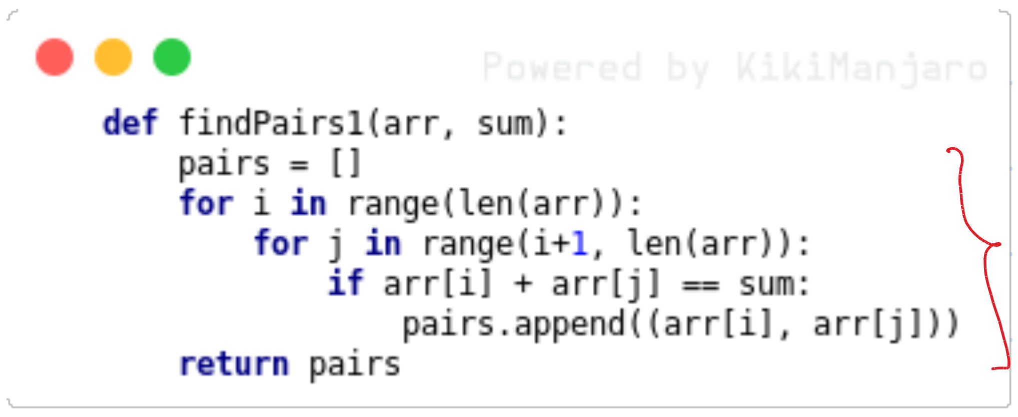
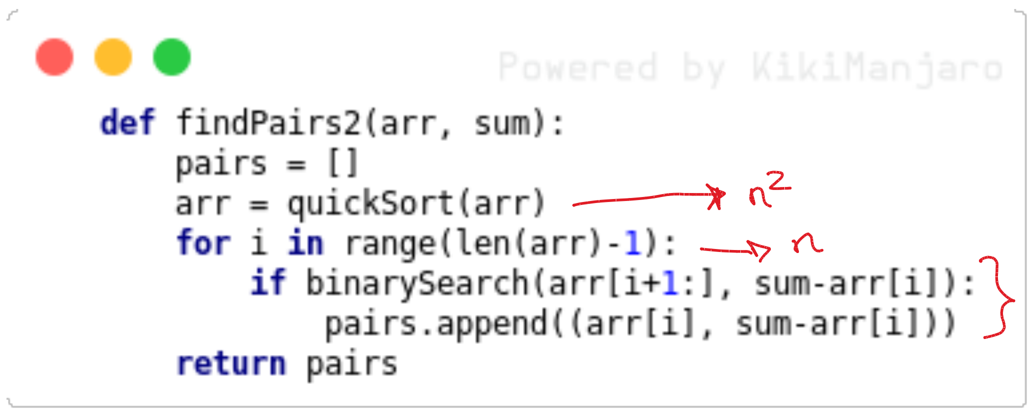


```python
def findPairs1(arr, sum):
    pairs = []
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            if arr[i] + arr[j] == sum:
                pairs.append((arr[i], arr[j]))
    return pairs
```

$\}\ O(n^2)$

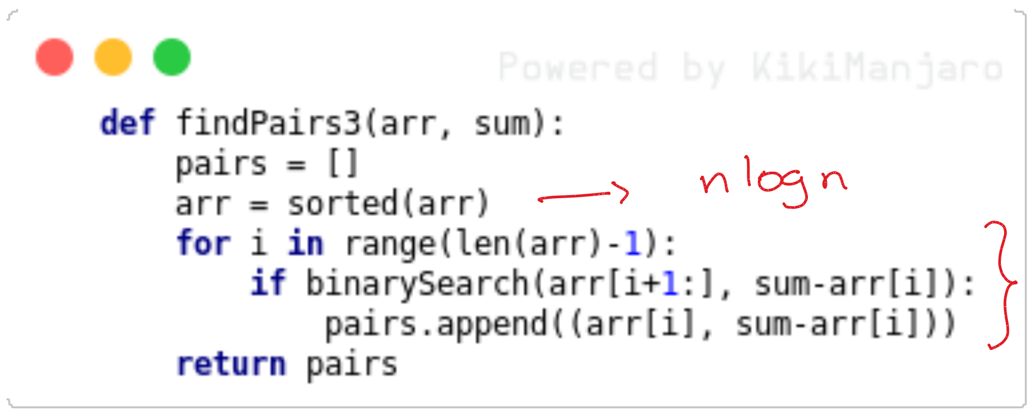


```python
def findPairs2(arr, sum):
    pairs = []
    arr = quickSort(arr)
    for i in range(len(arr)-1):
        if binarySearch(arr[i+1:], sum-arr[i]):
            pairs.append((arr[i], sum-arr[i]))
    return pairs
```

$\rightarrow n^2$

$\rightarrow n$

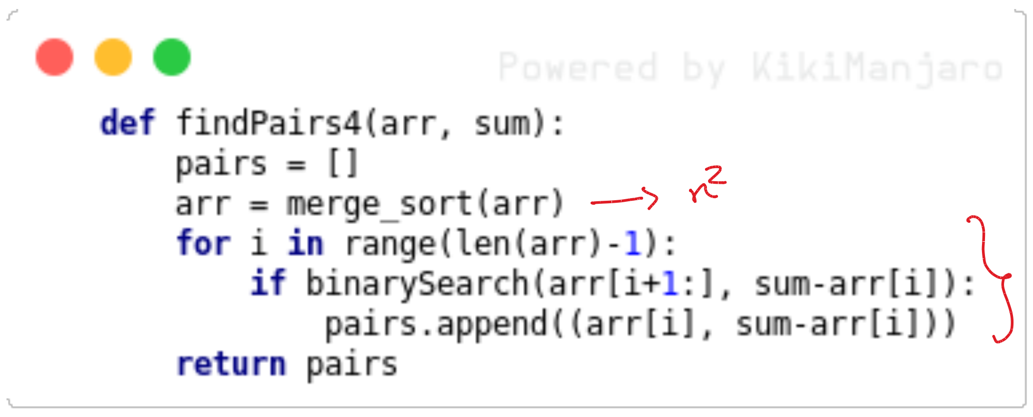
$\}\ \log(n)\ \}\ O(n^2 + n\log n) = O(n^2)$



```python
def findPairs3(arr, sum):
    pairs = []
    arr = sorted(arr)
    for i in range(len(arr)-1):
        if binarySearch(arr[i+1:], sum-arr[i]):
            pairs.append((arr[i], sum-arr[i]))
    return pairs
```

