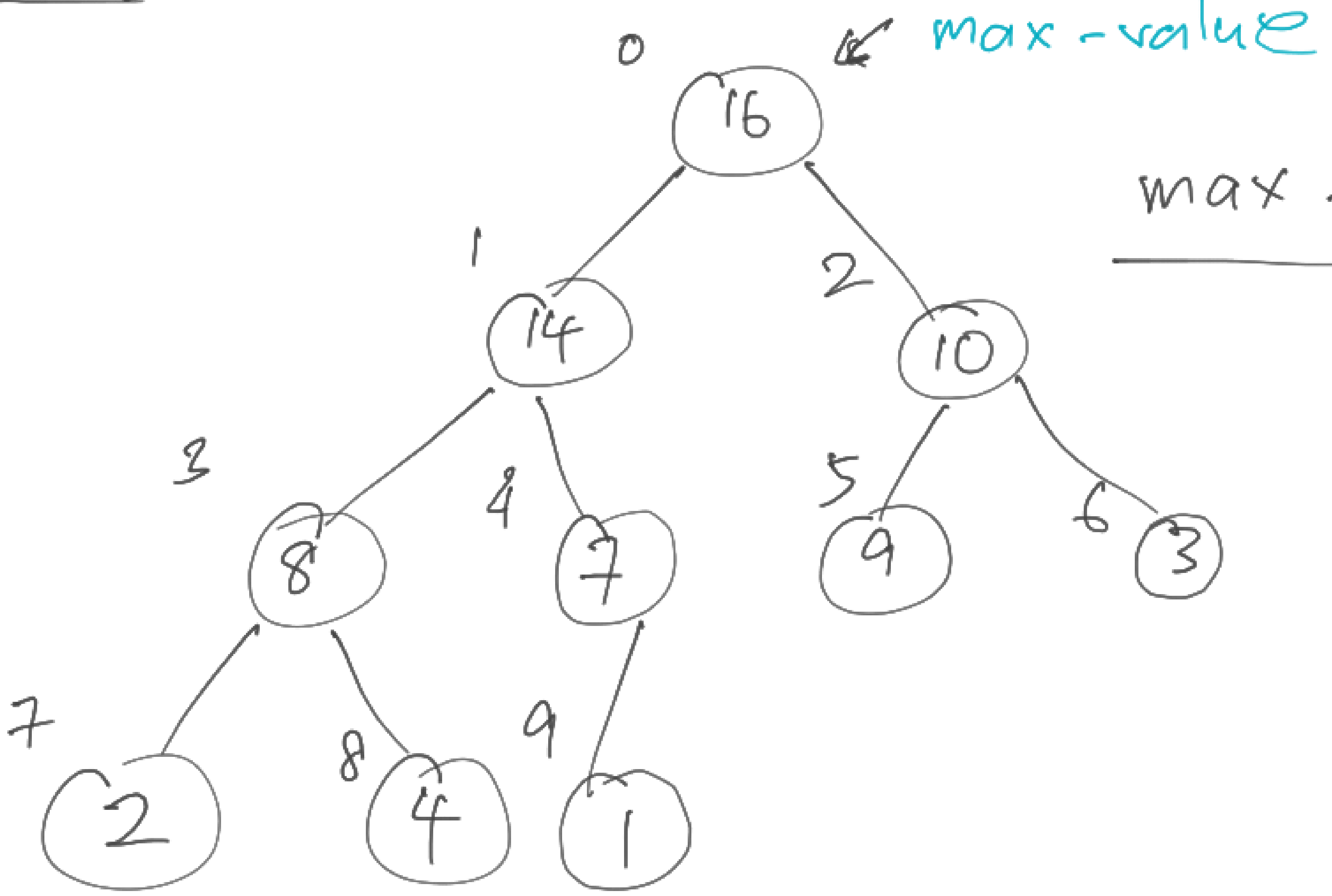
0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1



