161044071
Nevra GÜRSES

~ CSE 321 HOMEWORK 3 ~

1-) In my code, I use a helper function that does alternate operation recursively and in boxes function I call this helper function. I use helper function because boxes function takes array as parameter, but according to my algoritm, I use some other parameter variables for recursive calls.

For alternate black-white-black-white pattern, I use decrease and conquer algoritm. For each recursive call, I decrease boxes array size by 4 (2 from start point, 2 from end point) and also I make swap operation, so in every recursive call left and right side become bw-bw pattern.

For calculating time complexity:

If condition of my code there is only return statement and it is O(1) complexity. Swap operation in the else condition there is again constant time complexity → O(1). Recursive call operation is T(n-4). So recurrence relation is → $T(n) = T(n-4) + 1$

To solving these complexity:

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

$$T(n-4) = T(n-8) + 1 \rightarrow T(n) = T(n-8) + 2$$

$$T(n-8) = T(n-12) + 1 \rightarrow T(n) = T(n-12) + 3$$

$$\vdots$$

$$T(n) = T(n-4k) + k \rightarrow \text{Assume that } k = \frac{n}{4} \text{ and } T(0) = 0$$

$$T(n) = \underbrace{T(0)}_{0} + \frac{n}{4} \rightarrow \text{so } T(n) \in \Theta\left(\frac{n}{4}\right) \in \Theta(n)$$

There is only one case in this boxes alternate problem. Because black boxes is always first n box, white boxes is always remaing n boxes. So best case, average case and worst case is same and $\{T(n) \in \Theta(n)\}$

2-) In my code, I use a helper fakeCoin function that does all operations and in fakecoin function I call this helper fonction. I use helper function because fakeCoin function takes only array as parameter but in my algoritm, I use some other parameter variables for recursive calls.

For explaining finding fake coin algoritm:

In my algoritm, I divide coin array in 3 port. If sum of coins in first port equal to sum of coins in second port, I make a recursive call for third port because fake coin in this port. If sum of coins in first port less than second port, I make a recursive call for first port because fake coin in this part. If sum of coins in first port much more than second port, I make a recursive call for second port because fake coin in that port. I make above operations if size of coins more than 2. If size of coins 2 and first coin amount less than other, it is fake coin, if size of coins 2 and second coin amount less that other it is fake coin. The terminating condition of my algoritm is size==1. Because every recursive call I divide coin size by 3, if size==1 I return fake coin

My algoritm is decrease and conquer algoritm (variation is decrease by a constant factor)

Time complexity calculation is in another page

→

Time complexity calculation:

Worst Case: In my algoritm, I divide coins array in 3 part in every recursive call. That is similar to ternary search algoritm.

After 1^{st} iteration, N/3 coins remain $(N/3^1)$
After 2^{st} iteration, N/9 coins remain $(N/3^2)$
After 3^{nd} iteration, N/27 coins remain $(N/3^3)$

Searching stops when coins to search $(N/3^k) \to 1$

$n = 3^k \longrightarrow \log_3 n = k$. So worst case complexity $\to \Theta(\log_3 n)$

Average Case: $T_{av}(N) = \sum_{|I|=N} \underbrace{T(I)}_{\text{Number of basic operations performed by algoritm for input } I.} \cdot \underbrace{Pr(I)}_{\text{Probability of occurence}}$ $\to I \in T_n$

Probability is $= \frac{1}{3}$

There are 3 if condition in my algoritm.

$$\underbrace{\frac{1}{3} \cdot T(1)}_{\text{1. if condition (if size==1)}} + \underbrace{\frac{1}{3} \cdot \left(\frac{1}{2} \cdot T(2) + \frac{1}{2} \cdot T(2) \right)}_{\text{2. if condition (if size==2)}} + \underbrace{\frac{1}{3} \cdot \left(\frac{1}{3} \cdot T\left(\frac{n}{3}\right) + \frac{1}{3} \cdot T\left(\frac{n}{3}\right) + \frac{1}{3} \cdot T\left(\frac{n}{3}\right) + 1 \right)}_{\text{3. if condition (if size} \geq 3)}$$

hese are constant time

This condition determine average case complexity

$$\frac{1}{3} \cdot \underbrace{\left(3 \frac{3}{3} \cdot T\left(\frac{n}{3}\right) + 1 \right)}$$

