Comenzado el	viernes, 19 de noviembre de 2021, 11:26
Estado	Finalizado
Finalizado en	viernes, 19 de noviembre de 2021, 11:53
Tiempo empleado	27 minutos 7 segundos
Puntos	16/20
Calificación	**8** de 10 (**78**%)

Pregunta **1**

Correcta

Puntúa 1 sobre 1

¿Con qué nombre general se conoce en la *Teoria de la Complejidad* a un problema para el cual sólo se conocen algoritmos cuyo tiempo de ejecución es exponencial (o sea, problemas para los que todas las soluciones conocidas son algoritmos con tiempo $O(2^n)$?

Seleccione una:

a. Problemas Inmanejables

b. Problemas Irresolubles

c. Problemas Intratables ✔ ¡Correcto!

d. Problemas Imperdonables

¡Correcto!

La respuesta correcta es:
Problemas Intratables

Pregunta **2**

Correcta

Puntúa 1 sobre 1

Suponga que dispone de cuatro algoritmos diferentes para resolver el mismo problema, y que se sabe que los tiempos de ejecución (en el peor caso) son, respectivamente: $O(n*log(n))$, $O(n^2)$, $O(n^3)$ y $O(n)$.

¿Cuál de esos tres algorimos debería elegir, suponiendo que todos hacen el mismo consumo razonable de memoria?

Seleccione una:

a. El algoritmo cuyo tiempo de ejecución es $O(n^2)$

b. El algoritmo cuyo tiempo de ejecución es $O(n^3)$

c. El algoritmo cuyo tiempo de ejecución es O(n*log(n))

d. El algoritmo cuyo tiempo de ejecución es O(n) ✔ ¡Correcto! Efectivamente, este algoritmo sería el más rápido...

¡Correcto!

La respuesta correcta es:
El algoritmo cuyo tiempo de ejecución es O(n)

Pregunta **3**

Parcialmente correcta

Puntúa 1 sobre 1

¿Cuáles de los siguientes son *factores de eficiencia comunes* a considerar en el análisis de algoritmos? (Más de una respuesta puede ser válida... marque *todas* las que considere correctas).

Seleccione una o más de una:

- [x] a. El tiempo de ejecución. ✔ ¡Correcto!
- [ ] b. La complejidad aparente del código fuente.
- [ ] c. La calidad aparente de la interfaz de usuario.
- [x] d. El consumo de memoria. ✔ ¡Correcto!

Las respuestas correctas son:
El tiempo de ejecución.,

El consumo de memoria.,

La complejidad aparente del código fuente.

Pregunta **4**

Correcta

Puntúa 1 sobre 1

Si los algoritmos de *ordenamiento simples* tienen todos un tiempo de ejecución $O(n^2)$ en el peor caso, entonces: ¿cómo explica que las mediciones efectivas de los tiempos de ejecución de cada uno sean diferentes frente al mismo arreglo?

Seleccione una:

- (•) a. La notación *Big O* rescata el término más significativo en la expresión que calcula el rendimiento, descartando constantes y otros términos que podrían no coincidir en los tres algoritmos. ✔ ¡Correcto!
- ( ) b. Los tiempos deben coincidir. Si hay diferencias, se debe a errores en los intrumentos de medición o a un planteo incorrecto del proceso de medición.
- ( ) c. La notación *Big O* no se debe usar para estimar el comportamiento en el peor caso, sino sólo para el caso medio.
- ( ) d. La notación *Big O* no se usa para medir tiempos sino para contar comparaciones u otro elemento de interés. Es un error, entonces, decir que los tiempos tienen "*orden n cuadrado*".

¡Correcto!

La respuesta correcta es:
La notación *Big O* rescata el término más significativo en la expresión que calcula el rendimiento, descartando constantes y otros términos que podrían no coincidir en los tres algoritmos.

Pregunta **5**

Correcta

Puntúa 1 sobre 1

Se tiene un algoritmo que realiza cierta cantidad de procesos sobre un conjunto de ***n*** datos y un minucioso análisis matemático ha determinado que la cantidad de procesos que el algoritmo realiza en el peor caso viene descripto por la función $f(n) = 3n^3 + 5n^2 + 2n^{1.5}$ ¿Cuál de las siguientes expresiones representa mejor el orden del algoritmo para el peor caso?

Seleccione una:

a. $O(3n^3 + 5n^2 + 2n^{1.5})$

b. $O(n^{1.5})$

c. $O(n^3 + n^2)$

d. $O(n^3)$ ¡Correcto!

¡Correcto!

La respuesta correcta es:
$O(n^3)$

Pregunta **6**

Correcta

Puntúa 1 sobre 1

¿Cuáles de las siguientes son características **correctas** del algoritmo *Shellsort*? (Más de una puede ser cierta... marque TODAS las que considere válidas)

Seleccione una o más de una:

a. El algoritmo *Shellsort* consiste en una mejora del algoritmo de *Selección Directa*, consistente en buscar iterativamente el menor (o el mayor) entre los elementos que quedan en el vector, para llevarlo a su posición correcta, pero de forma que la búsqueda del menor en cada vuelta se haga en tiempo logarítmico.

b. El algoritmo *Shellsort* consiste en una mejora del algoritmo de *Inserción Directa* (o *Inserción Simple*), consistente en armar suconjuntos ordenados con elementos a distancia $h > 1$ en las primeras fases, y terminar con $h = 1$ en la última. ¡Correcto!

c. El algoritmo *Shellsort* es complejo de analizar para determinar su rendimientos en forma matemática. Se sabe que para la serie de incrementos decrecientes usada en la implementación vista en las clases de la asignatura, tiene un tiempo de ejecución para el peor caso de $O(n^{1.5})$. ¡Correcto!

d. En el caso promedio, el algoritmo *Shellsort* es tan eficiente como el *Heapsort* o el *Quicksort*, con tiempo de ejecución $O(n*log(n))$.