max-heap

↳ min-heap.

```python
def parent_of(index):
    return (index-1)// 2
```

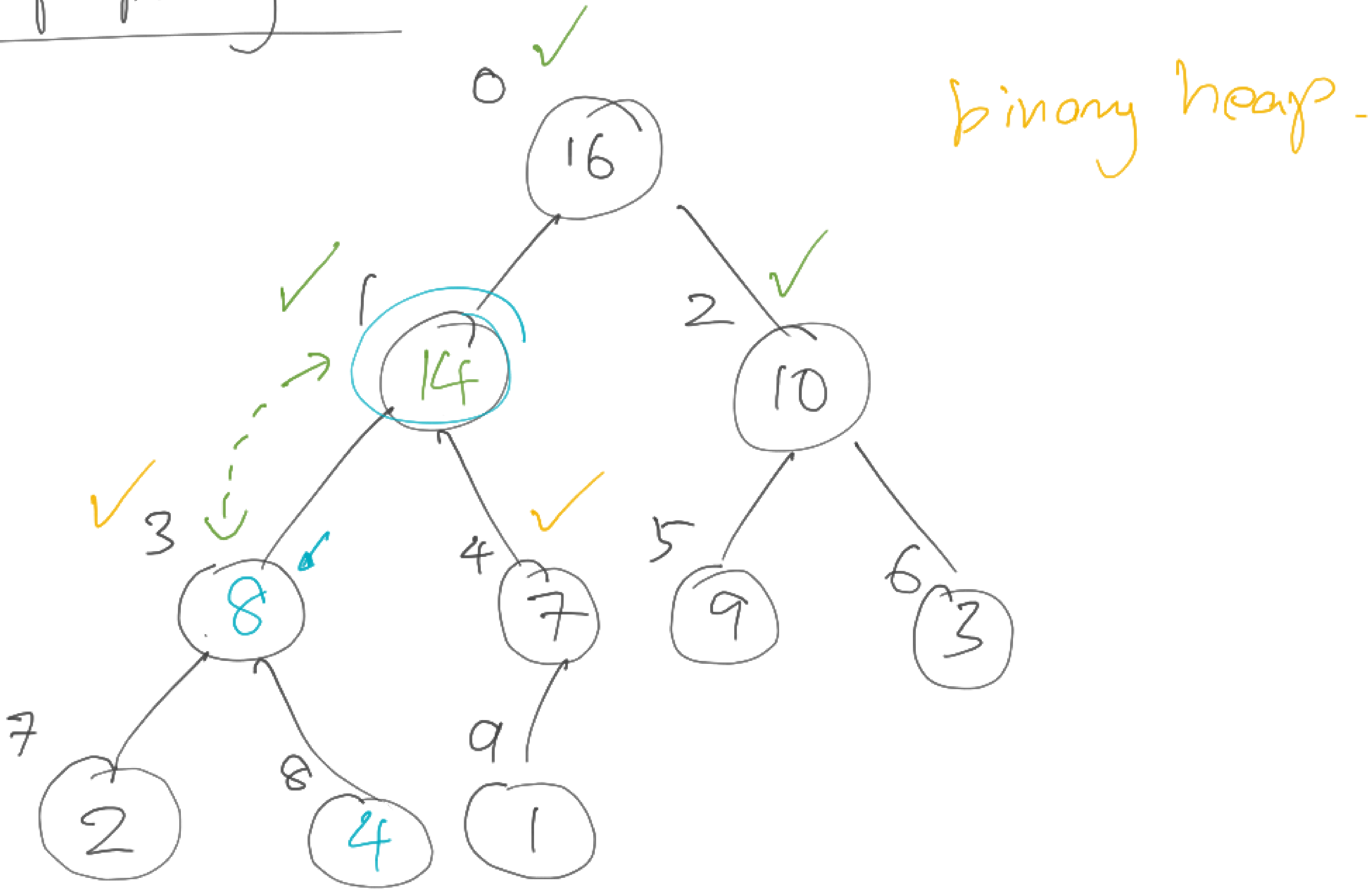
```python
assert parent_of(1)==0
assert parent_of(2)==0
⋮
```

```python
def left_of(index):
    return 2*index + 1
```

```python
def right_of(index):
    return (index + 1) * 2.
```

