Prova Antiga 2010/1

1) a- Caso base ($n = 1$)

$T(1) = 1 \quad \Theta(1^2) = \Theta(1)$

$0 \leq c_1 \cdot 1 \leq 1 \leq c_2 \cdot 1$

$\forall c_1 < c_2$

H.I.: $T(n) \in \Theta(n^2) \iff \exists\, c_1, c_2, n_0 : 0 \leq c_1 n^2 \leq T(n) \leq c_2 n^2 \quad \forall n \geq n_0$

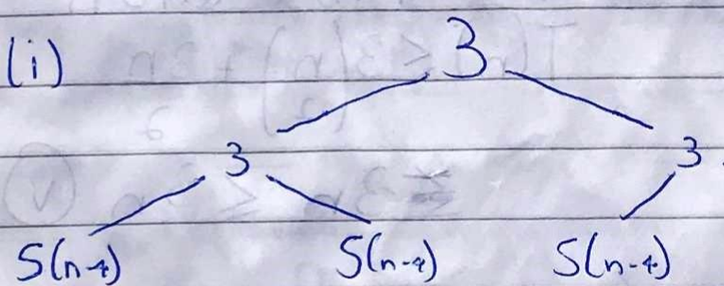
$0 \leq c_1 n^2 \leq (n-1)^2 + n \leq c_2 \cdot n^2$

$c_1 n^2 \leq n^2 - n + 1 \leq c_2 n^2$

$c_1 n^2 \leq n^2 - n + 1$ $\qquad$ $n^2 - n + 1 \leq c_2 n^2$

(V) $c_1 = \frac{1}{2}, \; n_0 = 2$ $\qquad$ (V) $c_2 = 2 \quad n_0 = 2$

b- (i)



(ii) $\lfloor \frac{n}{2} \rfloor$ $\qquad$ (iii) $C(k) = 2^k \cdot 3$ $\qquad$ (iv) $S(n) = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor - 1} 2^k \, 3$

$= \frac{3(2^{\frac{n}{2}} - 1)}{2 - 1} \qquad f(n) = 2^{\frac{n}{2}}$

c- Pelo teorema mestre, $a = 4 \quad b = 2 \quad \log_2 4 = 2$

$n^2 \log n \in \Theta(n^2 \cdot (\lg n)^k), \; k = 1$

Logo $f(n) = n^2 (\log_2 n)^2$

② a- O algoritmo ordena os números da lista em ordem crescente

b- Eu utilizaria o seguinte loop invariante:

"Na j-ésima iteração, a sublista $A[j+1 \dots n]$ ~~contém os~~) contém os $n-j$ últimos elementos da lista original, em ordem crescente"

c- X: total de operações (*)

Na iteração i, a chance de $A[i] < key$ é 50% então esperamos ter $\frac{n-1}{2}$

$$E[X_i] = \frac{n-1}{2} \quad E[X] = \sum_{i=1}^{n-1} \frac{n-1}{2} = (n-1)\frac{n}{2} - \sum_{i=1}^{n-1} \frac{1}{2}$$

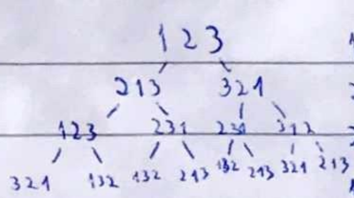
$$= \frac{n^2-n}{2} - \frac{\frac{1}{2}\left(\frac{1}{2^{n-1}} - 1\right)}{\frac{1}{2} - 1}$$

$$= \frac{n^2-n}{2} + \frac{1-2^{n-1}}{2^{n-1}}$$

③ a- Falsa, pois para n par, se quando i for ímpar o número aleatório escolhido for $i+1$ e quando i for par, o número escolhido for $i-1$, a permutação identidade será atingida

b- X_k: chance do elemento k terminar na posição k+1

se $i=k$ $\quad\frac{1}{n-1}$ $\quad$ X ~~ou~~ $\quad$ se $i=k+1$ $\quad\frac{1}{n-1}$



1	2	3	4
2	1	3	4
2	3	1	4
1	3	2	4
4	3	2	1

54321

FALSO! com 3 elementos ele nunca gera a perm. identidade

132 - 3x
213 - 3x
123 - 0x

④ a - Isso ocorre pois para executarmos a operação custosa Dequeue(F), uma vez, precisamos ter executado Enqueue(x, F) pelo menos uma vez, e ela é barata.

b - Cada vez que enfileiramos um elemento já temos o custo do Push. Além dele, já precisamos contabilizar o custo futuro de levá-lo de P para Q, e de removê-lo de Q. Logo, escolhemos custo $4.

Enqueue(x, F) = $4
Dequeue(F) = 0

No pior caso (só n Enqueues) teremos custo $4n \in O(n)$