HW-4

Can Duyar
171044075

Q1)

```
Algorithm minimumNumberofCuts (Nmeter , time ← 0 , Control ← 1)
  If(Nmeter <= 0)
    Print ("Minimum number of Cuts for steel wire is: ", time)
    return time
  endif

  Nmeter = Nmeter - control
  Control ← control * 2
  time ← time + 1
  Call minimumNumberofCuts (Nmeter , time , control)
end
```

Constant time operations.

Time Complexity Analysis

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

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

$T(n) = [T(n-4) + 1] + 1$

$T(n) = T(n-k) + 1 \cdot (\frac{k}{2})$

if we give n for k then

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

$T(n) = T(0) + \frac{n}{2}$

$T(n) = 0 + \frac{n}{2}$

$T(n) = \frac{n}{2} \longrightarrow \boxed{T(n) \in \Theta(n)} \rightarrow$ Result.

→ This algorithm finds minimum number of cuts for n-meter-long steel wire. It is needed to be cut into 1-meter-long pieces, based on decrease & conquer.

For example, if Nmeter = 100 then program gives 7 as result because,

(100 units) → (2 partitions of 50 units) → (4 partitions of 25 units) → (4 partitions of 12 units and 4 partitions of 13 units) → (12 partitions of 6 units and 4 partitions of 7 units) → (28 partitions of 3 units and 4 partitions of 4 units) → (28 partitions of 1 units and 36 part. of 2 units) → (100 partitions of 1 unit)

Q2) Algorithm worstBestResults (listSuccess, l, r, worst ← sys.maxsize, best ← -sys.maxsize)

```
    if (l==r)
        if worst > listSuccess[r]
            worst ← listSuccess[r]
        if best < listSuccess[l]
            best ← listSuccess[l]
        endif
        return worst,best.                          O(1)

    if r-l==1
      if listSuccess[l] >= listSuccess[r]
        if worst > listSuccess[r]
           worst ← listSuccess[r]
        if best < listSuccess[l]
           best ← listSuccess[l]                    O(1)
        endif
      else
          if worst > listSuccess[l]
             worst ← listSuccess[l]
          if best < listSuccess[r]
             best ← listSuccess[r]
          endif
      endif
      return worst,best
    endif                                    → Constant time operations O(1)
    mid ← (l+r) //2
    worst,best ← Call worstBestResults (listSuccess, l, mid, worst, best)
    worst,best ← Call worstBestResults (listSuccess, mid+1, r, worst, best)
    return worst,best
end
```

Time Complexity

$T(n) = 2T(n/2) + O(1)$ → constant time operations.

→ We can apply master theorem to solve this recurrence relation!

③

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

a, b, $f(n)$

$f(n) = \Theta(n^k \log^p n)$ $\rightarrow 1$

$n^k \log^p n = 1$
then $k=0$ and $p=0$

$\log_b^a = \log_2^2 = 1$

if we compare the values of k and $\log_b^a$ then we see this,

$\log_b^a > k$ so it's the first case!

Therefore Time Complexity is,

$$T(n) = \Theta(n^{\log_b^a}) = \Theta(n^1)$$

$\boxed{T(n) = \Theta(n)}$ → Result.

Note!

Master Theorem

$T(n) = a \cdot T(n/b) + f(n)$ $\quad a \geq 1$, $b > 1$ and

$f(n) = \Theta(n^k \log^p n)$ $\quad f(n)$ is asymptotically +

Case-1: if $\log_b^a > k$ then $\Theta(n^{\log_b^a})$

Case-2: if $\log_b^a = k$

i) if $p > -1 \rightarrow \Theta(n^k \log^{p+1} n)$

ii) if $p = -1 \rightarrow \Theta(n^k \log \log n)$

iii) if $p < -1 \rightarrow \Theta(n^k)$

Case-3:

if $\log_b^a < k$:

if $p \geq 0 \rightarrow \Theta(n^k \log^p n)$

if $p < 0 \rightarrow \Theta(n^k)$

→ This algorithm finds the best and worst results from the given array based on divide & conquer. The solution is to recursively divide the array into two equal parts and update the best and worst of the whole array in recursion by passing worst and best variables