For solving this, I using master theorem $\to T(n) = T\left(\frac{n}{3}\right) + 1$

$a = 1$, $b = 3$, $d = 0$

$a = b^d \to 1 = 3^0 \to$ so $T(n) \in \Theta(n^0 \cdot \log n)$

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

So average case is $\Rightarrow \Theta(\log n)$

Best Case: Best case is also log(n) according to my algoritm. Because in every cases, coin array divided 3 part and fake coin is searching recursively when remain 1 element (that is fake coin) finded. So best case complexity is also $\Theta(\log_3 n)$

3-) Analyzing average-case complexity of quick sort:

$$\underset{\nearrow \text{Average-case}}{A(n)} = E[T] = \underbrace{E[T_1]}_{\text{This is for partiation operation. high-low+2 comparisions } n+1 \text{ is fixed}} + \underbrace{E[T_2]}_{\text{depends on where pivot element replaced}}$$

$$E[T_2] = \sum_{x} E[T_2 [\bar{x} = x]] \cdot \underbrace{\overbrace{p(\bar{x} = x)}^{\text{probability}}}_{\frac{1}{n}}$$

$\nearrow$ position of pivot

$$A(n) = E[T] = E[T_1] + E[T_2]$$

$$= (n+1) + \sum_{i=1}^{n} E[T_2[x] = i] \cdot \underbrace{p(\bar{x} = i)}_{\frac{1}{n}}$$

$$= (n+1) + \sum_{i=1}^{n} A[i-1] + A[n-i]) \cdot \frac{1}{n}$$

$$= (n+1) + \left.\begin{bmatrix} A[0] + A[n-1] \rightarrow \text{for } i=1 \\ A[1] + A[n-2] \\ \vdots \\ A[n-1] + A[0] \end{bmatrix}\right\} 2 \cdot [A[0] + A[1] + \dots A(n-1)]$$

$$A(n) = (n+1) + \frac{2}{n} \cdot [A[0] + A[1] + \dots A[n-1]]$$

$$n \cdot A(n) = n \cdot (n+1) + 2[\cancel{A[0]} + \cancel{A[1]} + \dots \cancel{} A[n-1]]$$

$$- (n-1) \cdot A(n-1) = n \cdot (n-1) + 2[\cancel{A[0]} + \cancel{A[1]} + \dots \cancel{A[n-2]}]$$

$$n \cdot A(n) - (n-1) \cdot A(n-1) = 2n - 2A(n-1) \rightarrow n \cdot A(n) - A(n-1)(n-1+2) = 2n$$

$$\frac{A(n)}{(n+1)} - \frac{A(n-1)}{n} = \frac{2}{n+1} \rightarrow T(n) = T(n-1) + \frac{2}{n+1}$$

To solving $T(n) - T(n-1) + \frac{2}{n+1}$

$\Longrightarrow$

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

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

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

$\vdots$

$$T(n) = T(n-k) + \frac{2}{n+1} + \frac{2}{n} + \frac{2}{n-1} \cdots \frac{2}{n-k} \rightarrow \text{Assume that } k = n-1 \rightarrow 2$$

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

$$\sum_{i=1}^{n} \frac{2}{i+1} \longrightarrow 2.H(n+1) - 3 \rightarrow \text{So } T(n) = 2.(n+1).H(n+1) - 3.(n+1))$$

$$T(n) = A(n) = 2.(n+1).H(n+1) - 3.(n+1) \in \Theta(n.\log n)$$

Analyzing average - case complexity of insertion sort:

T_i is the number of basic operations at step i where $1 \leq i \leq n-1$

$$T = T_1 + T_2 + \cdots T_{n-1} = \sum_{i=1}^{n-1} T_i$$

$$A(n) = E[T] = E\left[\sum_{i=1}^{n-1} T_i\right] = \sum_{i=1}^{n-1} E[T_i] = E[T_1] + E[T_2] + \cdots E[T_{n-1}]$$

Calculating $E[T_i]$:

$$E[T_i] = \sum_{j=1}^{i} j . \text{Probability}(T_i = j) \rightarrow$$ Probability that there are j comparisions in the i^{th} step.

