There is an n-meter-long steel wire and it is needed to be cut into 1-meter-long pieces. There is a mechanic who asks money for every cutting. The machine that the mechanic uses allows multiple pieces to be cut at the same time. Before going to the mechanics, you want to calculate the minimum number of cuts that are needed to cut several pieces at the same time. Design a decrease-and-conquer algorithm that gives **the minimum number of cuts** needed.

After trying different n values on paper, I saw that this problem basically asks for log2 function. If n is a power of 2, answer is log(n), otherwise it is ceil(log(n)).

Algorithm:

```
def cut(n):
    if(n==1):
        return 0
    return cut(math.ceil(n/2))+1
```

$T(n) = T(n/2) + 1$ (Decrease-by-const-factor)

$T(1) = 1$

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

$= T(n/4) + 1 + 1$

$= T(n/2^k) + \underbrace{1+1+1\ldots}$

for $2^k = n$ $\quad \# = k$

$k = \log_2 n \Rightarrow T(n) = T(1) + \underbrace{1+1+1\ldots}_{\# = \log_2 n} = \boxed{T(n) = \log_2 n + 1}$

Proof by Induction:

$T(1) = 1$

$T(2) = T(1) + 1 \overset{?}{=} \log_2 2 + 1$ ✓ True for $n=2$

$T(n/2) = \log_2 n/2 + 1$ (Assume True)

$T(n) = T(n/2) + 1 = \log_2 n/2 + 1 + 1 =$

$= \log_2 n - \log_2 2 + 2 = \log_2 n + 1$ ✓ True for general n

So, $T(n) = \Theta(\log n)$

2. In a science laboratory, n experiments were performed to generate a new vaccine, and the success rates (size of n) were saved. A scientist from the laboratory will write an article about the outcome of the experiments, and she needs **the worst** and **the best** results. Design a divide-and-conquer algorithm to help her.

First return value is min, second return value is max.

```
def minMax(start,end,a):
    if(start==end): --> 1 element left
        return a[start],a[start]
    elif end == start+1: --> 2 element left
        if a[start] > a[end]:
            return a[end],a[start]
        else:
            return a[start],a[end]
    else:
        min1,max1 = minMax(start,floor((start+end)/2),a)
        min2,max2 = minMax(floor((start+end)/2)+1,end,a)
        return min(min1,min2),max(max1,max2)
```

const. time

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

Master Theorem

$f(n) = 1 = O(n^{\log_2 2 - \varepsilon}) \Rightarrow T(n) = \Theta(n)$

3. Regarding the experiments mentioned in the previous question, another scientist from the laboratory noticed that the first *k-1* number of experiments were meaningless when we sort the experiments in terms of the success rates in increasing order. He wants to know **the success rate** of the first meaningful k^{th} experiment. Design a decrease-and-conquer algorithm to help him without sorting the success rates of the experiments.

I used the QuickSelect approach which we learned in the lesson

```
def findKthSmallest(arr, l, r, k):

    if(k<0 or k > r-l+1):
        return -1

    pos = partition(arr, l, r)
    if (pos - l == k - 1): #if pos is kth smallest
        return arr[pos]
    elif (pos - l > k - 1): #if kth smallest is on left of pos
        return findKthSmallest(arr, l, pos - 1, k)
    else: #if kth smallest is on right of posA
        return findKthSmallest(arr, pos + 1, r, k - pos + l - 1)
    return -1
```

→ Partition alghorithm from Quicksort $\Theta(n)$

→ Base Case $\Theta(1)$

Decrease by a constant factor.

$T(n) = T(n-k) + \Theta(n)$

↳ Depends on input

Worst case: $\Theta(n^2)$ → Pivot is first element, array is sorted and $k = n-1$

Best case: $\Theta(n)$ first pivot is kth element

4. In a given array containing n real numbers $A[1 \ldots n]$, the pair $(A[i], A[j])$ is called reverse-ordered pair if $i < j$ and $A[i] > A[j]$. The more the number of reverse-ordered pairs the sequence has, the further from being sorted the sequence is.

In a military location, they receive information as number sequences (with size n) sorted in increasing order from a special telecommunication device. If there are reverse-ordered pairs in the sequence, that means the information is corrupted. Since security is crucial for this operation, they want to know how corrupted the information is. Design a divide-and-conquer algorithm that finds **the number of reverse-ordered pairs** in the received sequence to help them.

I remember reverse-ordered pairs are called inversions from the class and the number of inversions can be found by using merge sort. So I used the merge sort algorithm and added a counter. Everytime an element is picked from the right half (it means there is a smaller element on right half) counter is increased by one.

```
def mergeSort(arr):
    global count
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        mergeSort(L)
        mergeSort(R)

        #merge part:
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
                count+=1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

$\sum_{i=1}^{n} 1 = \Theta(n)$

Recurrence Relation:

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

Master Theorem

$f(n) = n = O(n^{\log_2 2}) \Rightarrow T(n) = \Theta(n \log n)$

5. Design a brute-force algorithm and a divide-and-conquer algorithm to solve the exponentiation problem, which is to compute a^n, where $a > 0$ and n is a positive integer.

```
def powerBrute(x,y):
    res=1
    while(y>0):
        res*=x
        y-=1
    return res
```

$\Theta(1)$

$$\sum_{i=1}^{y} 1 = \Theta(y)$$

```
def powerDQ(x, y):

    if (y == 0): return 1 #base case

    elif (y % 2 == 0): #x^4 = x^(4/2) * x^(4/2)
        return (power(x, y/2) * power(x, y/2))
    else: #x^5 = x * x^2 * x^2
        return (x * power(x, floor(y/2)) * power(x, floor(y/2)))
```

→ one of two is picked at every iteration so problem is divided into 2, not 4.

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

$= 4T(n/4) + 1 + 1$

$\vdots$

$2^k T(n/2^k) + \underbrace{1 + 1 \ldots + 1}_{\# = k}$

for $n = 2^k$, $k = \log_2 n$

$T(n) = n \cdot T(1) + \log_2 n = \Theta(n)$

Proof by Induction:

$T(1) = 1$

✓ $T(2) = 2 + 1 = 3 = 2 \cdot 1 + \log_2 2$

Assume $T(n/2) = \frac{n}{2} \cdot 1 + \log_2(n/2)$

$\rightarrow T(n) = 2(n/2 + \log_2(n/2)) + 1$

$= n + \log_2 n$ ✓