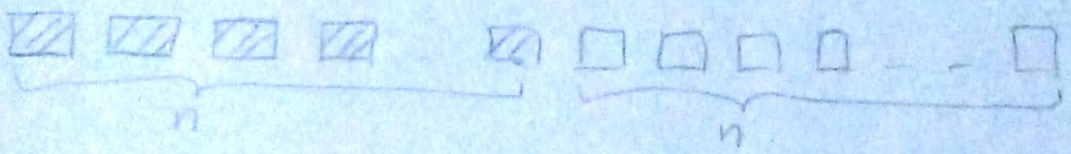
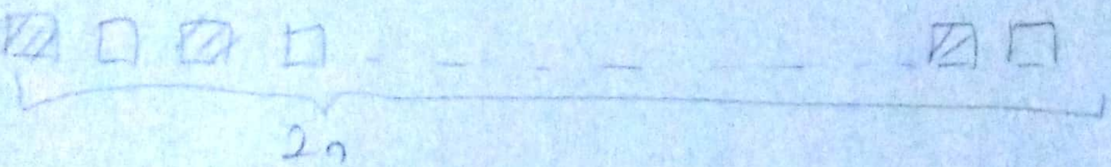
1- There are 2n boxes standing in a row, the first n of them are black and remaining n boxes are white. Design a decrease-and-conquer algorithm that makes the boxes alternate in a black-white-black-white pattern using minimum number of moves. Remember that interchanging any two boxes is considered to be one move. Analyze the worst-case, best-case and average-case complexities of your algorithm. Explain your algorithm in the report file.

First, let's visualize this question.



There are n black and n white boxes. And we want to sort the boxes like:



We need to use insertion sort for this problem. But we have two type same elements. So we must give an id for theese elements

Black one's $\longrightarrow 0, 2, 4, \ldots, 2n-2$

White one's $\longrightarrow 1, 3, 5, \ldots, 2n-1$

```
procedure InsertionSort(L[1:(2*n)])
    for i=2 to (2*n) do
        current = L[i]
        position = i-1
        while ((position ≥ 1) and (current < L[position])) do
            L[pos+1] = L[pos]
            pos = pos-1
        end while
        L[pos+1] = current
    end for
end
```

Best Case → Occurs if the input is already sorted

$$B(n) = \sum_{i=2}^{n} 1 = n-1 \in \Theta(n)$$

Worst Case → For each iteration of the for loop, the basic operation is executed maximum number of times.

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

Average Case → • Let T_i be number of basic operations at step i, $1 \leq i \leq n-1$

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

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

• Each of T_i's are random variables.

• Calculating $E[T_i]$:

$$E[T_i] = \sum_{j=1}^{i} j \cdot P(T_i = j)$$

→ probability that there are j comparisions in the i^{th} step.

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

$$E[T_i] = \sum_{j=1}^{i-1} j \cdot \frac{1}{i+1} + i \cdot \frac{2}{i+1} \Rightarrow \frac{i(i-1)}{2(i+1)} + \frac{2i}{i+1} \Rightarrow \frac{i^2+3i}{2(i+1)} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

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

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

$$H(n) \downarrow \quad \underbrace{\sum_{i=1}^{n-1} \frac{1}{i+1} = \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}}_{\text{Harmonic Series}}$$

2- Fake coin problem is a famous problem and there are many versions of it in the literature. In our version, there are n coins which look exactly the same but one of them is fake. You can use a weighbridge which has two scales to find the fake coins. Design a decrease-and-conquer algorithm which finds the fake coin. Analyze the worst-case, best-case and average-case complexities of your algorithm. Explain your algorithm in the report file.

* We can assume that fake coin is lighter then other coins. And we divide the coins in 2 parts. ($\frac{n}{2} + \frac{n}{2} \to$ 2 parts, if n is odd leaving 1 extra)

Then compare theese 2 parts. If the parts has same weigh, the coin put aside must be fake otherwise the lighter part has fake coin.

We apply same things to the lighter part until we get the fake one.

* FakeCoin(Coin [ ], n)

```
if (n=1) THE COIN IS FAKE
else
    if(n mod 2 =1) divide the coins into two piles of
    [n/2], [n/2] and 1
        if two piles has same weigh
            FAKE COIN IS LEFT ONE
        else compare two piles and continue with lighter one
    else divide the coins into two piles of [n/2], [n/2]
        and compare two piles and continue with lighter one
```

* It is almost identical to the number of comparisons in binary search (The differerre is the initial condition)

Worst-case $\to W(n) = W\lfloor n/2 \rfloor + 1$ for $n>1$, $W(1)=0$

$W(n) = \log_2 n \in \Theta(\log_2 n)$

Best-case $\to B(n) = 1 \in \Theta(1)$

Average-case $\to A(n) = \log_2 n \in \Theta(\log_2 n)$

3. Implement the quicksort and insertion sort algorithms and count the number of swap operations to compare these two algorithms. Analyze the average case of the algorithms. Compare the operations count in your report file to decide which algorithm is better and support your analysis by using the theoretical average-case analysis of your algorithms. Make a comparative evaluation of your experimental analysis with the theoretical analysis. Discuss the results.

