1. a) $T(1) = 1$

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

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

→ $T(n) \in O(n^2)$ ①

→ $T(n) \in \Omega(n^2)$ ②

① Para isso, precisamos provar que $\exists$ c, k constantes t.q. $\forall n > k$ $T(n) \leq c \cdot n^2$.

Tomemos então $k = 1$, $c = 1$

Passo Base: $T(1) = 1 \leq 1 \cdot 1^2$

Passo Indutivo: Suponha que para $n-1$, temos $T(n-1) \leq c(n-1)^2$. Então:

$T(n) = T(n-1) + n \leq c(n-1)^2 + n$

$\leq c(n^2 - 2n + 1) + n \leq cn^2 + (1-2c)n + c$

Como $c = 1$ e $n \geq 1$ temos que $(1-2c)n + c \leq 0$

Logo:

$T(n) \leq cn^2$

Assim, $T(n) \in O(n^2)$

② Para isso, precisamos provar que $\exists$ c, k constantes t.q. $\forall n > k$ $T(n) \geq c \cdot n^2$

Tomemos então $k = 1$, $c = \frac{1}{2}$

Passo Base: $T(1) = 1 \geq \frac{1}{2} \cdot 1^2$

Passo Indutivo: Suponha que para $n-1$ temos $T(n-1) \geq c(n-1)^2$. Então:

$T(n) = T(n-1) + n \geq c(n-1)^2 + n \geq c(n^2 - 2n + 1) + n$

$\geq cn^2 + (1-2c)n + c$ (como $c = \frac{1}{2}$ e $n \geq 1$, temos que $(1-2c)n + c \geq 0$. Logo:)

$T(n) \geq cn^2$ Assim $T(n) \in \Omega(n^2)$

Como $T(n) \in O(n^2)$ e $T(n) \in \Omega(n^2)$, temos que $T(n) \in \Theta(n^2)$.

