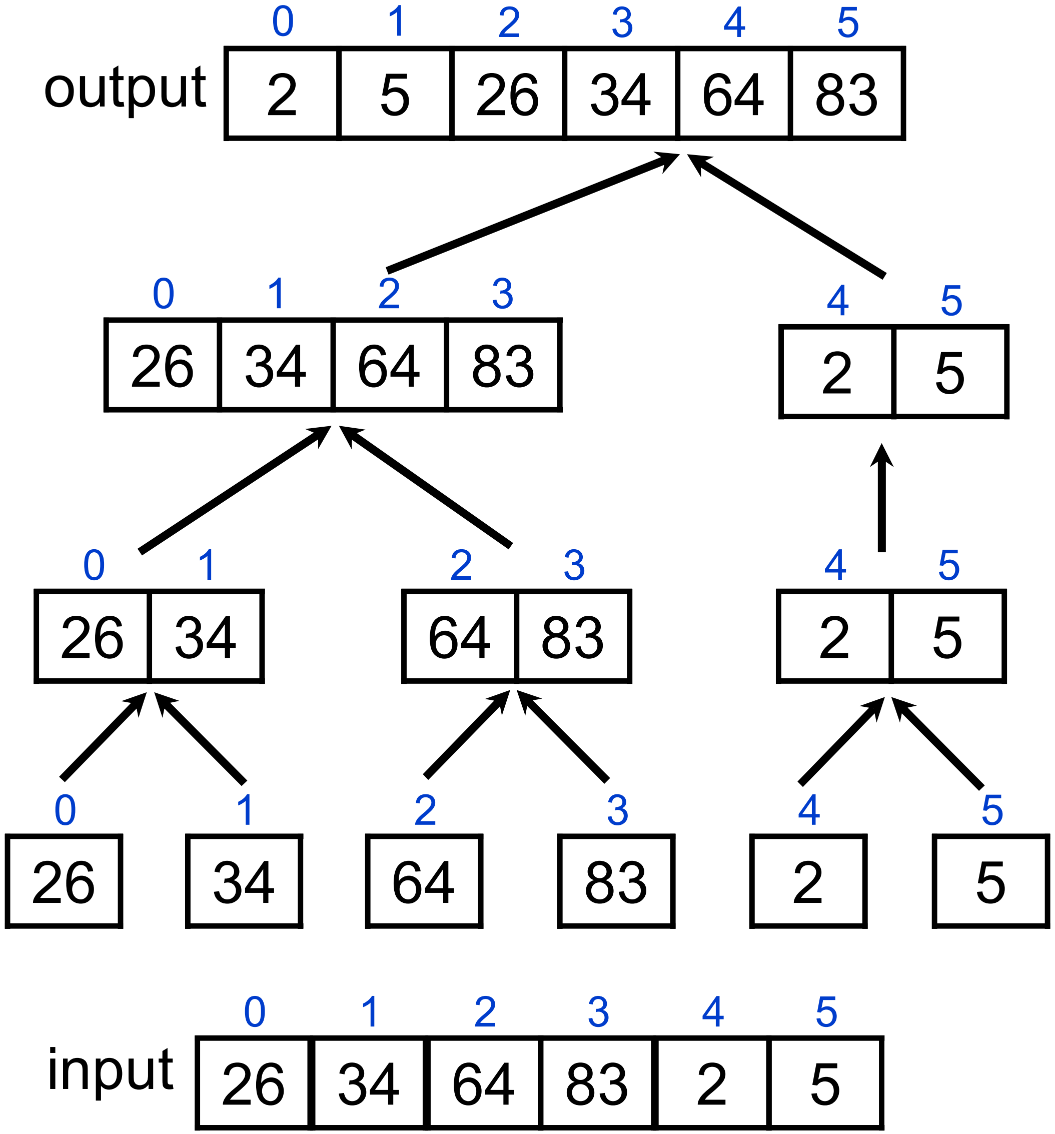
For the following input arrays, determine the state of the array just before the final merge step for the two versions of merge sort:

