= CSE321 – Homework 3 =

1) There are 2n boxes.

$n \rightarrow$ Black (B)

$n \rightarrow$ White (W)

Assume that this array keeps same number of B and W.

| B | B | B | B | W | W | W | W |

This array's last version is that,

| B | W | B | W | B | W | B | W |

The algorithm which will be use for this problem likes insertion sort. Because insertion sort is an example of decrease-and-conquer. We select 1 element of the array like the last element and we decreased elements which will be look by 1. Add this last element after B when there are 2 B one after the other, and shift all the elements to right. so, there are less element to look.

| B | B | B | B | W | W | W | W | $\rightarrow$ | B | W | B | B | B | W | W | W | $\rightarrow$ | B | W | B | W | B | B | W | W | $\rightarrow$ | B | W | B | W | B | W | B | W |

1 comparison 1 swap
shifting $\rightarrow$ swap. for 8 element, 3 comparison $\rightarrow$ 2n elements, n-1 comp.

Pseudocode for sortingBoxes (arr [0... 2n])

```
for i <- 0 to 2n do
        curr <- arr [i]
        next <- arr [i+1]
        if curr = 'B' and next = 'B'
              for j <- 2n-1 to i+1 do
                    arr [j] <- arr [j-1]
              endfor
              arr [j] <- last
        endif
endfor
```

* Worst case complexity:

$$\sum_{i=0}^{2n} \left(\sum_{j=i+1}^{2n-1} 1 \right) \rightarrow \text{max } 2n \text{ times} = \sum_{i=0}^{2n} 2n = \sum_{i=1}^{2n+1} 2(n-1) = \underbrace{(2n-2) + (2n-2) + \dots + (2n-2)}_{2n \text{ times}}$$

$$A(n) = 2n \cdot (2n-2) = 4n^2 - 4n \rightarrow \underline{A(n) = O(n^2)}$$

* Best case complexity: if array is already sorted like $n=1 \rightarrow 2n=2 \rightarrow$ | B | W |

$$\sum_{i=0}^{2n} \left(\sum_{j=i+1}^{2n-1} \right) \rightarrow \text{min } 0 \text{ times} \qquad \sum_{i=0}^{2n} 1 = \sum_{i=1}^{2n+1} 1 = \underbrace{1+1+\dots+1}_{2n \text{ times}} \rightarrow 2n.(1) = 2n \rightarrow \underline{B(n) = O(n)}$$

* Average case complexity:

$$\frac{\text{All possible cases}}{\#\text{ of cases}} = \frac{2n \cdot (2n-2)}{2n+1} = 4n^2 - 4n \,/\, 2n+1 \rightarrow \underline{A(n) = O(n)}$$

2) Assume that, fake coin is lighter than others.

solving with Decrease and conquer Algorithm

Assume that, we have 10 coins and one of them is fake.

Decrease by 2 and weigh this 2 coin.

1) $(C_1\ C_2)$ 2) $(C_3\ C_4)$ 3) $(C_5\ C_6)$ 4) $(C_7\ C_8)$ 5) $(C_9\ C_{10})$

1) if their weights are equal, then weigh other 2 coins till found the fake lighter coin.

For this algorithm:

* worst case:

Assume that, C_{10} is fake coin, and goes through with subarrays by 2.

Starts from C_1 and goes till C_{10} → So, all the coins visited.

So, worst case complexity is $W(n) = O(n)$.

* Best case:

Assume that, C_1 is fake coin and goes through with subarrays by 2, decrease by 2.

Compare C_1 and C_2 and C_1 is lighter than C_2, fake coin is found.

So, best case complexity is $B(n) = O(1)$

* Average case:

Average case must be between or equal $O(1)$ and $O(n)$.

Assume that C_5 is fake coin, then searches $n/2$ times.

$$\sum_{i=0}^{n} 1 = \underbrace{1 + 1 + \dots + 1}_{n \text{ times}} = n \rightarrow \underline{O(n)}$$

3) Analyze Insertion sort:

it consists of two functions. These two functions also include 1 loop inside.