$\rightarrow n\log n$

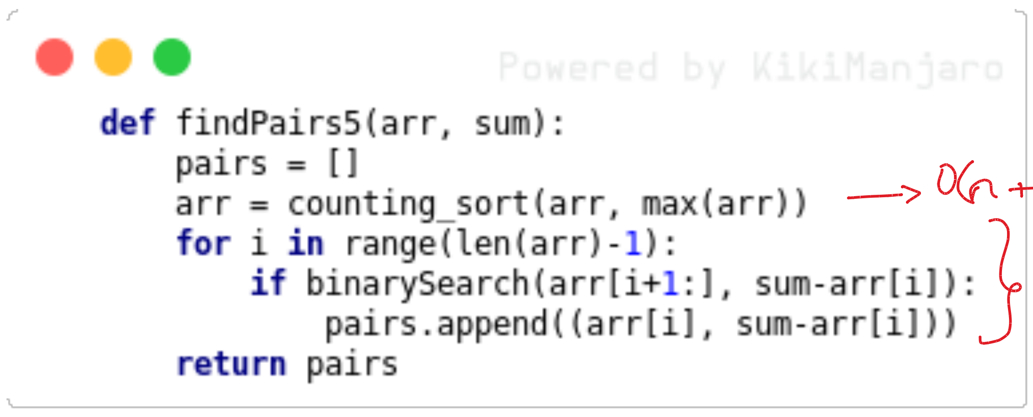
$\}\ n\log n\ \}\ O(n\log n)$



```python
def findPairs4(arr, sum):
    pairs = []
    arr = merge_sort(arr)
    for i in range(len(arr)-1):
        if binarySearch(arr[i+1:], sum-arr[i]):
            pairs.append((arr[i], sum-arr[i]))
    return pairs
```

$\rightarrow n^2$

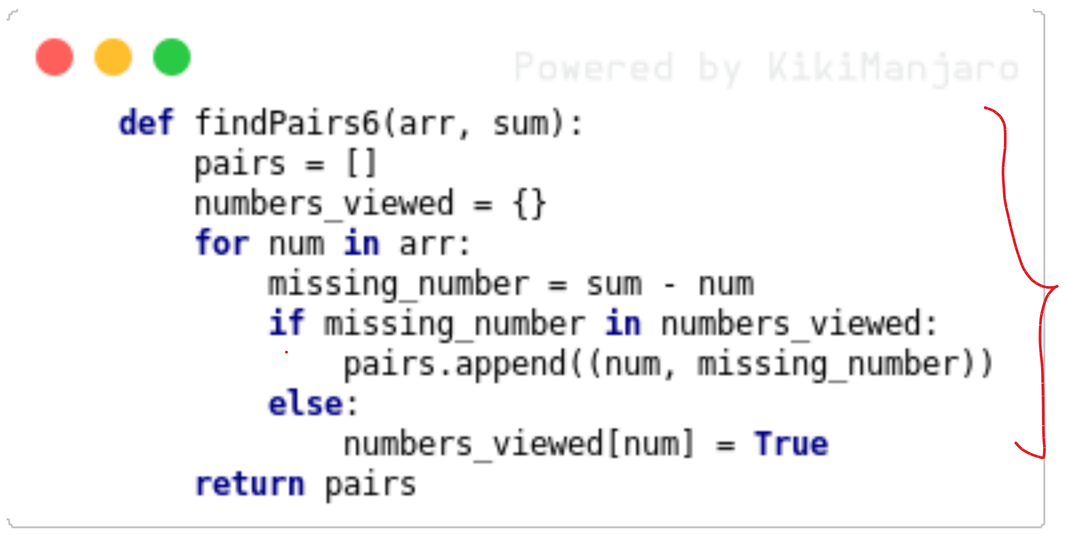
$\}\ n\log n\ \}\ O(n^2 + n\log n) = O(n^2)$



```python
def findPairs5(arr, sum):
    pairs = []
    arr = counting_sort(arr, max(arr))
    for i in range(len(arr)-1):
        if binarySearch(arr[i+1:], sum-arr[i]):
            pairs.append((arr[i], sum-arr[i]))
    return pairs
```

$\rightarrow O(n+k)$

$\}\ O(n\log n)\ \}\ O(2n + n\log n) = O(n\log n)$



```python
def findPairs6(arr, sum):
    pairs = []
    numbers_viewed = {}
    for num in arr:
        missing_number = sum - num
        if missing_number in numbers_viewed:
            pairs.append((num, missing_number))
        else:
            numbers_viewed[num] = True
    return pairs
```

$\}\ O(n)$

The complexity of the code you've provided is O(n), since the execution time of the function increases linearly with the number of elements in the list.

The function goes through the list arr once and for each element calculates the number that is missing to reach the desired sum. If the missing number is found in the numbers_viewed dictionary, the pair is added to the result, otherwise the current number is added to the dictionary. This means that the execution time of the function is directly proportional to the size of the arr list.