Ejercicio 1

Funciones ordenadas según su velocidad de crecimiento de forma descendente:

1. 3^n
2. 2^n
3. $(\frac{3}{2})^n$
4. $2n + n^2$
5. $n * log_2(n)$
6. $n + log_2(n)$
7. $(log_2(n))^2$
8. $n^{\frac{1}{2}}$
9. $log_2(n)$
10. $(\frac{1}{3})^n$

Ejercicio 2

a.- Asumo que es Verdadero
Existe un $c > 0$ y n_0 tal que $3^n \leqslant c * 2^n$, $\forall_{n \geqslant n_0}$

$$3^n \leqslant c * 2^n$$

$$\frac{3^n}{2^n} \leqslant c$$

$$\left(\frac{3}{2}\right)^n \leqslant c$$

Absurdo, c es una constante y $\left(\frac{3}{2}\right)^n$ es una función creciente.
$\therefore$ es Falso.

b.- Asumo que es Verdadero
Existe un $c > 0$ y n_0 tal que $\frac{n}{log_2(n)} \leqslant c * log_2(n)$, $\forall_{n \geqslant n_0}$

$$\frac{n}{log_2(n)} \leqslant c * log_2(n)$$

$$\frac{n}{log_2(n) * log_2(n)} \leqslant c$$

$f_1(n) = (log_2(n))^2$ tiene menor orden de magnitud que $f_2(n) = n$.

$f_3(n) = \frac{n}{log_2(n)^2}$ es creciente.

$$\frac{n}{(log_2(n))^2} \leqslant c$$

Es un absurdo, no puedo acotar una función creciente con una constante $\therefore$ es Falso.

c.- Existe un $c > 0$ y n_0 tal que $n^{\frac{1}{2}} + 10^{20} \leqslant c * n^{\frac{1}{2}}$, $\forall_{n \geqslant n_0}$

$$n^{\frac{1}{2}} + 10^{20} \leqslant c * n^{\frac{1}{2}}$$

$$\frac{n^{\frac{1}{2}}}{n^{\frac{1}{2}}} + 10^{20} * \frac{1}{n^{\frac{1}{2}}} \leqslant c$$

$$1 + \frac{10^{20}}{n^{\frac{1}{2}}} \leqslant c$$

Y como $\frac{10^{20}}{n^{\frac{1}{2}}}$ es una función decreciente, todo el lado izquierdo es decreciente.

Por lo que la desigualdad se cumple con $c = 1 + 10^{20}$ y $n_0 = 1$.
$\therefore$ es Verdadero.

d.- Existe un $c > 0$ y n_0 tal que $n + log_2(n) \leqslant c * n$, $\forall_{n \leqslant n_0}$

$$n + log_2(n) \leqslant c * n$$

$$\frac{n}{n} + \frac{log_2(n)}{n} \leqslant c$$

$$1 + \frac{log_2(n)}{n} \leqslant c$$

Como n tiene mayor orden de magnitud que $log_2(n)$, $\frac{log_2(n)}{n}$ es decreciente.
La desigualdad se cumple con $c = 1$ Y $n_0\ = 1$.
$\therefore$ es Verdadero.

e.- $P(n) = a_k * n^k\ + \ldots + a_2 * n^2\ + a_1 * n + a_0$

$P(n) = O(nk) \iff \exists c, n_0\ (a_k * n^k\ + \ldots + a_2 * n^2\ + a_1 * n + a_0\ \leqslant c * n_k$ $, \forall_{n \geqslant n_0})$
$a_k * n^k\ + \ldots + a_2 * n^2\ + a_1 * n + a_0\ \leqslant c * n_k$
$a_k * \dfrac{n^k}{n^k}\ + \ldots + a_2 * \frac{n^2}{n^k}\ + a_1 * \frac{n}{n^k} + a_0 \frac{1}{n^k}\ \leqslant c$

La desigualdad se cumple para $c = \displaystyle\sum_{i=0}^{k} a_k$ y $n_0 = 1$.

$\therefore$ es Verdadero.

Ejercicio 3

El algoritmo randomUno() podría no terminar nunca. Por ejemplo, podría ocurrir que en un momento dado de la ejecución del código: n=5, a=$\{1, 2, 0, 0, 0\}$, i=3. Y a partir de este momento, la función ran_int(0,n-1) devuelva siempre el valor 1. Esto provocaría que el while nunca termine.

El algoritmo randomDos() podría no terminar nunca. Por ejemplo, podría darse el siguiente escenario: n=3, a=$\{0, 0, 0\}$, used=$\{true, false, false\}$, i(del segundo for)=1. Si consideramos que a partir de este momento, la función ran_int(0,n-1) devuelve siempre el valor 0, el algoritmo nunca finalizará.

