Recursion

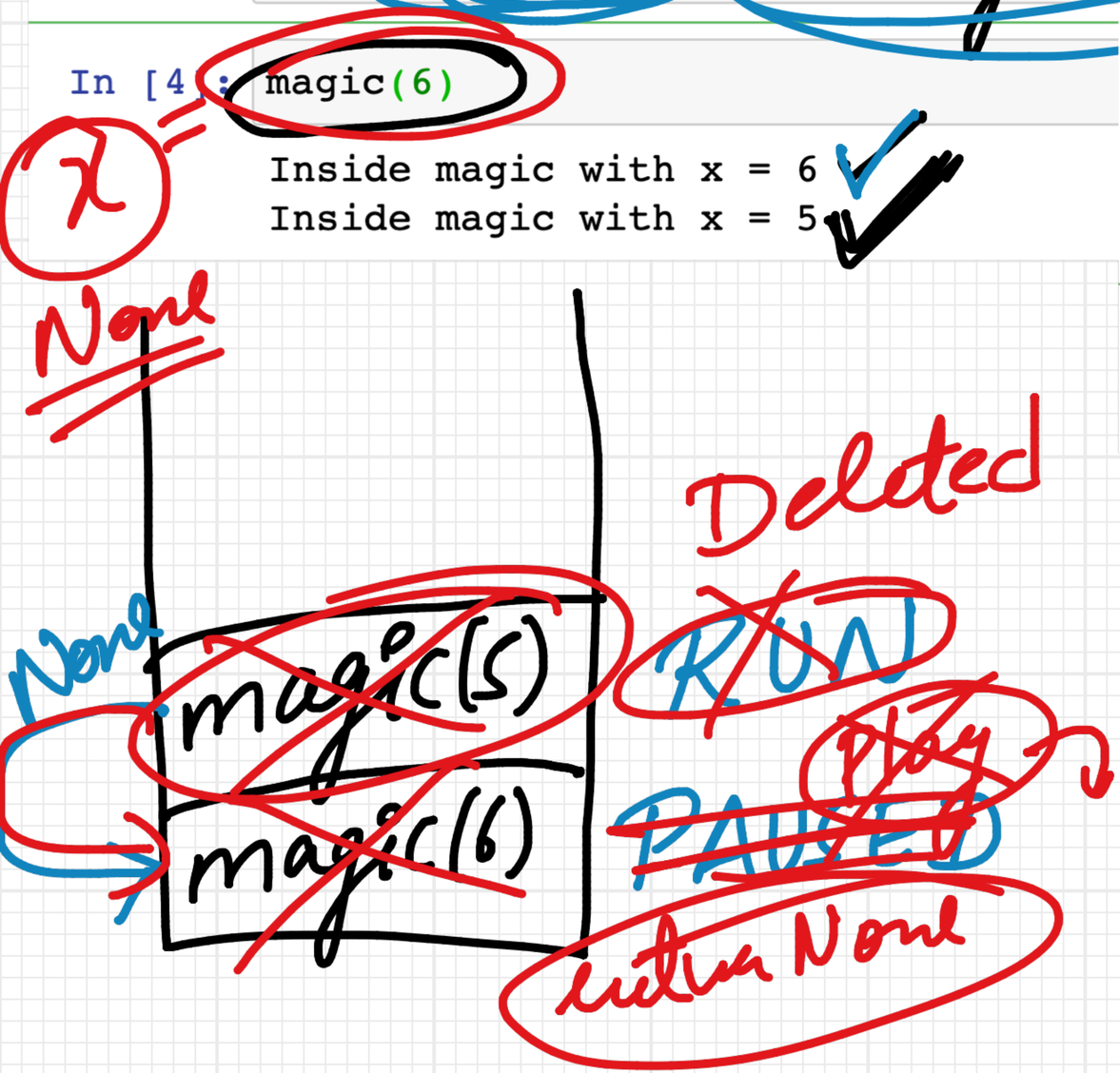
=> function calling itself

```
def magic(x):
    if x == 5:
        return
    magic(x-1)
```

```
In [2]: def magic(x):
            print(f'Inside magic with x = {x}')
            if x <= 5:
                return
            else:
                magic(x-1)
```

```
In [4]: magic(6)

        Inside magic with x = 6
        Inside magic with x = 5
```




```
In [2]: def magic(x):
            print(f'Inside magic with x = {x}')
            if x <= 5:
                return
            else:
                magic(x-1)
```

```
In [3]: magic(5)

        Inside magic with x = 5
```

Who called magic(5)?

