Gökhan HAS
161044067

CSE 321 HW-03

1-) Black-White Boxes Problem

In this problem, there are 2n boxes. We can start by moving the second and 2n-1 boxes. The fourth box is then replaced by the 2n-3 box. Hence, this solution reduces the problem to the same problem with 2(n-2) middle boxes. Here, n can be changed depending on whether it is an odd or even number. If n is even, the number of times it needs to be repeated is equal to n/2. If n is odd, it is equal to (n-1)/2. So, this problems closed-form answer is $\lfloor \frac{n}{2} \rfloor$. Recurrence equation is:

T(n) is the number of moves

$$T(n) = T(\underbrace{n-2}_{\text{decrease}}) + \underbrace{1}_{\text{swap}} \qquad n>2 \qquad T(2)=1 \qquad T(1)=0$$

$$\left.\begin{array}{l} T(2) = 1 \\ T(4) = T(2)+1 \\ T(6) = T(4)+1 \end{array}\right\} \quad T(n) = \frac{n}{2}$$

Time Complexity

$$T(n) \in O(n)$$

→ There are no other possibility in this problem. So worst, best and average case is same O(n).

2-) Fake-Coin Problem

We can use this problems solutions binary search. We can compare any two sets of coins. That is, by tipping to the left, to the right, or staying even, the balance scale will tell whether the sets weigh the same or which of the sets is heavier than the other but not by how much. We can divide n coins into two piles of $\lfloor n/2 \rfloor$ coins each, leaving one extra coin aside if n is odd, and put the two piles on the scale. If the piles weigh the same, the coin put aside must be fake. Otherwise, we can use same method with the lighter pile, which must be the one with the fake coin.

Best Case: $B(n) \in \Theta(1)$, when the fake coin is middle element.

Worst Case: We can easily set up recurrence relation for worst case

$$W(n) = W(\lfloor n/2 \rfloor) + 1 \quad \text{for } n>1, \quad W(1) = 0$$

Suppose that $n = 2^k - 1$ where $k \in \mathbb{Z}^+$. For this algorithm there are k comparisons in the worst case. $k = \log_2(n+1)$

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

Average Case:

Assume that probability of succesfull search is p, $0 \le p \le 1$

Prob. that fake coin can be found in any position $L[i] = \frac{p}{n}$ where $1 \le i \le n$ and n is internal nodes number.

Internal Path Length: The sum of the lengths of the paths from root to the internal nodes.

Average # of comparisons $\frac{p}{n}(1+2+\dots) \Rightarrow$ This corresponds to internal path length of the tree.

$$= \frac{p}{n} \cdot (IPL(T) + n)$$

Prob. that (when fake coin is not in the list) fake coin falls into any one of the $n+1$ intervals $= \frac{1-p}{n+1}$

Leaf Path Length: The sum of the lengths of the paths from the root to the leaf nodes.

Average # of comparisons $= \frac{1-p}{n+1}(3+3+\dots) = \frac{1-p}{n+1} \cdot LPL(T)$

$$A(n) = \sum T(I) . P(I) = \frac{p}{n} \cdot (IPL(T) + n) + \frac{1-p}{n+1} LPL(T)$$

↓

Total Avg. Case

For 2-tree T, $IPL(T) = LPL(T) - 2I$, where I is the number of internal nodes. T, $LPL(T) \geq L * \lfloor \log_2 L \rfloor + 2(L - 2^{\lfloor \log_2 L \rfloor})$ where L is the number of leaf nodes.

$$A(n) = \frac{p}{n}(IPL(T)+n) + \frac{1-p}{n+1} LPL(T) = \frac{p}{n}(LPL(T)-n) + \frac{1-p}{n+1} LPL(T)$$

$$= \left(\frac{p}{n} + \frac{1-p}{n+1}\right) LPL(T) - p \quad \left(\text{There are } n+1 \text{ leaf nodes and } n \text{ internal nodes } L = n+1\right)$$

$$\geq \underbrace{\left(\frac{p}{n} + \frac{1-p}{n+1}\right)}_{\frac{1}{n}} \underbrace{\left[(n+1) * \lfloor \log_2(n+1) \rfloor + 2\left((n+1) - 2^{\lfloor \log_2(n+1) \rfloor}\right)\right]}_{n \log n} - p$$

$$\frac{1}{\cancel{n}} \cdot \cancel{n} \log n$$

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

3-) Quick Sort and Insertion Sort

Quick Sort's Avg. Case:

$T = T_1 + T_2$

- T → random variables
- T_1 → # of operations in rearrange
- T_2 → # of operations in recursive calls.

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

- $E[T_1]$ → fixed (high-low+2 comparisons)
- $E[T_2]$ → depends on where the pivot has been placed.

$$E[T_2] = \sum_{x} E[T_2 \mid \bar{x} = x] \cdot \underbrace{P(\bar{x} = x)}_{\frac{1}{n}}$$

- x → position of the pivot
- T_2 → random variable
- $\bar{x}$ → random variable represent the pivot

$E[T_1] = high - low + 2$
$= n+1$

$$A(n) = E[T] = E[T_1] + E[T_2]$$
$$= (n+1) + \sum_{i=1}^{n} E[T_2 \mid \bar{x} = i] \cdot \underbrace{P(\bar{x} = i)}_{\frac{1}{n}}$$