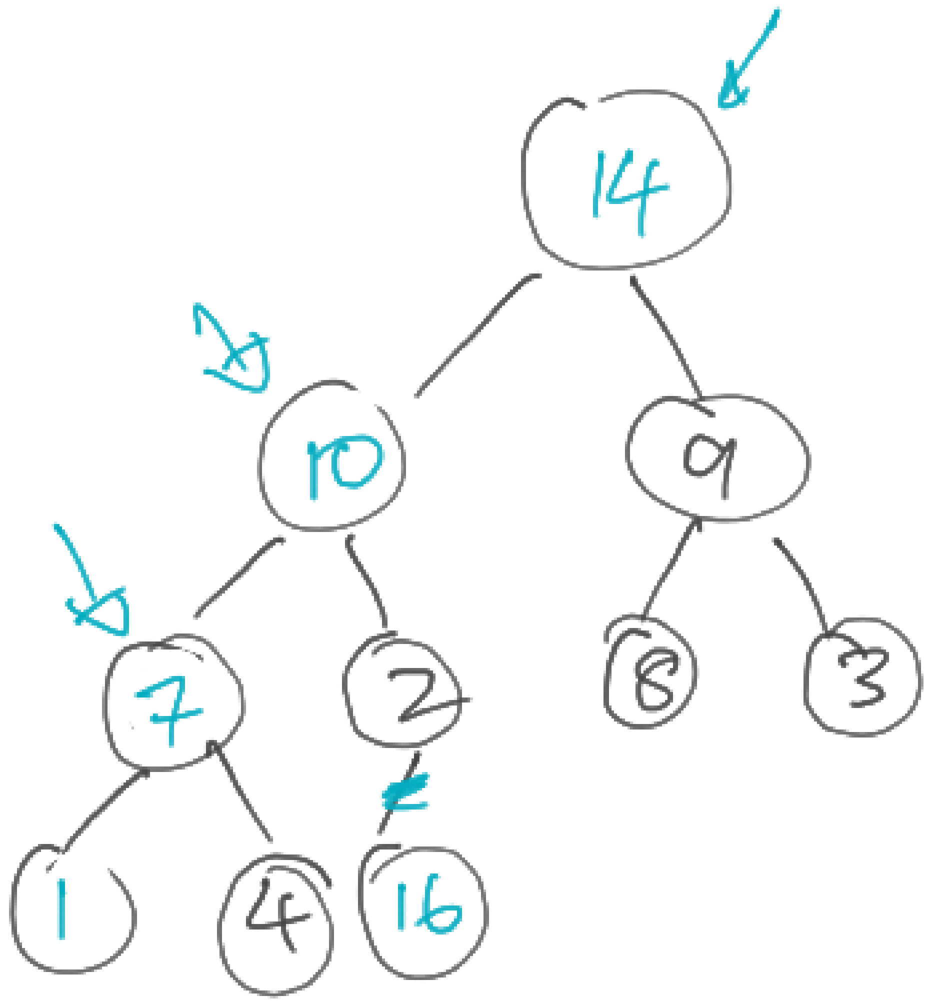
```
def max-child (array, index, heap_size):
    if right_of(index) >= heap_size:
        return left_of(index)

    else:
        if array[left_of(index)] > array[right_of(index)]:
            return left_of(index)

        else
            return right_of(index)
```

Heap property



```
def max_heapify(array, index, size):
    cur_idx = index
    while left_of(cur_idx) < size:
        max_chid_idx = max_child(array, cur_idx, size)
        if array[max_child_idx] > array[cur_idx]:
            array[max_child_idx], array[cur_idx] = array[cur_idx],
                                                   array[max_child_idx]
        cur_idx = max_child_idx
```

```
def build_max_heap(array):
    for pos in range(len(array)//2 -1, -1, -1):
        max_heapify(array, pos, len(array))
```



[16, 14, 9, 10, 2, 8, 3, 7, 4, 1]

sorted list

[16]

[1, 14, 9, 10, 2, 8, 3, 7, 4] [16]

[14, 10, 9, 7, 2, 8, 3, 1, 4] [16]

max

memory

[4, 10, 9, 7, 2, 8, 3, 1] [14, 16]

in-place

```
def heapsort(array):
    heap_size = len(array)
    build_max_heap(array)

    while heap_size >= 1:
        array[0], array[heap_size - 1] = array[heap_size - 1], array[0]
        heap_size -= 1
        max_heapify(array, 0, heap_size)
```

swap.

Big Oh $\rightarrow$ $O(n) \rightarrow$ linear time

$O(1) \rightarrow$ constant time

$O(n^2) \rightarrow$ quadratic time.

$$f = O(g)$$

$$\boxed{\limsup_{x \to \infty} \frac{f(x)}{g(x)} < \infty}$$

$\rightarrow x \log x$

$\rightarrow x \log x$

$g(x) \rightarrow$ tight upper bound on $f(x)$

$\rightarrow O(\underline{n \log n})$

2

$$\log(y) = 2\log(x)$$

computation time

no. of inputs.

$$y = \underline{\underline{x^2}}$$

step 1 : n = length of array . $\longrightarrow$ O(1) .