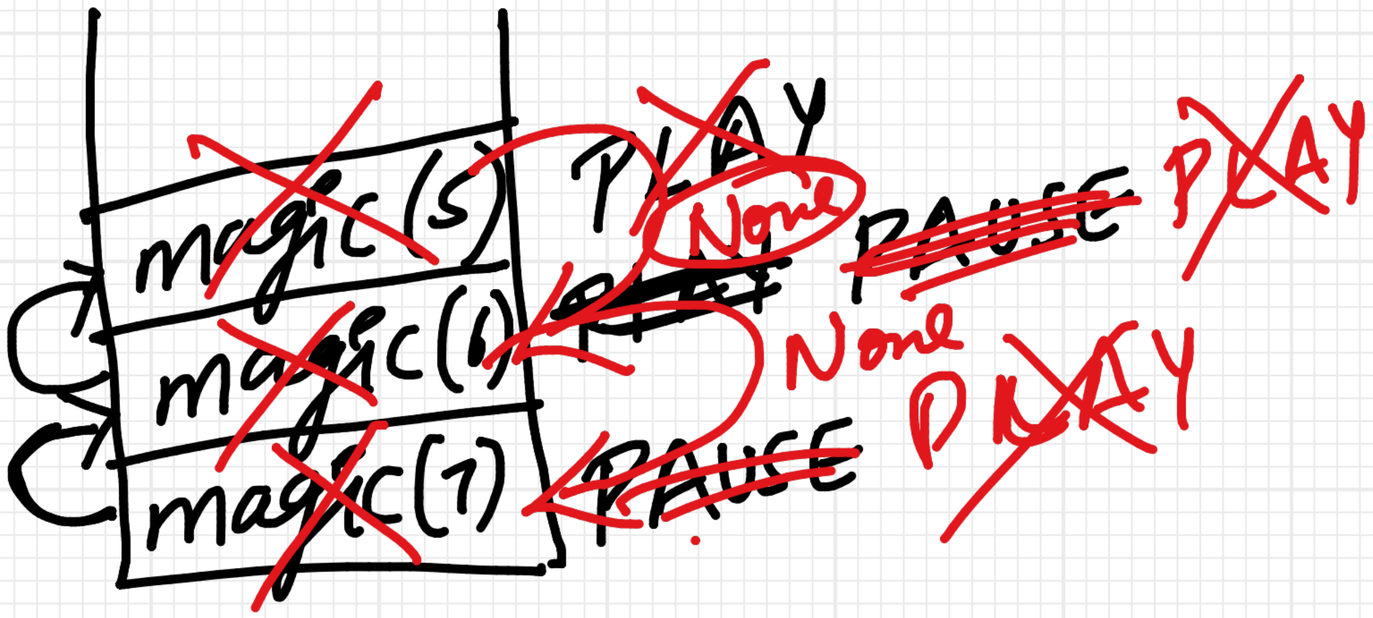
magic(1)

magic(5) is executed completely?

In [5]:
```
def magic(x):
    print(f'Inside magic with x = {x}')
    if x <= 5:
        return
    else:
        magic(x-1)
```

In [8]:
```
magic(7)
```

```
Inside magic with x = 7
Inside magic with x = 6
Inside magic with x = 5
```



Factorial

$0! = 1$

$1! = 1 \times 0! = 1 \times 1 = 1$

$2! = 2 \times 1! = 2 \times 1 = 2$

$3! = 3 \times 2! = 3 \times 2 = 6$

$4! = 4 \times 3 \times 2 \times 1 = 24$

$4 \times 3! = 4 \times 6 = 24$

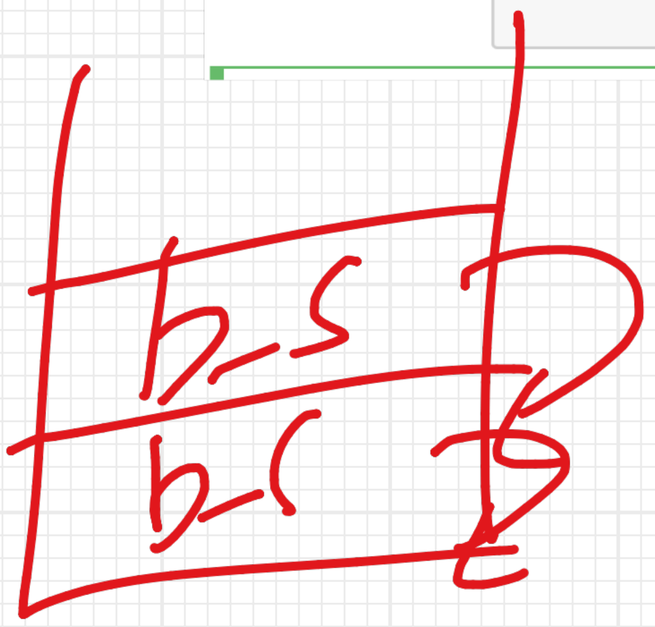
In [13]:
```
def factorial(x):
    print(f'factorial of {x}')
    if x == 0:
        return 1
    else:
        return x*factorial(x-1)
```