El algoritmo randomTres() siempre termina su con ejecución. Para las primeras 2 lineas (int i; int[] a = new int[n];) se puede considerar que se ejecutan en un tiempo constante

$$cte_1$$

Luego, el: for(int i=0;i<n;i++) a[i]=i; se puede ecribir como:

$$\sum_{i=0}^{n-1} T_{instruccion:a[i]=i}(n)$$

Como

$$T_{instruccion:a[i]=i}(n) = cte_2$$

entonces quedaría:

$$\sum_{i=0}^{n-1} cte_2$$

La siguiente línea for(int i=1;i<n;i++) swap(a,i,ran_int(0,i-1); puede ser considerada como:

$$\sum_{i=1}^{n-1} T_{swap}(n)$$

Como

$$T_{swap}(n) = cte_3$$

, quedaría:

$$\sum_{i=1}^{n-1} cte_3$$

La instruncción “return a”lleva un

$$T(n) = cte_4$$

Todo junto, quedaría:

$$T(n) = cte_1 + \sum_{i=0}^{n-1} cte_2 + \sum_{i=1}^{n-1} cte_3 + cte_4$$

Como

$$\sum_{i=0}^{n-1} cte_2 = n * cte_2$$

entonces:

$$T(n) = cte_1 + n * cte_2 + \sum_{i=1}^{n-1} cte_3 + cte_4$$

Finalmente, como

$$\sum_{i=1}^{n-1} cte_3 = (n-1) * cte_3$$

entonces

$$T(n) = cte_1 + n * cte_2 + (n-1) * cte_3 + cte_4$$

Simplificado, quedaría:

$$T(n) = cte_1 + n * cte_2 + n * cte_3 - cte_3 + cte_4$$

Cálculo del Orden: Por regla de orden de máximos, es posible inferir que T(n) es (n). Entonces, se debe demostrar que ocurre que $T(n) \leq c * f(n)$, $\forall n \geq n_0$. Entonces, se deben dar los siguientes casos:

- $cte_1 \leq c_0 n$, $\forall n \geq n_0$, que se cumple con $c_0 = cte_1$ y $n_0 = 1$.

- $n * cte_2 \leq c_1 * n$, $\forall n \geq n_0$, que se cumple con $c_1 = cte_2$ y $n_0 = 0$.
- $n * cte_3 \leq c_2 * n$, $\forall n \geq n_0$, que se cumple con $c_2 = cte_3$ y $n_0 = 0$.
- $-cte_3 \leq c_3 * n$, $\forall n \geq n_0$, que se cumple con $c_3 = 0$ y $n_0 = 0$.
- $cte_4 \leq c_4 * n$, $\forall n \geq n_0$, que se cumple con $c_4 = cte_4$ y $n_0 = 1$.

Juntando los cinco casos anteriores, quedaría:
$cte_1 + n * cte_2 + n * cte_3 - cte_3 + cte_4 \leq (c_0 + c_1 + c_2 + c_3 + c_4) * n$, $\forall n \geq 1$ (elegimos el mayor de los n_0, en este caso, 1)

Que es lo mismo que:

$T(n) \leq (cte_1 + cte_2 + cte_3 + 0 + cte_4) * n$, $\forall n \geq 1$

Finalmente, se logra que $T(n) \leq c * n$, $\forall n \geq 1$, con c=$cte_1 + cte_2 + cte_3 + cte_4$, por lo tanto, $T(n)$ es $O(n)$.

Ejercicio 4

a.- 1) Si *f(n)* es $\log_{10} n$ y el problema es de tamaño $n = 10000$:

$$f(10000) = \log_{10} 10000 = 4 \ \textit{Operaciones}$$

$$\begin{array}{lcl} 100 \ \textit{operaciones} & \text{———————} & 1 \text{ seg.} \\ 4 \ \textit{operaciones} & \text{———————} & \text{x seg.} \end{array}$$

$$x = \frac{4 \ \textit{Operaciones}}{100 \ \textit{Operaciones}} = 0,04 \ \textit{seg.} = 40 \ \textit{ms.}$$

El algoritmo tarda 40 *ms.*

2) Si *f(n)* es $\sqrt{n}$ y el problema es de tamaño $n = 10000$:

$$f(10000) = \sqrt{10000} = 100 \ \textit{Operaciones}$$

El algoritmo tarda 1 *seg.*

b.- Si ALGO-1 tiene $T(n) = 10n^2$ podemos decir que:

$$
\begin{gathered}
10n^2 = x \\
10(2n)^2 = 10(2^2)(n^2) = (2^2)x = 4x \\
10(3n)^2 = 10(3^2)(n^2) = (3^2)x = 9x \\
...
\end{gathered}
$$

Podemos ver que si el tamaño de la entrada se duplica, ALGO-1 se vuelve 4 veces más lento y si triplicamos la entrada, ALGO-1 se vuelve 9 veces más lento.