1)

```
def cut(n_):
    count = 0
    while n > 1:
        count = count + 1
        n = round(n / 2)

    return count

print(cut(16))
```

$\rightarrow O(1)$

$$\sum_{k=0}^{n} \frac{1}{2^k} = n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8}$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right)$$

$$= 2n = O(n)$$

$$O(n) \cdot O(1) = O(n)$$

2)

```
experiments = [ 15 , 30 , 8 , 50 , 90 , 70 , 100 , 1 , 3 ]

def worst_best(arr , l , r):

    print( "Best test result is: " + str(best(arr , l , r)))
    print( "Worst test result is: " + str(worst(arr, l, r)))


def best(arr , l , r):
    if (l == r):
        return arr[l]

    m = (l + r) // 2

    return max(best(arr, l, m), best(arr, m + 1, r))

def worst(arr , l , r):
    if (l == r):
        return arr[l]

    m = (l + r) // 2

    return min(worst(arr, l, m), worst(arr, m + 1, r))

worst_best(experiments, 0, len(experiments) - 1)
```

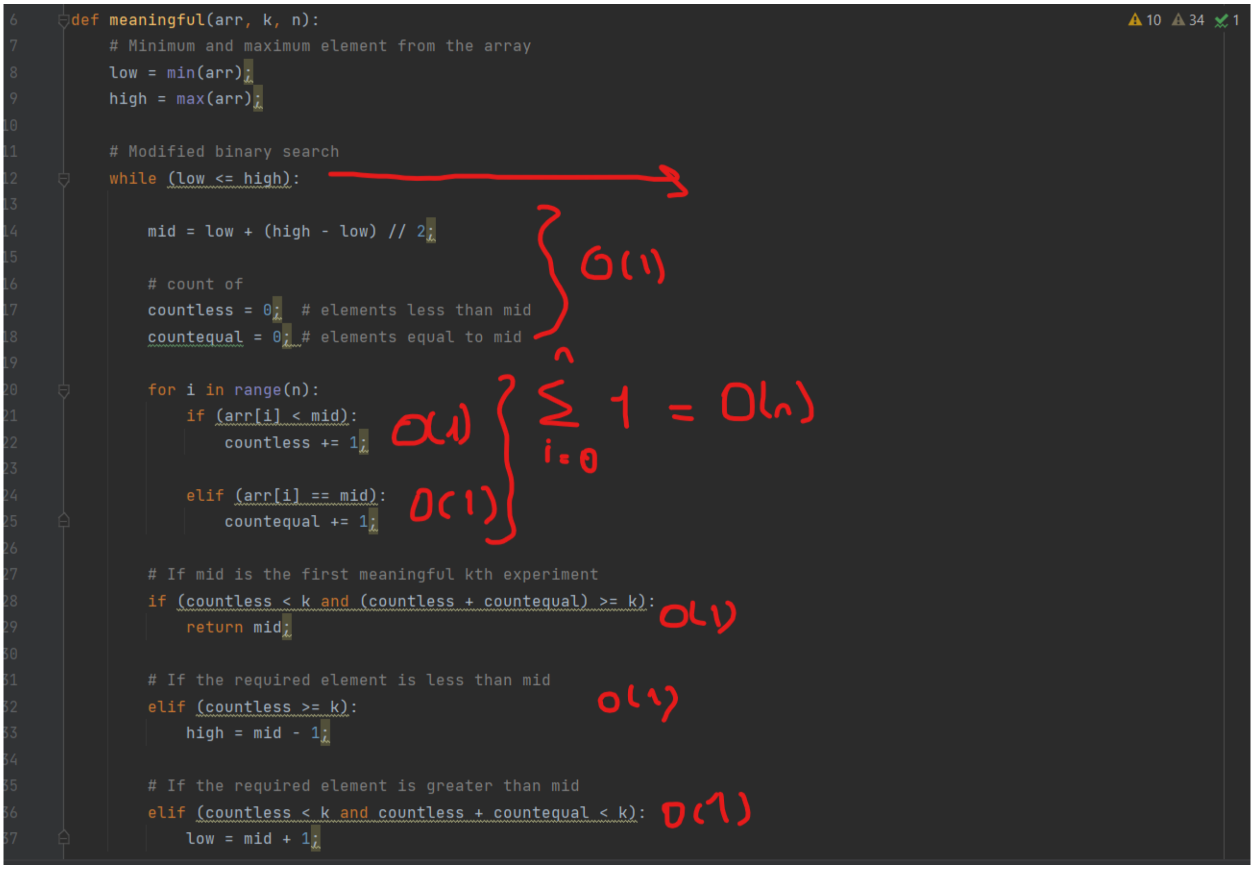
$T(n) = T(n/2) + T(n/2)$

$P(n) = P(n/2) + P(n/2)$

Time complexity of best() is => T(n) = 2T(n/2) and Time complexity of worst() is => P(n) = 2P(n/2):

With master theorem , $n^{\log_b a}$ = n^1 , so case 1 will be applied. $\Theta(n^{\log_b a}) = \Theta(n)$ is the time complexity of worst() and best() .