In [15]:
```
y = 4
answer = factorial(y)
print(f'factorial of {y} is {answer}')
```

```
factorial of 4
factorial of 3
factorial of 2
factorial of 1
factorial of 0
factorial of 4 is 24
```

Recursion

In [49]:
```
def binary_search(l, key):
    n = len(l)
    mid = n//2
    if l[mid] == key:
        print(f'found it in list {l}')
    elif key > l[mid]:
        print(f'look at the right handside of {l[mid]} in array {l}')
        binary_search(l[mid+1:],key)
    else:
        print(f'look at the left handside of {l[mid]} in array {l}')
        binary_search(l[:mid],key)
```



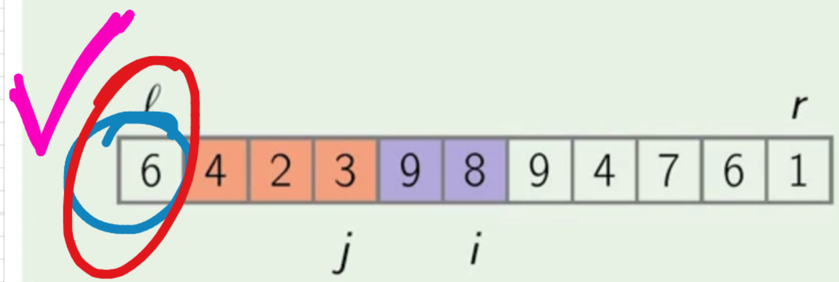
apply the same logic but on a different value.

