1)

```
def min_time_to_cut(N,a):
    count
    if(N<=1) -> T2(n)
        return count
    N=N-a -> T(1)
    a*=2 -> T(1)
    count+=1 T(1)
    return min_time_to_cut(N,a) -> T1(n)
```

The aim of algorithm that I design is to decrease a number that is given and minimize it while doing this decreasing approach I try to find the cutting number.

$T(n) = T_1(n) + T_2(n)$

$T(n) = T(n/2) + 1$

↓

Because of a is increasing 2 times at every rec part. using Master Theorem I found the complexity

$T(n) = T(n/2) + n^0 \cdot 1 \rightarrow a=1 \quad b=2 \quad d=0$

$a = b^d$

$1 = 2^0$

$1=1 \Rightarrow T(n) = \Theta(n^0 \cdot \log n)$

$= \Theta(\log n)$ //

2) Because of the code is long I did not write here.

The aim of algorithm is. Find worst and best case if the list contains only one element worst and best case are same and I return it. If list contains two element only I found which one is best then I return them. But if has more than two elemant I found middle element and using recursion I found best and worst case.

$T_1(n)$ = First rec calls.

$T_2(n)$ = Second rec calls.

$T(n) = T_1(n) + T_2(n) + 1$ → $T(n/2)$ → because of dividing 2 the array.

$T(n) = 2T(n/2) + 1$

↓

$a=2, b=2, d=0 \Rightarrow T(n) = \Theta(n^{\log_2 2})$

$T(n) = \Theta(n)$

3) Question is wanted a algorithm that find ifirst kth meaningful experiment. So it occured to me to use the quick select algorithm directly it is decrose and conquer also, so it makes our work easy, so I implement quick select to find kth meaningful experiment.

```
function T_meaningful (list, left, right, k)
    if left=right
      return list[left]
    pivotIndex := partition (list, left, right, pivotIndex)   T1(n)
    if k=pivotIndex
      return list[k]
    else if k < pivotIndex
      right:=pivotIndex-1
    else
      left=pivotIndex+1
```

Best case : 1st element is the solution. $T(n) = n+1 = \theta(n)$

Average case : $T(n) = T(n/2) + n+1 = \theta(n)$

Worst case : $\theta(n^2)$

A more sophisticated choice of the pivot leads to a complicated algorithm with $\theta(n)$ worst-case efficiency.

4) To find easily the number of reversed-ordered pairs i design merge and count algorithm because it is decide and conquer also. the pairs that need to be counted during the merge step. Therefore to get the total number of inversions. that needs to be added are the numbers of inversions in the left subarray right subarray and merge

Time complexity : $T(n) = T_1(n/2) + T_2(n/2) + n$ (merge)

$T(n) = 2T(n/2) + 1 \Rightarrow a=2, b=2, d=1$

$T(n) = \theta(n \log n)$ //

$T_1(n)$ = First rec call.

$T_2(n)$ = Second rec call.

Space Complexity : $O(n)$, Temporary array

5) function power

```
if. y:= 1
  return x  → T(1)
if. y = 0
  return 1  → T(1)
if( y%2=0 )
    temp = power (x, y/2) → T1(n)
    return temp * temp
else
    temp = power (x,(y-1)/2) → T2(n)
    return x * temp * temp
```

$T(n) = T1(n) + T2(n) + 1$

$T(n) = 2T(n/2) + 1$

$a=2 \quad b=2 \quad d=0$

$T(n) = \Theta(n)$ //