b) $S(0) = S(1) = 1$

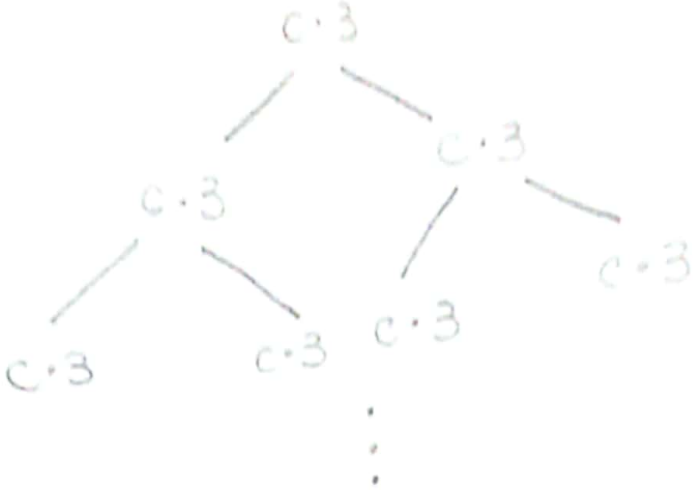
(i) $S(n) = 2S(n-2) + 3$

$S(3) = 5$ $S(6) = 29$

$S(2) = 5$ $S(7) = 29$

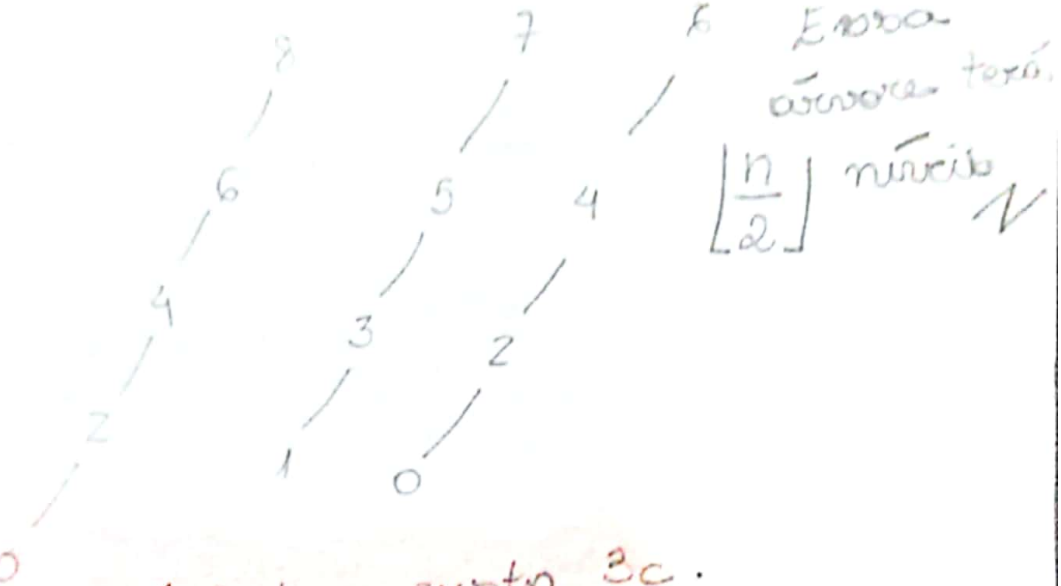
$S(4) = 13$

$S(5) = 13$



(ii) $2c = n$

$c = \left\lfloor \frac{n}{2} \right\rfloor$



Esta árvore terá $\left\lfloor \frac{n}{2} \right\rfloor$ níveis

(iii) O k-ésimo nível tem custo $2c$.

(iv) $f(n) = 2^{\lfloor \frac{n}{2} \rfloor}$ $S(n) \in \Theta\left(2^{\lfloor \frac{n}{2} \rfloor}\right)$

c) $R(n) = 4R(n/2) + n^2 \log n$

$a = 4$

$b = 2$

$f(n) = n^2 \log n$

$\log_b a = \log_2 4 = 2$

$f(n) = \Omega(n^{2+\varepsilon})$

$4f(n/2) = 4\left(\frac{n}{2}\right)^2 \log\left(\frac{n}{2}\right)$

$= n^2 \log\left(\frac{n}{2}\right)$

$\leq cn^2 \log n$

$c < 1$

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

$n^2(\log n - \log 2)$

$n^2 \log n - n^2 \log 2$

2 - [6 5 3 4]
1 ↑ ↑ ↑

key = 3

[3 3 4] i = 3

key = 5

[3 4 4] i = 2

key = 6

[3 4 5] i = 3

i = 4

a) Esse algoritmo ordena a lista A.

b) Ao início da iteração j do for, o subvetor A[j+1 ... n] está ordenado. Assim, nessa iteração, colocamos o elemento da posição A[j] do vetor original na posição correta dele no subvetor A[j ... n].

c) X = total de operações

X_i = total de operações na i-ésima iteração do for

$E(X_i) = \frac{1}{n-j+1}$ ⇒ Para ficar mais fácil tomemos $E(X_i) = \frac{1}{i+1}$ (Ir de trás para frente ou de frente para trás não fará diferença)

$E(X) = E\left(\sum_{i=1}^{n-1} X_i\right)$

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

$= \ln n + O(1) - 1$

3- a) Falso. Tomemos por exemplo a lista [1 2 3 4]. A série de operações a seguir gera a permutação identidade:

① [1 2 3 4]
② [2 1 3 4]
③ [1 2 3 4]
④ [1 2 4 3]
⑤ [1 2 3 4]

(Isso só é válido para listas de tamanho par, mas é possível).

b) [1 2 3]

[3 2 1]
[3 1 2]
[2 1 3]
[2 3 1]
[1 3 2]

[2 1 3]
[1 2 3]
[1 3 2]
[2 3 1]
[1 3 2]

Falso, para listas de tamanho ímpar esse algoritmo gera a permutação identidade com probabilidade 0, enquanto gera outras permutações com probabilidade maior.

4- a) Pois o número de operações Dequeue que eu posso realizar está ligado ao número de Push's em P que eu realizo (ou de Enqueue).

b) Enqueue = 4
Dequeue = 0

Escolhi esses valores para que a função Enqueue tenha todo o custo e eu não precise me preocupar com a Dequeue. Cada Push e Pop nas pilhas tem custo 1. Mas, para retirar um elemento da fila precisamos colocar ele na pilha P inicialmente (não se pode retirar o que nunca foi colocado), retirá-lo da pilha P, colocá-lo na pilha Q e então retirá-lo de Q. Não importa quantos elementos tenhamos nas pilhas, todo elemento para sair precisa passar pelos 4 passos, cada um de custo 1. Como é mais fácil ver que a função Enqueue é O(n) podemos colocar todo o "crédito" nela e, independente da ordem que executarmos Enqueue e Dequeue, teremos sempre crédito sobrando ao colocar 4 em Enqueue e 0 em Dequeue.