- bottom-up non-recursive version
- top-down recursive version

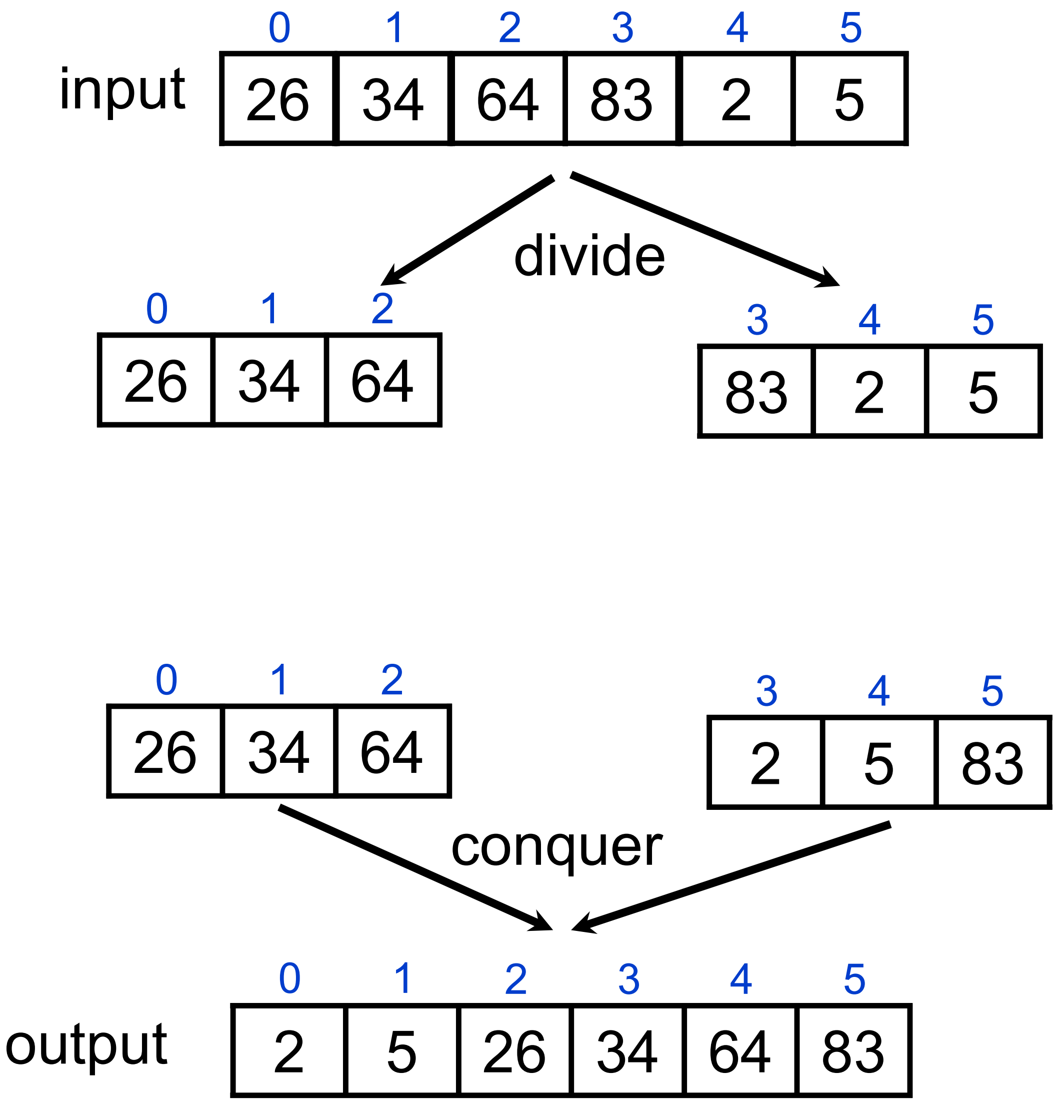
a. [26, 34, 64, 83, 2, 5]

b. [52, 30, 98, 59, 67, 4, 64, 12, 79]