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

$$= (n+1) + \begin{bmatrix} + A(0) + A(n-1) \rightarrow i=1 \\ + A(1) + A(n-2) \rightarrow i=2 \\ \vdots \\ + A(n-2) + A(1) \rightarrow i=n-1 \\ + A(n-1) + A(0) \rightarrow i=n \end{bmatrix} \cdot \frac{1}{n}$$

$$= (n+1) + \frac{2}{n}\left[A(0) + \dots + A(n-1)\right]$$ → Full history recurrence relation

$$\begin{array}{rl} & n.A(n) = n(n+1) + 2\left[A(0) + A(1) + \dots + A(n-1)\right] \\ - & (n-1).A(n-1) = n(n-1) + 2\left[A(0) + \dots + A(n-2)\right] \\ \hline & n.A(n) - (n-1).A(n-1) = 2n + 2A(n-1) \end{array}$$

$$\frac{1}{n(n+1)} \left\{ \frac{A(n)}{n+1} - \frac{A(n-1)}{n} = \frac{2}{n+1} \right. \Rightarrow \frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2}{n+1}$$

Change of variable $t(n) = \frac{A(n)}{n+1} \Rightarrow \boxed{t(n) = t(n-1) + \frac{2}{n+1}}$

$A(0) = 0 \rightarrow$ initial condition

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

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

$t(n) = t(n-3) + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$ → $t(n) = \sum_{i=2}^{n} \frac{2}{i+1}$

$t(n) = 2.H(n+1) - 3$ → Harmonic Series

$A(n) = (n+1).t(n)$

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

Hoare Partition Avg Case: (I implemented python code this method.)

A portition can happen in any position s $(0 \leq s \leq n-1)$ after $n+1$ comparisons are made to achive the partition. After the partition, left and right subarrays will have s and $(n-1-s)$ elements.

Assuming that the partition split can happen in each position s with probability $\frac{1}{n}$, we get the following recurrence

$$C_{avg}(n) = \frac{1}{n} \sum_{s=0}^{n-1} \left[C_{avg}(s) + C_{avg}(n-1-s) + (n+1) \right] \quad \text{for } n>1$$

$$C_{avg}(0) = 0$$
$$C_{avg}(1) = 0$$
$$\vdots$$
$$C_{avg}(n) \approx 2n \ln(n) \approx 1.39\, n \log_2(n)$$

Thus, on the avg, quicksort makes only 39 % more comparisons than in the best case.

INSERTION SORT AVG CASE:

We can say T_i be the number of basic operations at step i, where $1 \le i \le n-1$

$$T = T_1 + T_2 + T_3 + \dots + T_{n-1} = \sum_{i=1}^{n-1} T_i$$

$$A(n) = \overset{\nearrow \text{expected}}{E}(T) = E\left(\sum_{i=1}^{n-1} T_i\right) \qquad E[T_i] = \sum_{j=1}^{i} j \, Prob(T_i = j)$$

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

$$E[T_i] = \sum_{j=1}^{i-1} \left(j * \frac{1}{i+1}\right) + i * \frac{2}{i+1} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

$$A(n) = E[T] = \sum_{i=1}^{n-1} \left(\frac{i}{2} + 1 - \frac{1}{i+1}\right) = \frac{n(n+1)}{4} + (n-1) - \sum_{i=1}^{n-1} \frac{1}{i+1}$$

$$A(n) \simeq \frac{n(n+1)}{4} + n - \dots\dots \in \Theta(n^2)$$

Compare Swap Elements:

Lists	[50, 49, 48, 47, 46, 45]	[50, 49, 48, ..., 42, 41, 40]	[50, 49, ... 36, 35]
Quick Sort	3	5	8
Insertion Sort	15	55	120

According to above table, we can easly said that if we send an array in reverse order quicksort is increasing much slower than the insertion sort. I implemented quicksort by using hoare partition.

4-) FIND MEDIAN:

In this question, we wanted a decrease-and-conquer algorithm. So we can call insertion sort algorithm. Because insertion sort algorithm is a decrease-and-conquer algorithm. And then, we can return median by using if-else block.

So, this algorithm's worst case is insertion sort algorithm's worst case. For each iteration of the for loop, the basic operation is executed maximum number of times.

$$W(n) = \sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} + 1$$

$\downarrow$

comes if-else block to check list length's is odd or even

$$W(n) \in \Theta(n^2)$$

5-) OPTIMUM SUB-ARRAY

In this question, I am written some helper functions. (You can see 161044067.py file in this zip.)

The calculate_sum function worst case time complexity is $O(n)$. Because I call min and max python3 list's functions. (n is list's length.)

The clear_duplicates function worst case time complexity is total sublist's length in main list. Because parameter "array" has all sublist in main list. In fact, some (very rare) had two, so this function was needed. Number of subsets are 2^n. So this function's worst case time complexity is $O(2^n)$.

The multiplyList function worst case time complexity is $O(m)$. What is m? m is the length of the list of subsets greater than or equal to SumB.

The nth-sublist function is a recursive function. And I will call list size -1 and I have a for loop 0 to length of list. As there is a loop over n and recursive calls is $T(n-1)$.

$W(n) = W(n-1) \cdot n$ and $W(0) = 1$ is initial condition

$W(0) = 1$ for $n \geq 1$ $W(1) = 1$

$W(1) = W(0) \cdot 1 = 1$

$W(2) = W(1) \cdot 2 = 2$

$W(3) = W(2) \cdot 3 = 6$

$W(4) = W(3) \cdot 4 = 24$

We can show that this recurrence relation is $n!$.

The main find-optimal function calls other functions which are in above. No functions are called within the loop, and $n!$ has the most time complexity among the called functions. So, worst case $W(n) \in O(n!)$