CSE 321 ALGORITHM HW4

YAKUP TALHA YOLCU 1801042609

1. Cutting wires

```python
def takelog(n):
    if (n<=2):
        return 1
    else:
        return takelog(n/2) + 1

def cut(n):
    if(n<=1):
        return 0
    return takelog(n)
```

In this method I am taking log2 of the length of wire. If number is less than 3 then log if 1

Otherwise I add 1 to symbolize the cut operation

Firstly I take length of the wire. I check the wire length is less than 2, if it is less than 2, this means that we don't need to cut anything. There is no cut operation. Return 0

Otherwise I take log of the length of the wire and I add 1 at each step

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

By master theorem a=1 b=2

f(n)=1 => f(n) is element of the theta(n^d) class. D=0. So n^0=1.

B^d=1 a=b^d. => 1=2^0 => 1=1. So T(n)= theta(n^d * logn) => T(n)=theta(n^0 * logn) => theta(logn)

Test outputs:

```
Expected cut: 0
n=1, Calculated cuts:  0
----------
Expected cut: 1
n=2, Calculated cuts:  1
----------
Expected cut: 2
n=3, Calculated cuts:  2
----------
Expected cut: 2
n=4, Calculated cuts:  2
----------
```

```
Expected cut: 4
n=9, Calculated cuts:  4
----------
Expected cut: 4
n=10, Calculated cuts:  4
----------
Expected cut: 7
n=100, Calculated cuts:  7
----------
Expected cut: 5
n=18, Calculated cuts:  5
----------
Expected cut: 5
n=20, Calculated cuts:  5
```

1) $T(1)=1$ $T(2)=1$ $\qquad n/2^k=1$

$T(n)=T(n/2)+1$ $\qquad n=2^k$

$T(n/2)=T(n/4)+1$ $\qquad \boxed{k=\log n}$

$T(n)=T(n/4)+2$

$T(n)=T(n/2^k)+k \Rightarrow T(n)=T(1)+k$

1) $T(1)=1 \Rightarrow$ Induction $\Rightarrow$ $\qquad T(n)=1+k$

$T(1)=1$ ✓ $\qquad k=\log n$ $\boxed{T(n)=1+\log n}$ $=1+\log 1=1$

2) Assume $T(n/2)=1+\log n/2$ ✓

3) Try for $T(n) \Rightarrow T(n)=T(n/2)+1 \stackrel{?}{=} 1+\log n$

$T(n)=1+\log n/2+1 \stackrel{?}{=} 1+\log n$

$=2+\log n/2 \stackrel{?}{=} 1+\log n$

$2+\log n/2=2+\log n-\log 2$ $\qquad 1+\log n=1+\log n$ ✓

$=1+\log n$ $\qquad$ Proved

$T(n)=T(n/2)+1$ $\qquad \boxed{T(n)=1+\log n} \Rightarrow \in \Theta(\log n)$

remove constants

2. Worst best

```python
def find_worst_best(succes_rates):
    if(len(succes_rates)<2):
        return succes_rates[0],succes_rates[0]

    mid=int(len(succes_rates)/2)
    left_half_worst,left_half_best=find_worst_best(succes_rates[0:mid])
    right_half_worst,right_half_best=find_worst_best(succes_rates[mid:n])
    return min(left_half_worst,right_half_worst),max(left_half_best,right_half_best)
```

I take success rates as an argument. Firstly I check the length of the array, if length is less 2 this means that min and max is zeroth element of the array. Then I found middle element index of the array. I divide length by 2. Then continuously I find left half worst(min) left half best(max) and right half worst(min) right half best(max) of the array by using recursive. Then I compare the worst ones and best ones. Then I return the results.

T(n)=2T(n/2)+1(finding mid)

By master theorem a=2 b=2

f(n)=1 => f(n) is element of the theta(n^d) class. D=0. So n^0=1.

B^d=1 a>b^d. => 2>2^0 => 2>1. So T(n)= theta($n^{\log_b a}$) => T(n)=theta($n^{\log_2 2}$) => theta(n)

Test Outputs:

```
Input [20, 25, 97, 10, 15, 0, 99]
Expected worst:  0  Expected best:  99
Calculated worst:  0 Calculated Best:  99
---------
Input [20, 25, 97, 10, 15]
Expected worst:  10  Expected best:  97
Calculated worst:  10 Calculated Best:  97
---------
Input [2, 6, 7, 15, 9, 10]
Expected worst:  2  Expected best:  15
Calculated worst:  2 Calculated Best:  15
---------
Input [1, 2, 20, 6, 5, 15, 9, 10]
Expected worst:  1  Expected best:  20
Calculated worst:  1 Calculated Best:  20
```

(2) $T(n) = 2\,T(n/2) + 1$ → finding mid index

left, right call · dividing into 2

$T(1) = T(0) = 1$ base case

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

$T(n/2) = 2T(n/4)+1$

$T(n) = 2(2T(n/4)+1)+1$

$n/2^k = 1$

$k = \log n$

$T(n) = 2^k T(n/2^k) + 2^k - 1$

$T(n) = 2^{\log n} T(1) + 2^{\log n} - 1$

$= n + n - 1 \in \Theta(n)$

$= 2n - 1$

(1) for $n=1 \Rightarrow T(1)=1$ $\quad 2.n-1 \Rightarrow 2.1-1=1$ ✓ proved

(2) for $T(n/2) \Rightarrow 2(n/2)-1 \Rightarrow n-1$ assume

(3) for $T(n) \Rightarrow 2T(n/2)+1 \stackrel{?}{=} 2n-1$

$\Rightarrow 2(n-1)+1 \stackrel{?}{=} 2n-1 \Rightarrow 2n-2+1 \stackrel{?}{=} 2n-1$