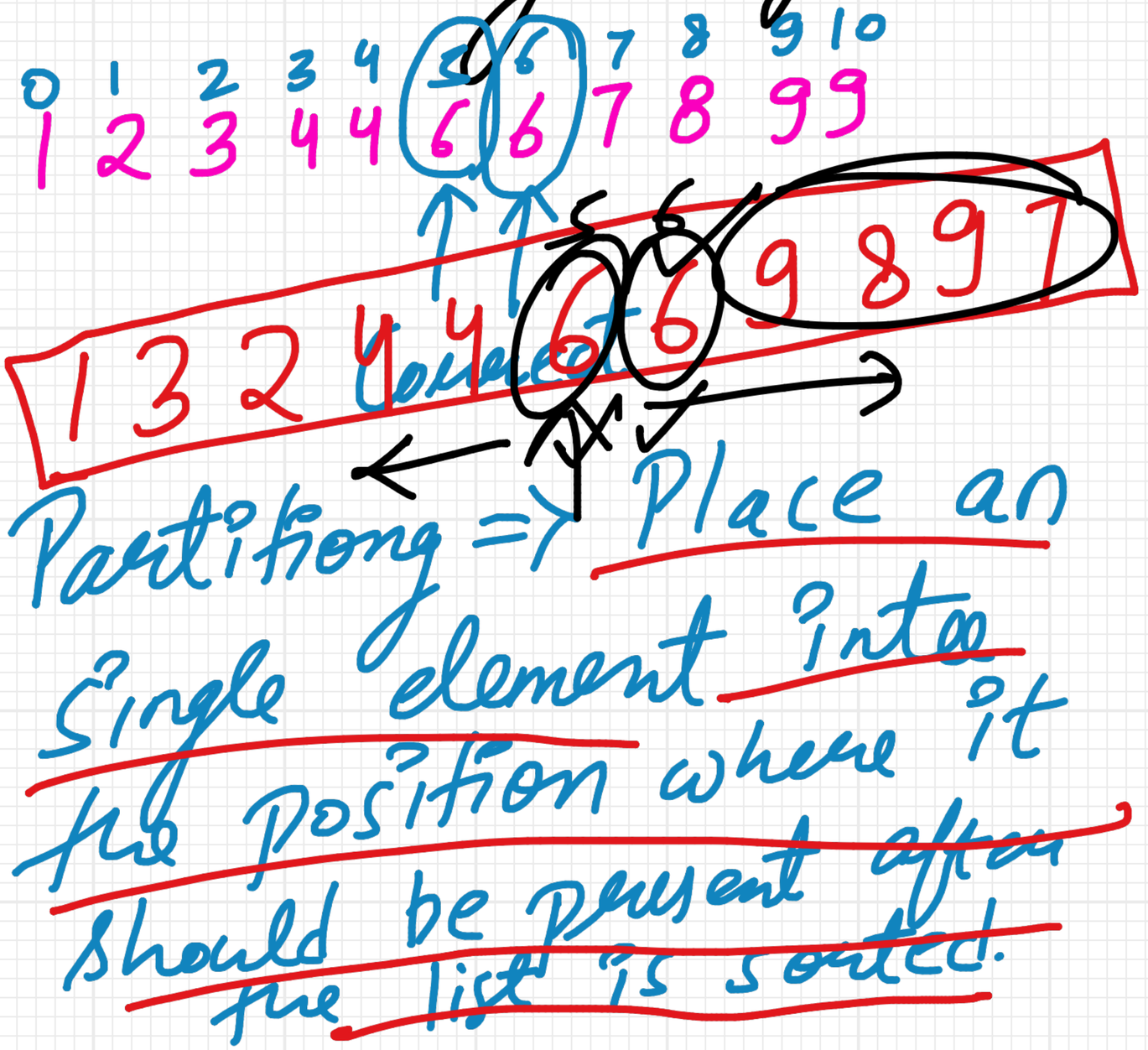
One of the best sorting algorithm

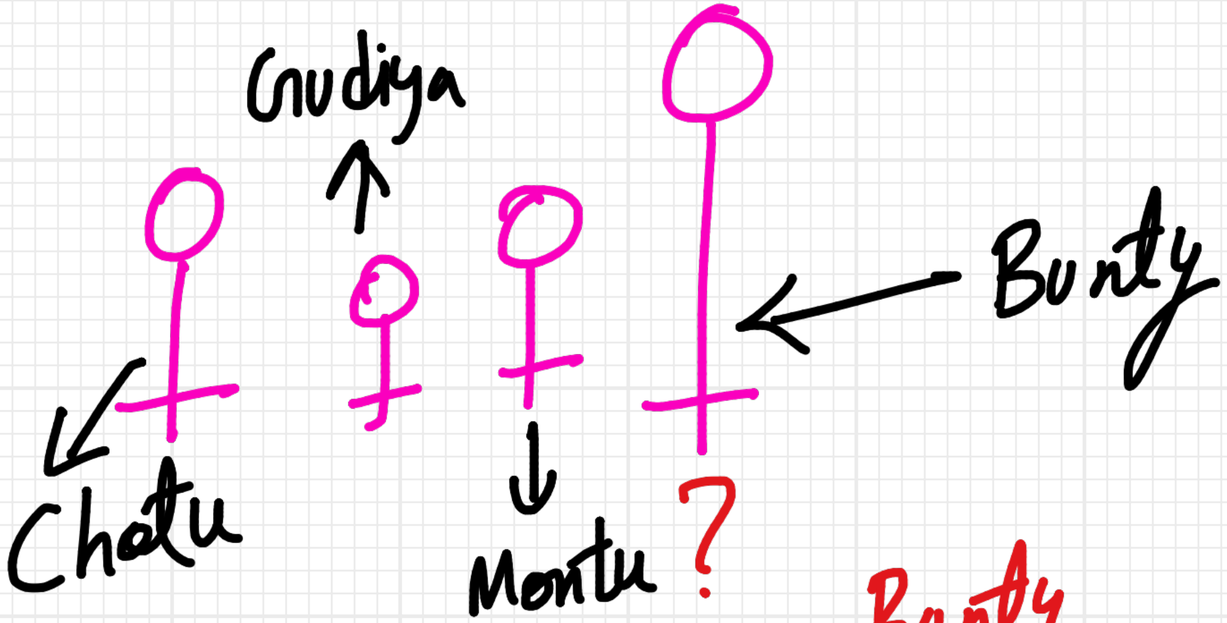