* Best case:

If there is a sorted list, then for loop, inside Isort will operate all the elements of the list, while loop which is inside Insert function will not operate elements because the list is already sorted.

Then the best case is $\underline{O(n)}$

* Worst case:

If there is a random sorted list, then Isort and Insert will iterate this list and for loop and while loop operate this list. For this reason worst case complexity is $\underline{O(n^2)}$.

* Average case:

Calculating average, I use a formula:

$$\underline{\text{All possible cases(time)}} \Rightarrow \sum_{i=0}^{n} \underbrace{\sum_{i=1}^{n}}_{\text{next values}} i = \sum_{i=0}^{n} (1+2+\dots+n) = \sum_{i=0}^{n} \frac{n.(n+1)}{2}$$

# of cases

$$= \frac{1}{2} \cdot \sum_{i=1}^{n+1} (n+1).n = (n+1).(n-1).(n) \rightarrow \text{all possible cases.}$$

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

$(n+1) \rightarrow$ # of cases

Analyze Quick sort:

it consist of 4 functions. Partition function consists nested while loops.

* Best case:

If there is a sorted list, then 1 while True loop will operate this list and look this list's sublist. Then the best case is $O(n.\log n)$

* Worst case:

If there is a random sorted list, then all the while loop will operate this list and look this list's sublists and swap elements if need.

Then worst case is $O(n^2)$.

* Average case: Assume that, sublists have i and $n-i$ elements.

$$T(n) = T(i) + T(n-1) + c.n$$

$$E(T(i)) = 1/n . \sum_{j=0}^{n-1} T(j)$$

$$E(T(n-i)) = E(T(i))$$

$$T(n) = 2/n \left(\sum_{j=0}^{n-1} T(j) \right) + c.n$$

$$n.T(n) = 2.\left(\sum_{j=0}^{n-1} T(j) \right) + c.n^2$$

$$(n-1).T(n-1) = 2.\left(\sum_{j=0}^{n-2} T(j) \right) + c.(n-1)^2$$

$$n.T(n) = 2.\left(\sum_{j=0}^{n-1} T(j) \right) + c.n^2$$

$$-\ (n-1).T(n-1) = 2.\left(\sum_{j=0}^{n-2} T(j) \right) + c.(n-1)^2$$

$$n.T(n) - (n-1).T(n-1) = 2.T(n-1) + 2cn - c$$

$$n.T(n) = (n+1).T(n-1) + 2cn \Rightarrow \frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$$

$$= \frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum_{j=2}^{n+1} 1/j$$

$$= \frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c.\ln(n) + 2c_8 = \ln(n) + c_2 = O(\ln n)$$

$$T(n) = O(n.\log n) \rightarrow \underline{A(n) = O(n.\log n)}$$

Analyze swap operations :

Insertion sort :

Insertion sort algorithm does not have any swap operation exactly But has some interchanging and shifting left by 1. I assumed this operations as swap. For this reason every compare and interchanging operation has shifting operation and much swap operation than quick sort.

Quick sort .

Quick sort algorithm does swap operation swap operations is located in while true loop and this operation will only be used when necessary. For this reason quick sort has less than insertion sort .

4) To find the median of an unsorted array, I used an algorithm that quick sort. After sorting array, median is found correctly.

* Worst case:

Complexity of this program is $O(n^2)$ because of quick sort.

5) To find optimal subarray, implemented 4 functions recursively.

1) findSubarrays → takes base array and finds all subarrays.

→ this function n times executed → $W(n) = O(n)$.

2) findSumofSub → takes all the subarrays and finds the summation of them

→ $2^n - 1$ times executed → $W(n) = O(2^n)$

3) findMultofSub → takes all the subarrays and finds the multiplication of them

→ $2^n - 1$ times executed → $W(n) = O(2^n)$

4) findsOptimalSubarray → takes sums, mults and calculates optimal subarray

→ $2^n - 1$ times executed → $W(n) = O(2^n)$

The complexity of this program is $\underline{W(n) = O(2^n)}$