1)

```python
def cutting(wire_len):
    if(wire_len < 2):
        return 0

    if(wire_len == 2):
        return 1

    return cutting(math.ceil(wire_len/2))+1
```

Recurrence relation is **T (n) = T(n/2) + 1**

a = 1, b=2, d=0

T(n) € Θ(log(n))

2)

```python
def worst_best(arr):
    if len(arr) == 0:
        return None, None

    if len(arr) == 1:
        return arr[0], arr[0]

    if len(arr) == 2:
        e_min = arr[0] if arr[0] < arr[1] else arr[1]
        e_max = arr[0] if arr[0] > arr[1] else arr[1]

        return e_min, e_max

    e_min1, e_max1 = worst_best(arr[0 : math.floor(len(arr)/2)])
    e_min2, e_max2 = worst_best(arr[math.ceil(len(arr)/2) : len(arr)])

    e_min = e_min1 if e_min1 < e_min2 else e_min2
    e_max = e_max1 if e_max1 > e_max2 else e_max2

    return e_min, e_max
```

Recurrence relation is **T(n) = 2T(n/2)+1**

a = 2, b=2, d=0

T(n) € Θ(n)

3)

```
def lamuto_partition(arr):

    pivot = arr[0]
    small = 0
    for i in range(len(arr)):
        if arr[i] < pivot:                O(n)
            small = small + 1
            swap(arr, small, i)

    swap(arr, 0, small)
    return small


def swap(arr, a, b):

    temp = arr[a]        O(1)
    arr[a] = arr[b]
    arr[b] = temp


def meaningful(arr, k):

    s = lamuto_partition(arr)     O(n)
    if s == k-1:
        return arr[s]
    elif s > k-1:
        return meaningful(arr[:s], k)
    else:                                       T(n/2)
        return meaningful(arr[s+1:], k-s-1)
```

Recurrence relation, **T(n) = T(n/2)+n**

a = 1, b=2, d=1

Complexity = n + log(n) = n

T(n) € Θ(n)

4)

```python
def merge_sort_and_count(arr):

    if len(arr) == 1:
        return 0, arr
    else:
        mid = len(arr)//2
        r_a, a = merge_sort_and_count(arr[:mid])
        r_b, b = merge_sort_and_count(arr[mid:])
        r_m, m = merge_and_count(a, b)
        return r_m+r_a+r_b, m
```

O(n log(n))

O(logn)

O(n)

```python
def merge_and_count(arr1, arr2):

    reverse_order = 0
    i = 0
    j = 0
    res = []
    while i < len(arr1) and j < len(arr2):

        if arr1[i] <= arr2[j] :
            res.append(arr1[i])
            i = i+1
        else :
            res.append(arr2[j])
            reverse_order = reverse_order+len(arr1)-i
            j = j+1

    while i < len(arr1):
        res.append(arr1[i])
        i = i+1

    while j < len(arr2):
        res.append(arr2[j])
        j = j+1

    return reverse_order, res
```

O(n)

Recurrence relation is the same with merge sort, the only difference is we are also counting in constant time complexity.

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

a = 2, b=2, d=1

T(n) € Θ(n log(n))

5)

```
def exp_bf(a, n):
    if a <= 0:
        return 0
    elif n == 0:            O(1)
        return 1

    res = 1
    for i in range(n):            O(n)
        res = res*a

    return res


def exp_dq(a, n):

    if a == 0:
        return 0

    if n == 0:
        return 1            O(1)

    if n == 1:
        return a

    return exp_dq(a, n//2) * exp_dq(a, n-(n//2))        2T(n/2)
```

For exp_bf is O(n)

For exp_dq = $T(n) = 2T(n/2)$
$a = 1, b=2, d=0$
$T(n)$ € $\Theta(\log(n))$