Quick Sort

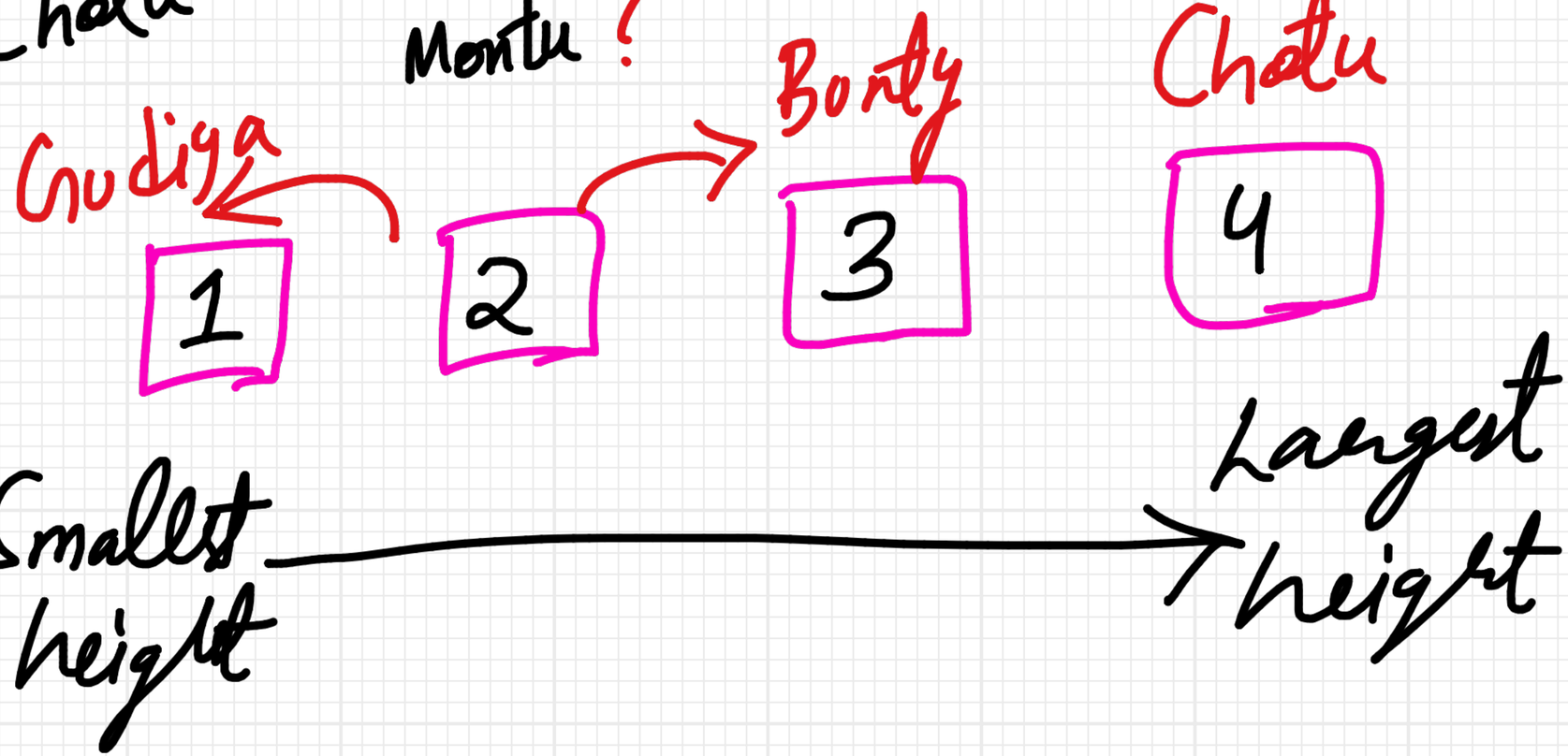
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$