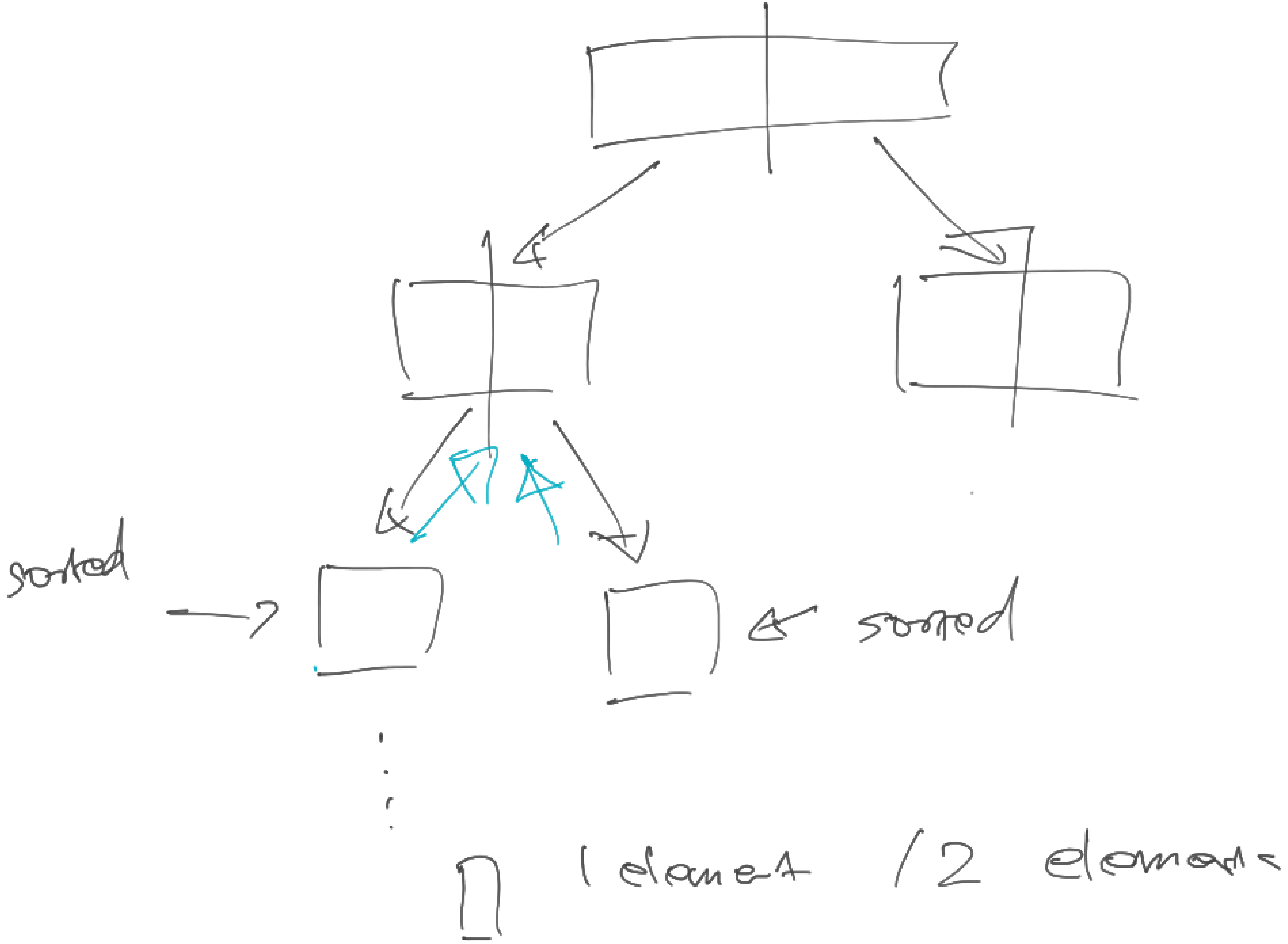
step 2 : for outer fr. $\boxed{1 \text{ to } n-1}$ do : $\longrightarrow$ ~~O(n-1)~~ O(n)

2.1 : for inner_index $\boxed{1 \text{ to } n-1}$ $\longrightarrow$ O(n)

~~3n~~ × O(1) $\longleftarrow$ O(1) .