Merge sort: Bottom-up Approach



Merge sort: Top-down Approach



input

0	1	2	3	4	5
26	34	64	83	2	5

0	1	2
26	34	64

3	4	5
83	2	5

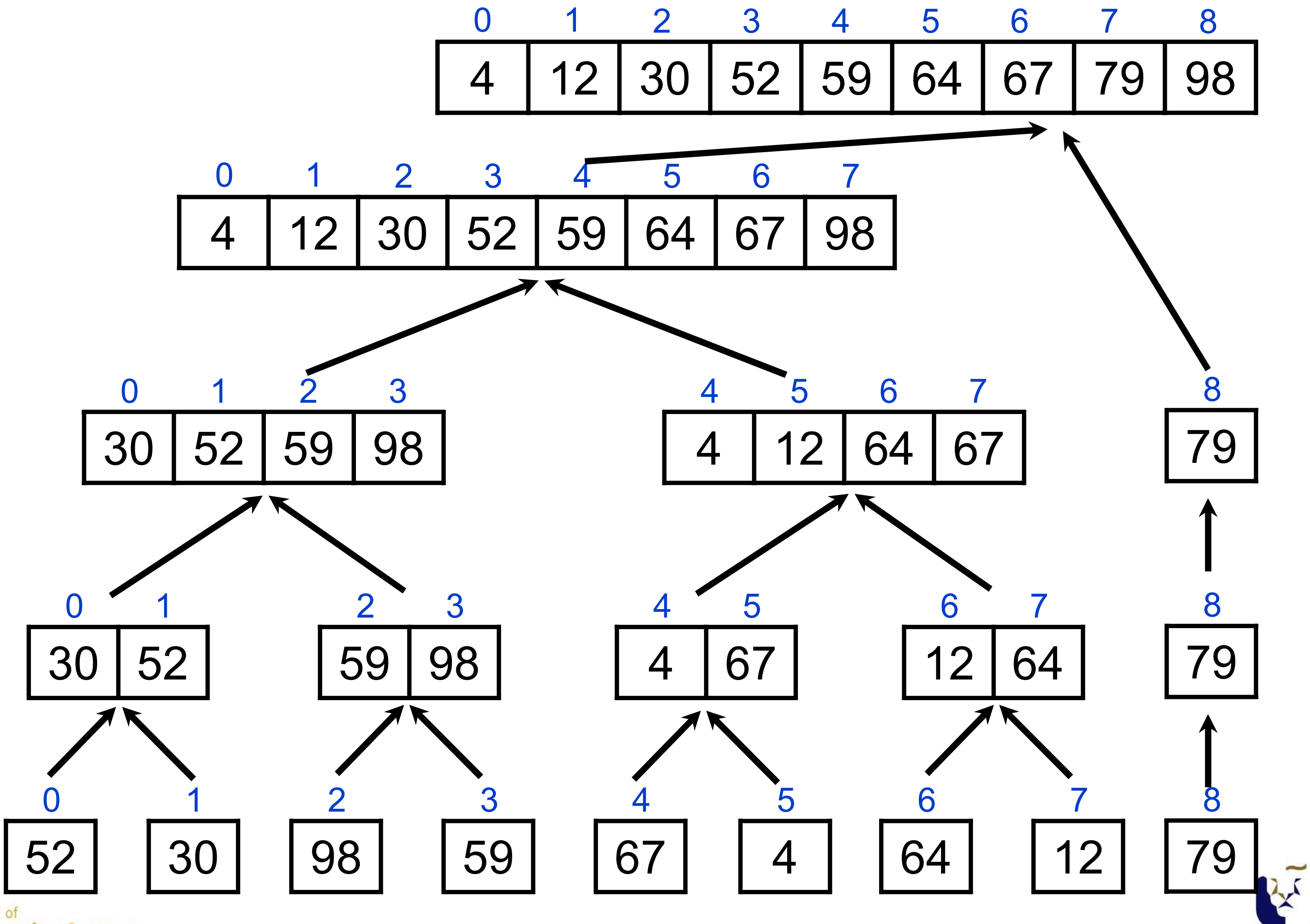
0	1	2
26	34	64

3	4	5
2	5	83

output

0	1	2	3	4	5
2	5	26	34	64	83

Merge sort: Bottom-up Approach



0	1	2	3	4	5	6	7	8
4	12	30	52	59	64	67	79	98

0	1	2	3	4	5	6	7
4	12	30	52	59	64	67	98

0	1	2	3
30	52	59	98

4	5	6	7