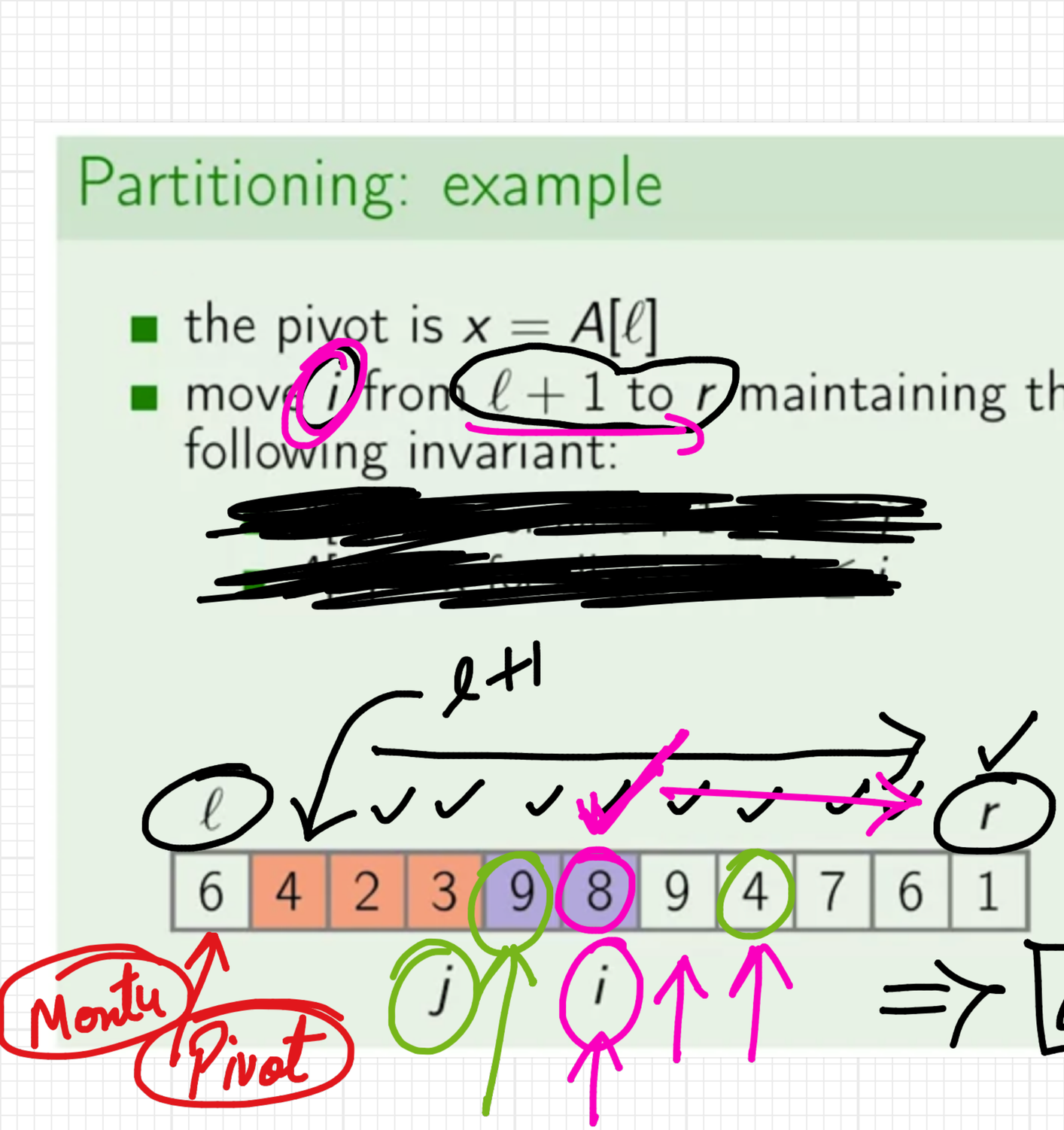

Gudiya
Bunty
Chotu
Montu
?

