CSE 321 - Introduction to Algorithm Design

Homework 4

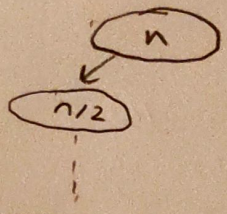
180104210 3
Ozan GEÇKİN

Q1) Explain the algorithm = I wrote my algorithm recursive I set a length first I am sending this to my recursive function If the base case length of my function is less than 1 meter, if it is not in the base case I divide by two, increase my counter by one, so that I can find the minimum number of cuts for the length I have given. For example, 12 meters steel is divided by 6,6. 11 meters steel is divided by 5,6. After that we divide resultent steels to two again, after one under the other. I made the algorithm design decrease-and-conquer appropriate.

Analyze =

```
def cutting(length, counter)
    if length > 1
        counter += 1
        return cutting(num/2, counter) → T(n) = T(n/2)
    else
        return count  ⟶ T(1) = 1
```

Not = Decrease_and_conquer
Reduction by a fixed factor



$$T(n) = \begin{cases} T(1) = 1 \\ T(n) = T(n/2) + 1 \end{cases}$$

Using Master Theorem

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

$a = 1 \quad b = 2 \quad f(n) = 1$

$n^{\log_b a} = n^0 = 1$

$f(n) = n^0 = 1$

$T(n) = \Theta(n^{\log_b a} \cdot \log n) =$

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

Proof by Induction

1) Base case $\Rightarrow T(1) = 1$

2) Assume $\Rightarrow T(n/2) = T(\frac{n}{2^2}) + 1$

$T(n/2) = \log \frac{n}{2}$

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

$= \log \frac{n}{2} + 1 = \log n$

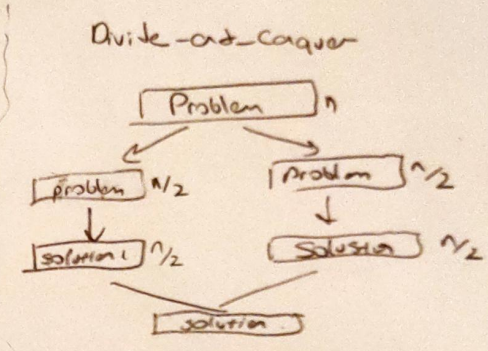
$= \log n - \log_2^2 + 1 = \log n$

$= \log n - 1 + 1 = \log n$

$= \log n = \log n$ ✓

1201042103
Ozan GECKIN

Q2) Explain the algorithm=I wrote the algorithm as recursive. I followed the Divide-and-Conquer algorithm design. My findWorstBest function takes parameter arr, left, right, min and max values. If my base case left and right are equal then I compare, sort and return. If the difference of the other base case right is 1, I sort again and exit. Then I divide the length of arr by 2. I am running my function again. Thus, I can reach the divide-and-conquer algorithm design.




```
def findWorstBest(arr, left, right, min, max)
    if left == right:
        return min, max          } O(1)
    if right - left == 1:
        return min, max          } O(1)
    mid = (left + right) // 2
    min, max = findWorstBest(arr, left, mid, min, max)   -> T(n/2)
    min, max = findWorstBest(arr, mid+1, right, min, max) -> T(n/2)
    return min, max
```

Analyze

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

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

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

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

$T(n) = 2^k T\left(\frac{n}{2^k}\right)$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$

$k = \log n$

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

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

Proof by induction

1) Base $T(1) = 1$

2) Assume $T(n/2) = \frac{n}{2}$

3) $T(n) = 2T\left(\frac{n}{2}\right)$

$2\frac{n}{2} = n$ ✓

1801062103
Ozan GEÇKİN

Q3) I used quick sort, but my goal is not to complete the quick sort, but to stop at the kth smallest item.
Worst case of the algorithm is n recursive calls, with kth being the largest element

```
def partition(arr, left, right)
    :
  for j in range(left, right):
      :
      :
  return

def meaningful(arr, left, right, k):
  if ____ ____
    index = partition(arr, left, right)  ⟶ n
    if ____ ____
      return
    if ____ ____
      return meaningful(arr, left, index-1, k) → T(n-1)
    return meaningful(arr, index+1, right, k-index+left-1)
```

$T(n) = T(n-1) + n$
$T(1) = 1$ ⟶ Worst case = $T(n) = O(n^2)$

backward substitution

$T(n) = T(n-1) + n$
$= T((n-1)-1) + (n-1) + n$
$= T(n-2) + (n-1) + n$
$= T(n-2-1) + (n-2) + (n-1) + n$
$= T(n-3) + (n-2) + (n-1) + n$
:
$= T(1) + 2 + 3 \text{ ---- } + n$
$= \frac{n(n+1)}{2}$