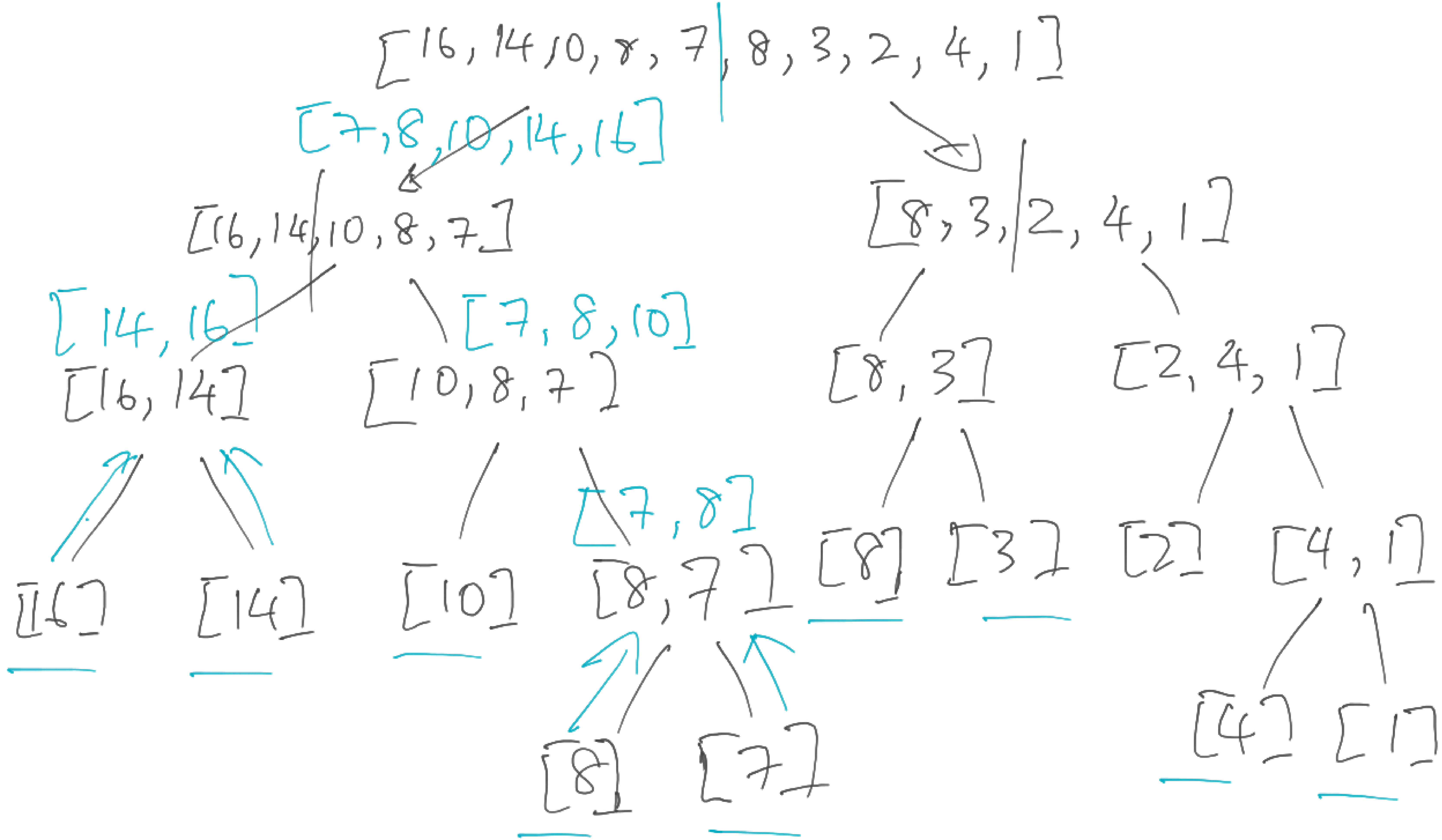
$$T(n) = \cancel{O(1)} + \underline{O(n) \times O(n) \times O(1)}$$

$$= O(n^2)$$



sorted
sorted
1 element / 2 elements

```
def move_disks (n, from_tower, to_tower, aux_tower):
    result = []

    if n==1:
        return [f"Move disk {n:} from {from_tower:}
                                    to {to_tower:}."]

    else :
        result = move_disk (n - 1, from_tower, aux_tower
                                    , to_tower)

        result += [f"Move disk {n:} from {from_tower:}
                                    to {to_tower:}."]
        result += move_disk(n-1, aux_tower, to_tower, from_tower)
    return result
```


[16, 14, 10, 8, 7, 8, 3, 2, 4, 1]
[7, 8, 10, 14, 16]
[16, 14, 10, 8, 7]
[8, 3, 2, 4, 1]
[14, 16]
[7, 8, 10]
[16, 14]
[10, 8, 7]
[8, 3]
[2, 4, 1]
[7, 8]
[16]
[14]
[10]
[8, 7]
[8]
[3]
[2]
[4, 1]
[8]
[7]
[4]
[1]

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n > 1 \end{cases}$$

$$T(n) = 2T(n/2) + cn$$