$= 2n-1 = 2n-1$ ✓ proved

$T(n) = 2n-1 \in \Theta(n)$

remove constants

3. Find kth meaningful

```python
def decrease(rates):
    min_v=min(rates)
    for i in range(len(rates)):
        if(rates[i]==min_v):
            rates.pop(i)
            return

def find_kth_meaningful(rates,k):
    for i in range(k-1):
        decrease(rates)
    return min(rates)
```

In this function I take success rates as an argument and I found the min value of the input array. Then I find the index of that value. Then I remove it from array because it is meaningless for our experiment

I take success rates and number k as an argument. Firstly I iterate until k. I decrease the input size by 1 at each iteration. At the end I remove k-1 elements from array. Then I returned the min value of the array. Which is first meaningless value.

$$\left(\sum_{i=0}^{k-1}\left(\sum_{j=0}^{n-1} 1\right) + \left(\sum_{i=0}^{n-1} 1\right)\right) = \left(\sum_{i=0}^{k-1}\left(\sum_{i=0}^{n-1} 1\right) + n\right)$$

$$\left(\sum_{i=0}^{k-1} n + n\right) = 2nk \in \Theta(n*k)$$

$$T(n,k) = \Theta(n*k)$$

Test Outputs:

```
Input array:  [20, 25, 97, 10, 15, 0, 99]
Expected 5th smallest: 25
Calculated 5th smallest:  25
---------
Input array:  [20, 25, 97, 10, 15]
Expected 3rd smallest: 20
Calculated 3rd smallest:  20
---------
Input array:  [7, 10, 4, 3, 20, 15]
Expected 2nd smallest: 4
Calculated 3rd smallest:  4

Process finished with exit code 0
```

4. Reverse ordered pairs

I used merge sort algo to find the reverse ordered pairs.
I just find the reverse ordered pairs when the element of the right sub array is copied to the merged array. I increased the counter at that time. This means that we have find the reverse ordered pair.

```
while i < n1 and j < n2:
    if LEFT[i] <= RIGHT[j]:
        arr[k] = LEFT[i]
        i += 1
    else:
        arr[k] = RIGHT[j]
        count += (n1 - i)
        j += 1
    k += 1
```

Complexity of the merge sort is theta (n*logn)

<=

```
Input array [20, 1, 2, 6, 7, 15, 9, 10]
Expected number of rop 9:
Calculated number of rop:  9
---------
Input array [20, 25, 97, 10, 15, 0, 99]
Expected number of rop 11:
Calculated number of rop:  11
---------
Input array [1, 2, 20, 6, 7, 15, 9, 10]
Expected number of rop 7:
Calculated number of rop:  7
---------
Input array [1, 2, 20, 6, 5, 15, 9, 10]
Expected number of rop 8:
Calculated number of rop:  8
```

5. Exponential Problem

```python
def exponential_brute_force(a,n):
    res=1
    for i in range(n):
        res=res*a
    return res

def exponential_divide_and_conq(a,n):
    if(n==0):
        return 1
    elif (n%2==0): return exponential_divide_and_conq(a,n/2) * exponential_divide_and_conq(a,n/2)
    else : return a * exponential_divide_and_conq(a,n-1)
```

In brute force algorithm I just used for loop and each iteration I just multiplied the numbers.

In Divide and Conquer algorithm firstly I checked the if the exponent is zero or not then if it is zero I returned 1. Otherwise I take mod of the exponent to divide the problem into 2.

If it can't be divided into two then I take number and I decreased the number by 1.

For Brute Force Algorithm

T(n)=sum base=> I=0 ceil=> n-1 inside 1 => theta(n)

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

By master theorem a=2 b=2

f(n)=1 => f(n) is element of the theta(n^d) class. D=0. So n^0=1.

B^d=1 a>b^d. => 2>2^0 => 2>1. So T(n)= theta($n^{\log_b a}$) => T(n)=theta($n^{\log_2 2}$) => theta(n)

```
2^5 BRUTE FORCE:  32
2^5 DIVIDE AND CONQUER:  32
---------------
---------------
5^3 BRUTE FORCE:  125
5^3 DIVIDE AND CONQUER:  125
---------------
```

(3) Brute force => $\sum_{i=0}^{n-1} 1 = n$

Divide and conquer => $T(n)=2T(n/2)+1$ multiplication $T(1)=1$

For Divide and conquer algo

② $T(n) = 2\,T(n/2) + 1$ → Multiplying

left&right call — dividing into 2

$T(1) = T(0) = 1$ base case

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

$T(n/2) = 2\,T(n/4) + 1$

$T(n) = 2(2\,T(n/4) + 1) + 1$

$n/2^k = 1$

$k = \log n$

$T(n) = 2^k\,T(n/2^k) + 2^k - 1$

$T(n) = 2^{\log n}\,T(1) + 2^{\log n} - 1$

$= n + n - 1 \in \Theta(n)$

$= 2n - 1$

① for $n = 1 \Rightarrow T(1) = 1$ $\quad 2.n - 1 \Rightarrow 2.1 - 1 = 1$ ✓ proved

② for $T(n/2) \Rightarrow 2(n/2) - 1 \Rightarrow n - 1$ assume

③ for $T(n) \Rightarrow 2\,T(n/2) + 1 \stackrel{?}{=} 2n - 1$

$\Rightarrow 2(n-1) + 1 \stackrel{?}{=} 2n - 1 \quad \Rightarrow 2n - 2 + 1 \stackrel{?}{=} 2n - 1$

$= 2n - 1 = 2n - 1$ ✓ Proved

$T(n) = 2n - 1 \in \Theta(n)$

remove constants