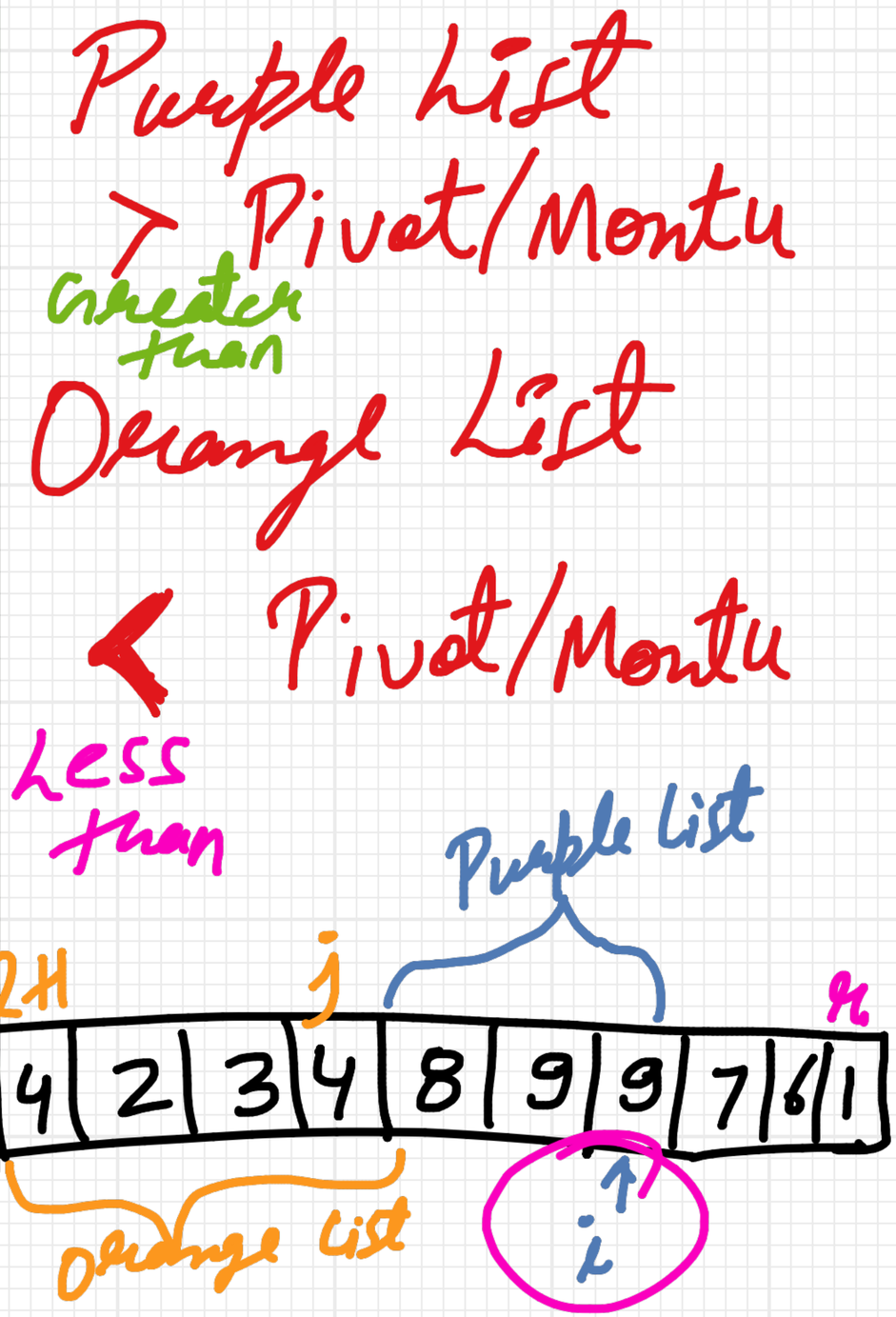

Gudiya
Bunty
Chotu
1
2
3
4
Smallest height
Largest height

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:

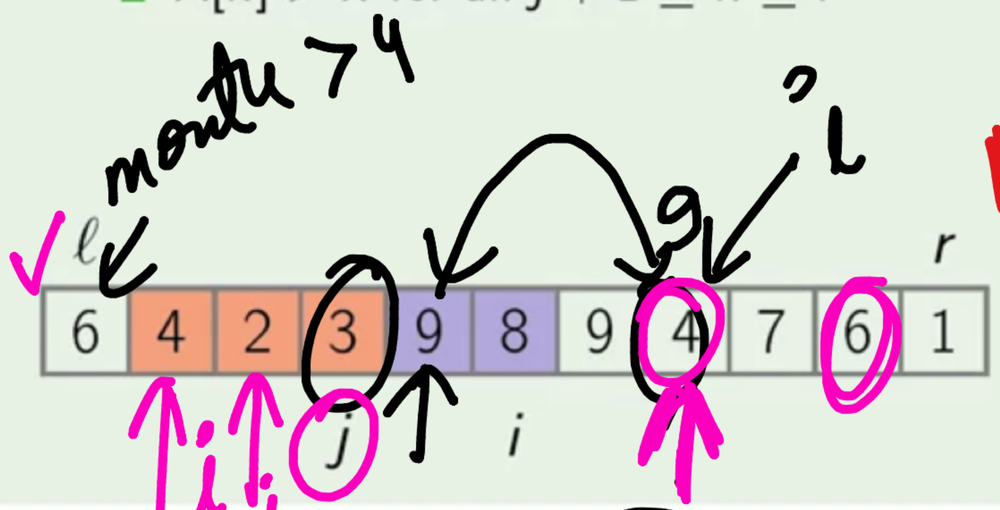
[redacted]