* Quick Sort → Worst case occurs the array is already sorted in decreasing order.

Recurrence for total number of swaps in this case:

$T(n) = T(n-1) + O(n)$ // $\boxed{O(n)}$ swaps will occur in alternate calls to partition algorithm.

$= O(n^2)$

Insertion Sort → Worst case occurs when the array already sorted in ascending order.

When a new element is inserted into an already sorted array of X size, it can lead to X swaps (in case it is the smallest of all) in worst case. For n-1 iterations of insertion sort total swaps will be $\boxed{O(n^2)}$

* INSERTION SORT AVERAGE CASE

- Let T_i be number of basic operations at step i, $1 \leq i \leq n-1$

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

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

- Each of T_i's are random variables.

- Calculating $E[T_i]$

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

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

$$E[T_i] = \sum_{j=1}^{i-1} j \cdot \frac{1}{i+1} + i \cdot \frac{2}{i+1} \Rightarrow \frac{i(i-1)}{2(i+1)} + \frac{2i}{i+1} \Rightarrow \frac{i^2+3i}{2(i+1)} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

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

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

$$\underbrace{\sum_{i=1}^{n-1} \frac{1}{i+1} = \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}}_{\text{Harmonic series}}$$

QUICK SORT AVERAGE CASE

$T = T_1 + T_2 \longrightarrow$ T's are random variables

T_1: numbers of operations in rearrange

T_2: numbers of operations in recursive call

$$A(n) = E[T] = E[T_1] + E[T_2] \Rightarrow E[T_2] = \sum_x E[T_2 \mid X = x] \cdot P(X = x)$$

$$A(n) = (n+1) + \sum_{i=1}^{n} E[T_2 \mid X = i] \cdot \underbrace{P(X=i)}_{1/n}$$

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

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

$$n \cdot A(n) = n(n+1) + 2(A(0) + A(1) + \dots + A(n-1))$$

$$-(n-1)A(n-1) = n(n-1) + 2(A(0) + A(1) + \dots + A(n-2))$$

$$n\,A(n) - (n-1)\,A(n-1) = 2n + 2A(n-1)$$

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

Change of variable: $T(n) = \frac{A(n)}{n+1}$

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

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

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

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

Scanned by CamScanner

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

$$= 2H(n+1) - 3$$

$$= 2\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n+1}\right)$$

$$= 2 + 1 + \cdots + \frac{2}{n} + \frac{2}{n+1}$$

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

* Quick sort has $O(n)$ number of swaps.
Insertion sort has $O(n^2)$ number of swaps.
Quick sort's average case is $\Theta(n\log n)$.
Insertion sort's average case is $\Theta(n^2)$.

So as you can see quick sort is much better than insertion sort.

* For make a comparative evaluation of experimental analysis with theoretical analysis, I choose an unsorted array. And implement it to count the number of swaps. Than calculate it with theoretically.

unsorted array for inserition sort

22	11	7	5	4	2

unsorted array for quick sort

2	4	7	5	22	11

Quick Sort Number of Swaps

experimentally analysis → $O(n)$
theoretical analysis → $O(n)$

In experimentally analysis of quicksort number of swaps is $O(n)$;
Also in theoretical analysis is too. So values are the same.

Insertion Sort Number of Swaps

experimentally analysis → $0{,}6\ O(n^2)$
theoretical analysis → $O(n^2)$

In experimentally analysis of insertion sort number of swaps less than $O(n^2)$. But it's close. It's $0{,}6\ O(n^2)$.

4- Design a decrease and conquer algorithm that finds the median of an unsorted array. Implement your algorithm and analyze the worst case complexity of your algorithm.

* Array is unsorted so we must sort the array first. And then do binary search for find the median. For sorting, we'll use insertion sort.

* procedure InsertionSort(L[1:n])
    for i=2 to n do
        current = L[i]
        position = i-1
        while((position ≥ 1) and (current < L[position])) do
            L[pos+1] = L[pos]
            pos = pos-1
        end while
        L[pos+1] = current
    end for
  end

procedure FindMedian (L[1:n])
    if n%2 = 1
        return float (L[int((n-1)/2)]
    else
        return float((L[int((n-1)/2)] + L[int(n/2)]) /2.0)
end

* The complexity of this algorithm is equal to sum of insertion sort and finding median function's time complexities. FindMedian's time complexity is just 1. But insertion sort time complexity is bigger than that. So we ignore the 1 and focusing the worst case of insertion sort.

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