Q3)

```
Algorithm meaningfulExperments (experments, left, right, kth)
    if (kth > 0 and kth <= right - left + 1)
        loc <- call helperMeaningful (experments, left, right)      -> Θ(n)
        if (loc - left > kth - 1)
            return meaningfulExperments (experments, left, loc-1, kth)

        if (loc - left == kth - 1)
            return experments [loc]
        endif
        return meaningfulExperments (experments, loc+1, right, kth - loc + left - 1)
    endif
    return sys.maxsize
end
```

```
Algorithm helperMeaningful (experments, left, right)
    rightKeep <- experments [right]   } -> constant time operations. Θ(1)
    temp <- left
    for t <- left (-> 0) to t <- right (-> n) do
        if (experiments [t] <= rightKeep)                                      constant time
            experments [temp], experments [t] <- experments [t], experments [temp]   operations Θ(1)
            temp <- temp + 1
        endif
    endfor
    experiments [temp], experiments [right] <- experiments [right], experments [temp]   -> constant time operations Θ(1)
    return temp
end
```

→ This Algorithm returns the success rate of the first meaningful k^{th} experment, based on decrease and conquer. This solution doesn't complete the sorting but stop at the point where pivot itself is k'th smallest success rate. Also not to recur for both left and right sides of pivot, but recur for one of them according to the position of pivot.

<u>Time Complexity Analysis</u>

For <u>helperMeaningful</u> function: → $T_2(n)$

$$T_2(n) = \sum_{t \leftarrow 0}^{n} 1 = \underbrace{1 + 1 \ldots + 1}_{n+1} = \underline{n+1} \rightsquigarrow \Theta(n)$$

$$\boxed{T_2(n) \in \Theta(n)}$$

For <u>meaningfulExperiments</u> function → $T_1(n)$

$$T_1(n) = 1 \cdot T(n/2) + n$$

($a = 1$, b, $f(n)$) This part comes from $\Theta(n)$ complexity of the helperMeaningful function. $T_2(n)$

→ we can apply master theorem to solve this recurrence relation! I used rules that I explained in page-3.

$\log_b^a = \log_2^1 = 0$

$f(n) \in \Theta(n^k \log^p n)$ → $k = 1$, $p = 0$ in this case

if we compare the values of $\log_b^a$ and k then we see this $k > \log_b^a$ so it's in the 3. case of the master theorem

if $p \geq 0$ then $\Theta(n^k \log^p n)$ so $\boxed{T(n) = \Theta(n)}$ Result.

Q4)

```
Algorithm countReversed (pairs, keepList, l, r)
    numberofinversions ← 0
    if l < r
        m ← (l+r)//2
        numberofinversions ← numberofinversions + call countReversed (pairs, keepList, l, m)
        numberofinversions ← numberofinversions + call countReversed (pairs, keepList, m+1, r)
        numberofinversions ← numberofinversions + call mergePairs (pairs, keepList, l, m, r)
    endif
    return numberofinversions
end
```

```
Algorithm mergePairs (pairs, keepList, l, m, r)
    x ← l
    y ← m+1
    k ← l
    numberofinversions ← 0
    while x <= m and y <= r do                          → constant time operations.
        if pairs[x] <= pairs[y]
            keepList[k] ← pairs[x]                      → O(1)
            k ← k+1
            x ← x+1
        else                                            → O(n)
            keepList[k] ← pairs[y]
            numberofinversions ← numberofinversions + (m-x+1)
            k ← k+1
            y ← y+1
        endif
    endwhile
    while x <= m do
        keepList[k] ← pairs[x]                          → O(1)
        k ← k+1
        x ← x+1
    endwhile                                            → O(n)
    while y <= r do
        keepList[k] ← pairs[y]                          O(1)
        k ← k+1
        y ← y+1
    endwhile
    for turn ← l to turn ← r+1 do                       → O(n)
        pairs[turn] ← keepList[turn]  → O(1)
    endfor
    return numberofinversions.
end
```