$$P(T_i = j) = \begin{cases} \frac{1}{i+1} & \text{if } 1 \leq j \leq i-1 \\ \frac{2}{i+1} & \text{if } j = i \end{cases}$$

$$E[T_i] = \sum_{j=1}^{i-1} j . \frac{1}{i+1} + i . \frac{2}{i+2}$$

$$\frac{i.(i-1)}{2i+1} + \frac{2i}{i+1} = \frac{i^2 - i + 4i}{2.i+1} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

$\Longrightarrow$

$$A(n) = E[T] = \sum_{i=1}^{n-1} E[T]$$

$$= \sum_{i=1}^{n-1} \left(\frac{1}{2} + 1 - \frac{1}{i+1} \right) \Rightarrow \frac{n.(n-1)}{4} + (n-1) - \underbrace{\sum_{i=1}^{n-1} \frac{1}{i+1}}_{\frac{1}{2}+\frac{1}{3}+ \cdots \frac{1}{n} = H(n) = \log n}$$

$$A(n) = \frac{n.(n-1)}{4} + n - \log n$$

So $A(n) \in \Theta(n^2)$

Number of Basic Operations and Number of Swaps:

Array: 10, 5, 8, 6, 1, 7, 3, 2, 4, 9

Number of Basic Operations in Insertion Sort: 36

Number of Basic Operations in Quick Sort: 35

Number of Swaps In Insertion Sort: 27

Number of Swaps In Quick Sort: 11

Array: 12, 11, 8, 5, 1, 3, 6, 2, 4, 7, 9

Number of Basic Operations in Insertion Sort: 43

Number of Basic Operations in Quick Sort: 40

Number of Swaps in Insertion Sort: 33

Number of Swaps in Quick Sort: 12

It is clear that, in experimental analysis and in theoretical analysis Quick Sort is better algoritm. Increasing rate of Quick sort is less than Insertion sort. Its time complexity much better.

Conclusion: Average case of Quicsort is $\Theta(n\log n)$ and average case of Insertion sort $\Theta(n^2)$. Quick sort is better algoritm than insertion sort, in theoretical and experimental results show this.

4-) In my algoritm, I used insertion sort algoritm for sorting elements because it is a decrease and conquer algoritm. In the remaining parts, I find median in sorted array. Median is middle element of array. If element number is even, median is $\frac{\text{middle 2 element}}{2}$. If element number is odd, median middle element

For calculating worst case complexity:

This algoritm's worst case complexity equal insertion sort's worst case complexity. Because there is insertion sort calling in my code and remaining parts constant time complexity.

Worst case of insertion sort occurs when array is sorted in reverse order and it is:

$$\underbrace{W(n)}_{\text{Worst case}} = \sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i = \frac{n.(n-1)}{2} = \Theta(n^2)$$

5-) In my algoritm, firstly I calculate $(\min(A) + \max(A)) \cdot \frac{n}{4}$.

After that, I create loop for all elements in array. In the outer loop, I calculate combinations for all steps and then, I create inner loop that is for combination lists in each step. I control if sum of elements in combination sub-array bigger than $(\min(A) + \max(A)) \cdot \frac{n}{4}$, I add that sub-array in comblist. After that I create exhaustive Search function that takes the combination sub-arrays list that satisfy the condition as parameter. For finding optimal subarray, I use recursive call for exhaustive search function and each recursive call, I control another sub array. In exhaustive search function, I call multElements fuction that multiplies every element of a subarray And also I keep optimal multiplication result and optimal subarray in each step. And when all sub-arrays are controlled, I returned optimal sub-array.

Analyzing worst-case complexity:

Outer while loop in optimalArr function $\rightarrow n$ times

Taking combination for each $i \rightarrow O(i\ (n \text{ choose } i))$

Inner while loop in optimalArr function:

$$C\binom{n}{0} + C\binom{n}{1} + C\binom{n}{2} + \text{------} C\binom{n}{n} = 2^n - 1$$

So time complexity of inner and outer while loop is $O(2^n \cdot n)$

At the end of optimalArr fuction I call exhaustive seach fuction.

Time complexity of exhaustive search function:

In this function I call multElement function and it includes while loop for current sub-array elements that satisfy condition And I make recursive calls for each sub-arrays that satisfy condition. So time complexity is $\rightarrow C\binom{n}{0} + C\binom{n}{1} + C\binom{n}{2} \text{---} C\binom{n}{n} = 2^n$

So worst case complexity of optimalArr function is:

$T_{worst} = O(n.2^n)$

Nevra Gürses
161044071