3)



```python
def meaningful(arr, k, n):
    # Minimum and maximum element from the array
    low = min(arr);
    high = max(arr);

    # Modified binary search
    while (low <= high):

        mid = low + (high - low) // 2;

        # count of
        countless = 0;  # elements less than mid
        countequal = 0; # elements equal to mid

        for i in range(n):
            if (arr[i] < mid):
                countless += 1;

            elif (arr[i] == mid):
                countequal += 1;

        # If mid is the first meaningful kth experiment
        if (countless < k and (countless + countequal) >= k):
            return mid;

        # If the required element is less than mid
        elif (countless >= k):
            high = mid - 1;

        # If the required element is greater than mid
        elif (countless < k and countless + countequal < k):
            low = mid + 1;
```

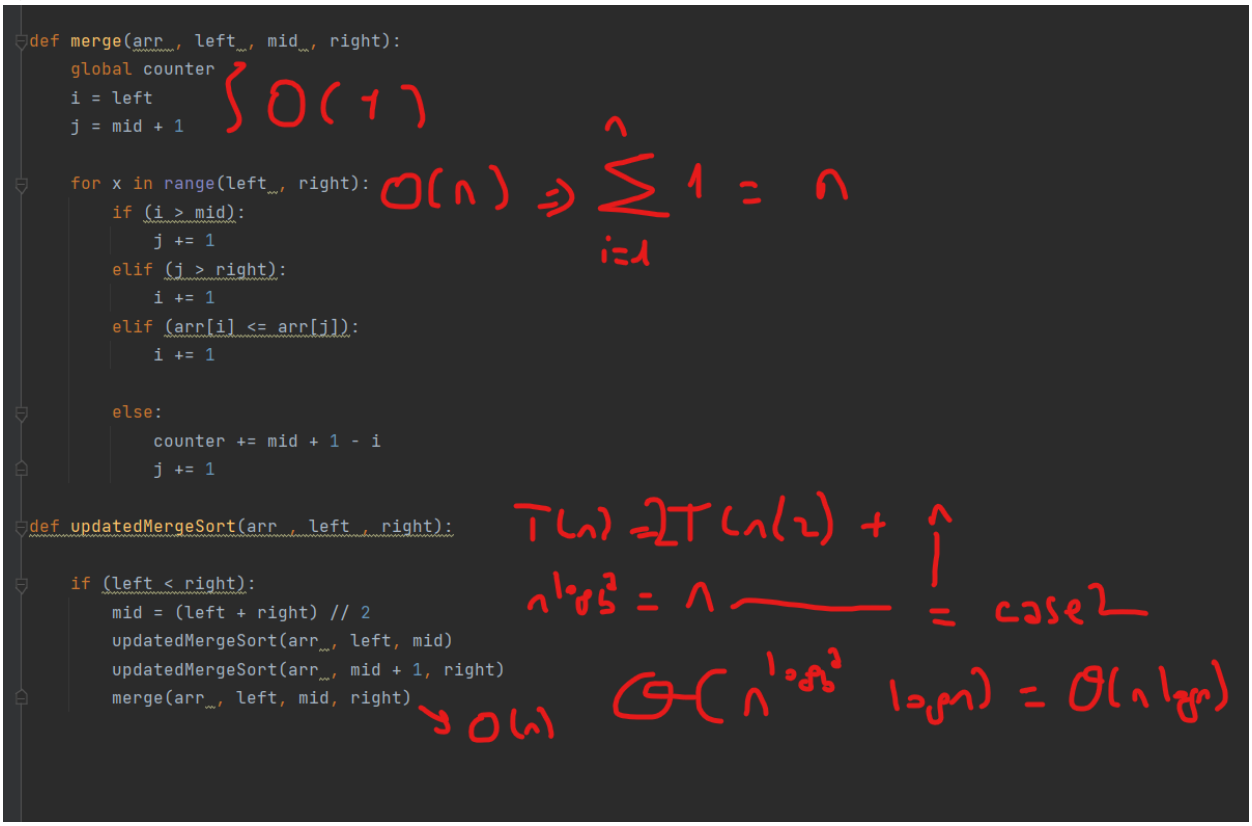

Worst case of while loop is k = n and best case of while loop is k = 0

Worst case time complexity = while loop will loop for logn times, so O(logn) * O(n) = O(nlogn)