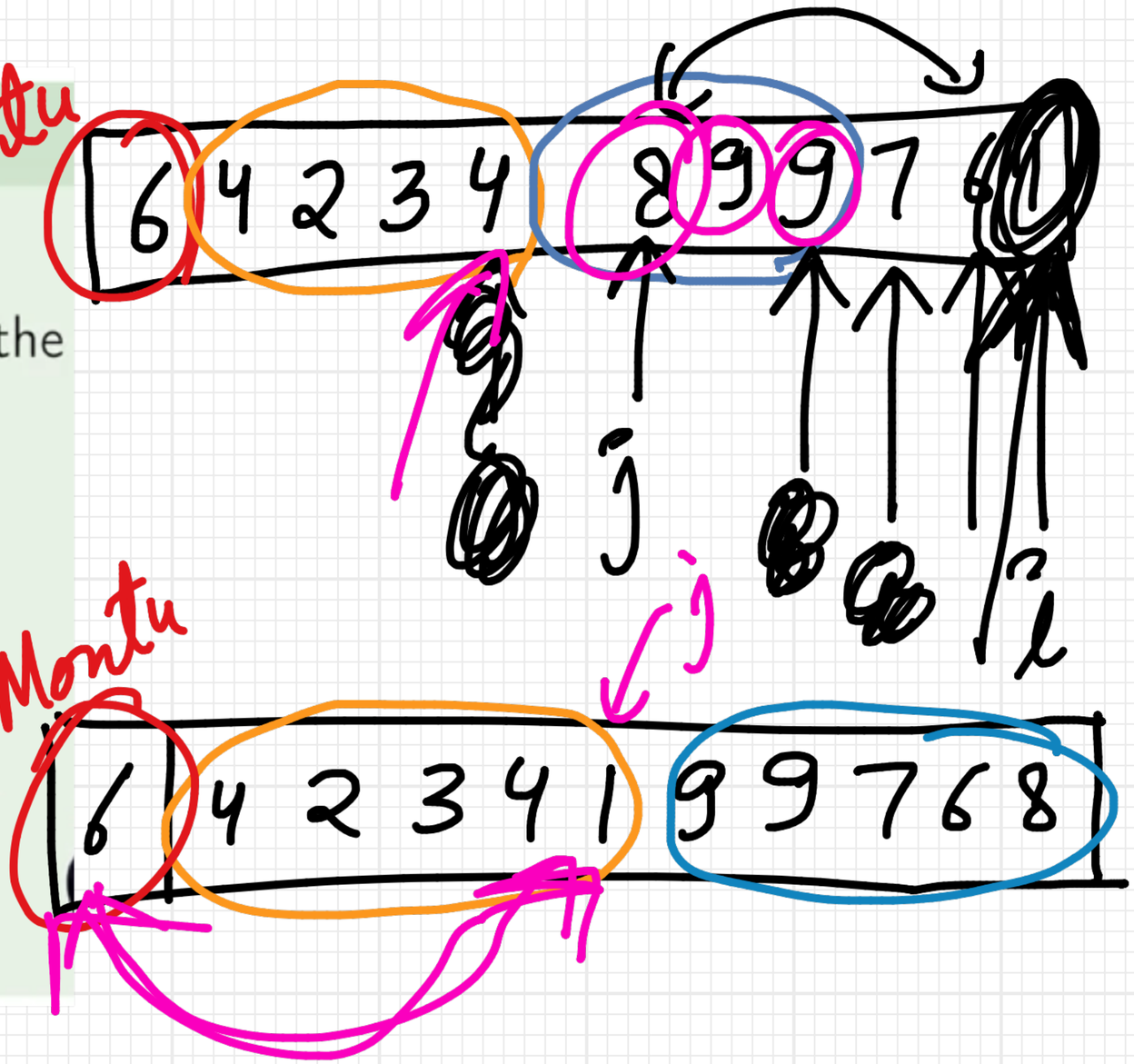




Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$





Quick Sort

```
In [42]: A = [6,4,2,3,9,8,9,4,7,6,1]
         m = partitioning(A, 0, len(A)-1)
         print(m)

         A now = [1, 4, 2, 3, 4, 6, 9, 9, 7, 6, 8]
         5
```

```
QuickSort(A, ℓ, r)
if ℓ ≥ r:
    return
m ← Partition(A, ℓ, r)
{A[m] is in the final position}
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```

exit

ℓ > r

ℓ = r

QuickSort m

QuickSort

(1) [4, 2, 3, 4]

[1], [2] m