Introduction to Algorithm Design

HW3 Report

- **Question 1:**

In this question I design a decrease-and-conquer algorithm for fake coin problem.

This algorithm’s Best case, Avarege case, Worst case are equal because input is always same. Input is “black, black, black, ….., White, White, White, …….”. There are n black boxes and n white boxes. My algorithm takes boxes array, firt index and last index of array. If first index % 2 == 1, firstelement replace with last element. If first index smaller than last index, function increase first index, decrease last index and calls itself.

So function call itself n times. And its complexity;

B(n) = A(n) = W(n) = O(n)

- **Question 2:**

In this question I design a decrease-and-conquer algorithm for fake coin problem.

In this problem, there are n coins which look exactly the same but one of them is fake. And we will find fake coin.

In my algorithm;

If the number of coins is odd, we set one aside and weigh the remaining money (n-1) / 2 into two parts. If the result is equal, the money we allocate is fake; if one side is lighter, the fake coin is among the coins in that group, and we weigh those coins into two equal groups. By dividing the coins in half, we will find the fake coin.

Best case:

If the fake coin in middle element of list
B(n) = O(1)

Worst case:

First purpose that n = $2^k - 1$ for k ∈ Z^+
X = L[n]
Compare x with L[2^{k-1}], L[2^{k-2}], L[2^{k-3}],……. L[1]

There are k comparisons in the worst case
K = log(n +1)
W(n) = O(log(n))

Average case:

First purpose that n = $2^k - 1$ for k Є Z^+

$\sum_{i=1}^{logn} \frac{((number\ of\ iterations\ in\ case\ i)*(\text{number of nodes in case } i))}{n}$

= $\sum_{i=1}^{logn} i * \frac{n}{2^i}$ / n

= (1 * $\frac{n}{2^{logn}}$ + …. +(log(n)-1) *$\frac{n}{2^2}$ + logn * $\frac{n}{2}$) / n

=log(n)

A(n) = O(logn)

- **Question 3:**

In this I implement the quick sort and insertion sort algorithms and this sort returned their the number of swap operations.

	Already Sorted	Reversly Soted	Unsorted
Insertion Sort	54 swap	45 swap	19 swap
Quick Sort	0 swap	29 swap	54 swap

Average case of insertion sort:

$\sum_{i=1}^{n-1} \frac{i}{2}$ = $\frac{1}{2}$ (1 + 2 +3 + …… + (n - 1)) = $\frac{(n-1)n}{4}$

A(n) = (n^2)

Average case of quick sort;

A(n) = $\frac{1}{n}$ * ($\sum_{i=0}^{n-1}(n+1) + A(i) + A(n-1-i)$)

For n > 1

A(0) = 0 A(1) = 1

……

A(n) ~ 2nln(n) ~ nlog(n)

A(n) = nlog(n)

Average case of quick sort is smaller than average case of insertion sort.

So quick sort better than insertion sort.

- **Question 4:**

  In this question, I design a decrease and conquer algorithm that finds the median of an unsorted array.
  I use quick select algorithm for find the median.

  Worst case: If input size is even number and array is sorted big to small or array is unsorted, quickSelect function calls quickSelect_helper 2 times. And quickSelect_helper function call itself n/2 times. There is a for loop in quickSelect_helper function. This loop turns everytime n time but quickSelect_helper function takes nearly half of array every time.
  So;

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

  a = 2
  b = 2
  d = 1
  a = b^d -> 2 = 2^1

  T(n) = 0(n.log(n))

- **Question 5:**

  In this part I design a recursive exhaustive search algorithm that finds the optimal sub-array of an array.

  The optimality criterion is to have the minimum number of multiplication of items in the sub-array.

  We find the optimal subarray with this formula;

  $SUM(B) \leq (max(A)+min(A)) * (n / 4)$

  In my algorithm there are two choices for every element in the array. First choice is subarray include element. Second choice is subarray not include element.

  If the array size is n, it takes 2^n time. Because for every element function calls itself 2 times. And it finds all subarray of array.

  Worst case;

  W(n) = O(2^n)