$T(n) = \frac{n(n+1)}{2} = \frac{n^2+n}{2} = O(n^2)$

Prof by induction

1) Base
$T(1) = 1$
$T(n) = \frac{n(n+1)}{2} = \frac{1(1+1)}{2} = 1 = T(1)$

2) Assume $T(n-1)$ is true
$T(n-1) = (n-1)(n-1+1)/2$

3) $T(n) = T(n-1) + n$
$= \frac{(n-1)n}{2} + n = \frac{n^2-n}{2} + n$
$= \frac{n(n+1)}{2}$ ✓

1801042103
Ozan GEÇKİN

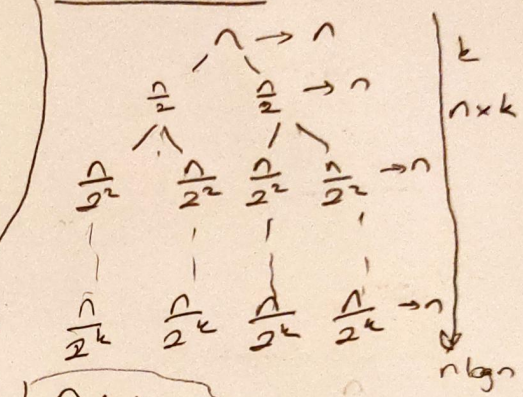
Q4) I used merge sort to find the number of reverse-ordered pairs. I followed the divide and conquer algorithm designing. While Merge was sorting, I kept a counter to find the number of reversed-ordered pairs.

Analyz

```
def find_rop(arr,n):
    return mergeSort(arr,tempArr,0,n-1)
def mergeSort(arr,tempArr,left,right):
    if left<right:
        mid=(left+right)//2
        count+=mergeSort(...)
        count+=mergeSort(...)
        count+=merge(...)
def merge(arr,tempArr,left,mid,right):
    while ___:
        while ___:
            while ___:
                for ___:
    return count
```

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

Tree method.



$\Theta(n \log n)$

Backward sub.

$T(n) = 2T(\frac{n}{2}) + n$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}$

$T(n) = 2(2T(\frac{n}{4}) + \frac{n}{2}) + n$

$T(n) = 2^k T(\frac{n}{2^k}) + kn$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log n$

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

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

Prove by induction $T(n) = 2T(\frac{n}{2}) + n$, $T(n) = n\log n + n$

1) Base case

$T(1) = 1$

$T(1) = 1.\log 1 + 1 = 0 + 1 = 1$

2) Assume

$T(\frac{n}{2}) = \frac{n}{2}\log\frac{n}{2} + \frac{n}{2}$

3) $T(n) = 2T(\frac{n}{2}) + n \stackrel{?}{=} n\log n + n$

$T(n) = 2\left(\frac{n}{2}\log\frac{n}{2} + \frac{n}{2}\right) + n = n\log n + n$

$T(n) = n\log\frac{n}{2} + n + n = n\log n + n$

$n(\log n - \log 2) + n + n = n\log n + n$ ✓

Q5] In this question, I wrote two functions, it does exponentiation in both functions. I made one suitable for the iterative brute-force algorithm design. I made my other function according to the recursive divide-and-conquer algorithm.

180104-2103
Ozan GEÇKİN
Q2

Analyze

```
def exponentiationBruce_force(base, exponent):
    res = 1
    for i in range(exponent):
        res *= base
    return res
```

$$\sum_{i=0}^{n} 1 \Rightarrow \Theta(n)$$

$T(n) \in \Theta(n)$

```
def exponentiationDivide_And_Conquer(base, exponent):
    if exponent == 0:
        return 1
    if exponent % 2 == 0:
        return exponentDivide(base*base, exponent/2)
    else:
        return base * exponentDivide(base*base, exponent/2)
```

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + 1 & n > 1 \end{cases}$$

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

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

$T(n) = T(\frac{n}{2^3}) + 1 + 1 + 1$

$T(n) = T\left(\frac{n}{2^k}\right) + k$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log n$

$T(n) = \underbrace{T(1)}_{1} + \log n = \Theta(\log n)$

Prove by Induction (Divide and conquer)

1) Base

$T(1) = 1$

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

$T(n) = 1 + \log n$

$T(n) = 1 + \log n = 1 + 0 = 1$

2) Assume

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

3) $T(n) = T(\frac{n}{2}) + 1$

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

$\log n - \underbrace{\log 2}_{-1} + 1 + 1 = \log n + 1$ ✓