Time Complexity Analysis

Analysis of mergePairs function: $\rightarrow T_2(n)$

$$T_2(n) = \sum_{x \leftarrow 0}^{n/2} 1 + \sum_{y \leftarrow (n/2+1)}^{n} 1 + \sum_{tum \leftarrow 0}^{n} 1 = \left(\frac{n}{2}+1\right) + \left(n - \frac{n}{2}\right) + (n+1)$$

$$= 2n+2 \in \Theta(n)$$

$$\underline{T_2(n) \in \Theta(n)}$$

Analysis of countReversed function: $\rightarrow T_1(n)$

$$T_1(n) = 2T(n/2) + \underbrace{T_2(n)}_{\in \Theta(n)}$$

$$T_1(n) = \overset{a}{2}T(n/\overset{b}{2}) + \underbrace{n}_{f(n)}$$

$\rightarrow$ we can apply master theorem to solve this recurrence relation! I exclained cases of master theorem in Page-3.

$$\log_b^a = \log_2^2 = 1$$

we know that $f(n) \in \Theta(n^k \log^p n)$ so $\Theta(n^k \log^p n) \overset{?}{=} \Theta(n)$

$\underline{k=1}$ and $\underline{p=0}$ in this case.

if we compare the values of $\log_b^a$ and k then we can say that $\log_b^a = k = 1$ so it's the second case of the master theorem.

Now, we need to check the value of "p" to decide one of the subcases of case-2 of Master theorem. $p=0$ so $p>-1$ and it's the first case of the case-2!

$$\text{if } p>-1 \rightarrow \Theta(n^{\overset{1}{k}} \log^{\overset{0}{p}+1} n) = \boxed{\Theta(n \log n)} \rightarrow \text{Result.}$$

$\rightarrow$ This algorithm finds the number of reverse-ordered-pairs, solution is similar to merge sort, divide the array into two equal or almost equal halves in each step until the base case is reached. I created a recursive function to divide the array into halves and find the answer by summing the number of inversions is the first half, the number of inversions in the second half and the number of inversions by merging the two.

Q5)

```
Algorithm expoBF (a,n)
  bf ← 1
  for t ← 0 to t ← n do
     bf ← bf * a
  endfor
  return bf
end
```

→ constant time operation = $O(1)$

```
Algorithm expoDC (a,n)
  if (a==0)
     return 0
  if (n < 1)
     return 1
  if (n % 2 == 0)
     return expoDC (a*a , n/2)
  else
     return a* expoDC (a*a , n/2)
  endif
end
```

constant time operations → $O(1)$

Time complexity Analysis

Time complexity for expoBF (Brute force design)

$$T(n) = \sum_{t\leftarrow 0}^{n} 1 = \underbrace{1+1+\dots+1}_{n+1} = n+1 \rightarrow \Theta(n)$$

$\boxed{T(n) \in \Theta(n)}$ → Result.

Time complexity for expoDC (Divide and Conquer Design)

$T(n) = \overset{a}{1}T(n/\overset{b}{2}) + \overset{f(n)}{1}$ → we can apply master theorem to solve this recurrence relation!

$\log_b^a = \log_2^1 = 0$ $\quad f(n) \in \Theta(n^k \log^p n)$

$n^k \log^p n = 1$

$k=0$ and $p=0$ then

$\log_b^a = k$ so it's the second case of the master theorem, if we check the value of "p" then we see that $p > -1$ then we say that $T(n) \in \Theta(n^k \log^{p+1} n)$ according to first case of the second case of Master Theorem!

$T(n) \in \Theta(n^0 \log n)$

$\boxed{T(n) \in \Theta(\log n)}$ → Result.

Scanned with CamScanner

8

→ My algorithms are based on brute-force and divide & conquer, to solve the exponentiation problem, which is to compute a^n, where $a>0$ and n is a positive integer.

→ In brute force solution, it multiplies it with a itself, but in divide & conquer approach square a each time, rather than multiplying it with a itself.