4	12	64	67

8
79

0	1
30	52

2	3
59	98

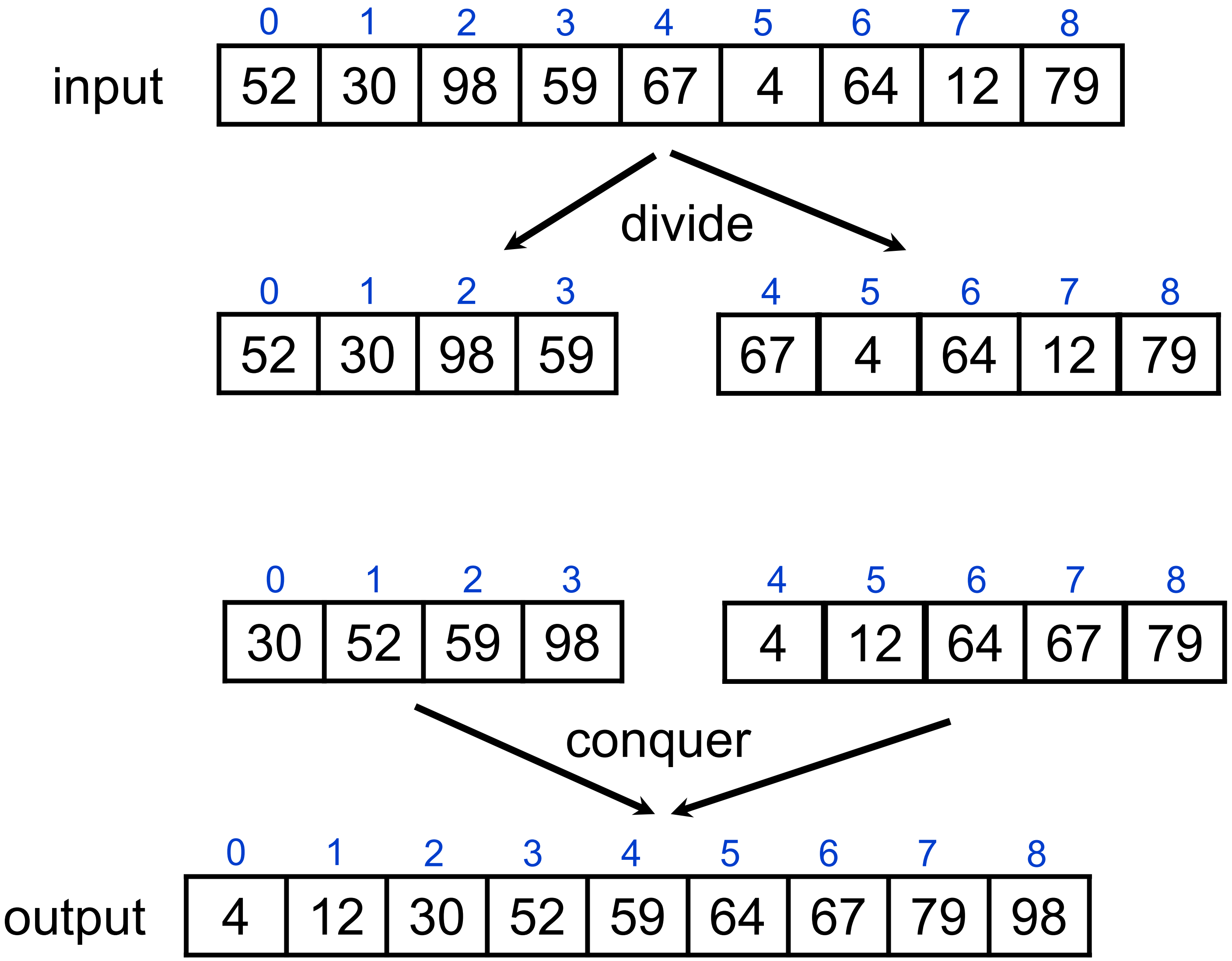
4	5
4	67

6	7
12	64

8
79

0	1	2	3	4	5	6	7	8
52	30	98	59	67	4	64	12	79

Merge sort: Top-down Approach



input

0	1	2	3	4	5	6	7	8
52	30	98	59	67	4	64	12	79

divide

0	1	2	3
52	30	98	59

4	5	6	7	8
67	4	64	12	79

0	1	2	3
30	52	59	98

4	5	6	7	8
4	12	64	67	79

conquer

output

0	1	2	3	4	5	6	7	8
4	12	30	52	59	64	67	79	98