Best case time complexity = while loop won't loop , so O(1) * O(n) = O(n)

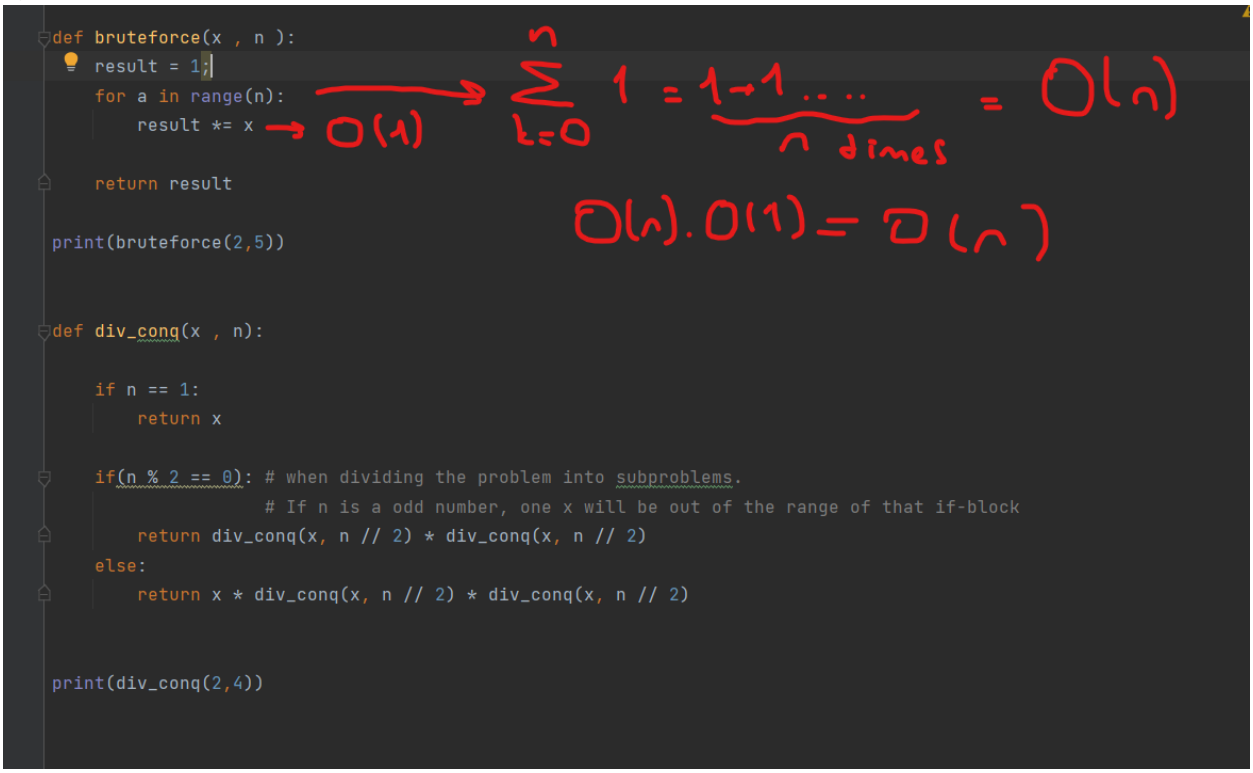
Avarage case time complexity depends the maximum and minimum value of items in the array, so avarage time complexity would be O(n log (arr[max] – arr[min])).

4)



```python
def merge(arr_, left_, mid_, right):
    global counter
    i = left
    j = mid + 1

    for x in range(left_, right):
        if (i > mid):
            j += 1
        elif (j > right):
            i += 1
        elif (arr[i] <= arr[j]):
            i += 1

        else:
            counter += mid + 1 - i
            j += 1

def updatedMergeSort(arr_, left_, right):

    if (left < right):
        mid = (left + right) // 2
        updatedMergeSort(arr_, left, mid)
        updatedMergeSort(arr_, mid + 1, right)
        merge(arr_, left, mid, right)
```

5)



```
def bruteforce(x , n ):
    result = 1;
    for a in range(n):
        result *= x

    return result

print(bruteforce(2,5))


def div_conq(x , n):

    if n == 1:
        return x

    if(n % 2 == 0): # when dividing the problem into subproblems.
                    # If n is a odd number, one x will be out of the range of that if-block
        return div_conq(x, n // 2) * div_conq(x, n // 2)
    else:
        return x * div_conq(x, n // 2) * div_conq(x, n // 2)


print(div_conq(2,4))
```


For div_conq(x, n) function:

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

With master theorem , $n^{\wedge} \log_b a$ = n^1 , so case 1 will be applied. Θ($n^{\wedge} \log_b a$) = Θ(n) is the time complexity of div_conq().