¡Correcto!

Las respuestas correctas son:
El algoritmo *Shellsort* consiste en una mejora del algoritmo de *Inserción Directa* (o *Inserción Simple*), consistente en armar suconjuntos ordenados con elementos a distancia $h > 1$ en las primeras fases, y terminar con $h = 1$ en la última.,

El algoritmo *Shellsort* es complejo de analizar para determinar su rendimientos en forma matemática. Se sabe que para la serie de incrementos decrecientes usada en la implementación vista en las clases de la asignatura, tiene un tiempo de ejecución para el peor caso de $O(n^{1.5})$.

Pregunta **7**

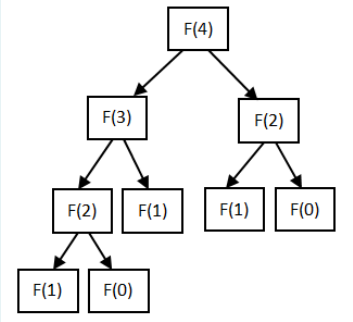
Correcta

Puntúa 1 sobre 1

Sabemos que la recursión es una de las técnicas o estrategias básicas para el planteo de algoritmos y que una de sus ventajas es que permite el diseño de algoritmos compactos y muy claros, aunque al costo de usar algo de memoria extra en el stack segment y por consiguiente también un algo de tiempo extra por la gestión del stack. Algunos problemas pueden plantearse en forma directa procesando recursivamente diversos subproblemas menores. Tal es el caso de la *sucesión de Fibonacci*, en la cual cada término se calcula como la suma de los dos inmediatamente anteriores [esto se expresa con la siguiente relación de recurrencia: *F(n) = F(n-1) + F(n-2)* con *F(0) = 0* y *F(1) = 1*]. La función sencilla que mostramos a continuación que calcula el término n-ésimo de la sucesión en forma recursiva:

```
def fibo(n):
    if n <= 1:
        return 1
    return fibo(n-1) + fibo(n-2)
```

Sin embargo, un inconveniente adicional es que la *aplicación directa* de *dos o más invocaciones recursivas* en el planteo de un algoritmo podría hacer que un mismo subproblema se resuelva más de una vez, incrementando el tiempo total de ejecución. Por ejemplo, para calcular fibo(4) el árbol de llamadas recursivas es el siguiente:



y puede verse que en este caso, se calcula *2 veces* el valor de *fibo(2)*, *3 veces* el valor de *fibo(1)* y *2 veces* el valor de *fibo(0)*... con lo que la cantidad de llamadas a funciones para hacer *más de una vez el mismo trabajo* es de ***7*** (= 2 + 3 + 2).

Suponga que se quiere calcular el valor del sexto término de la sucesión. Analice el árbol de llamadas recursivas que se genera al hacer la invocación ***t = fibo(6)*** **¿Cuántas veces en total la función *fibo()* se invoca para hacer más de una vez el mismo trabajo, SIN incluir en ese conteo a las invocaciones para obtener *fibo(0)* y *fibo(1)*?**

Seleccione una:

- a. 11
- b. 10 ✔ ¡Correcto! Efectivamente: *fibo(4)* se invoca 2 veces, *fibo(3)* se invoca 3 veces y *fibo(2)* se invoca 5 veces...
- c. 12
- d. 0

¡Correcto!

La respuesta correcta es: 10

Pregunta **8**

Correcta

Puntúa 1 sobre 1

¿Cuáles de las siguientes afirmaciones son correctas respecto del algoritmo de backtracking que se propuso en clases para el problema de las Ocho Reinas (que mostramos más abajo)? (Más de una respuesta puede ser cierta, por lo que marque todas las que considere correctas).

```
def intend(col):
    global rc, qr, qid, qnd

    fil, res = -1, False
    while not res and fil != 7:
        res = False
        fil += 1
        di = col + fil
        dn = (col - fil) + 7
        if qr[fil] and qid[di] and qnd[dn]:
            rc[col] = fil
            qr[fil] = qid[di] = qnd[dn] = False

            if col < 7:
                res = intend(col + 1)
                if not res:
                    qr[fil] = qid[di] = qnd[dn] = True
                    rc[col] = -1
            else:
                res = True
    return res
```

Seleccione una o más de una:

- [x] a. Una invocación de la forma intend(5) significa que se va a intentar posicionar en alguna fila a la reina de la columna 5. ¡Correcto!
- [x] b. Para mostrar todas las posibles soluciones básicas o simétricas, básicamente solo hay que cambiar el ciclo while y la bandera usada para cortar al encontrar una solución, por un for que obligatoriamente recorra las ocho filas en cada una de las ocho columnas. ¡Correcto!
- [ ] c. El algoritmo de resolución del problema nunca analiza las diagonales del tablero (solo las filas y las columnas), y de allí deduce las soluciones correctas.
- [ ] d. Se usan cuatro arreglos unidimensionales para llevar el control de las casillas disponibles y un quinto arreglo para representar la solución.

¡Correcto!

Las respuestas correctas son:
Una invocación de la forma intend(5) significa que se va a intentar posicionar en alguna fila a la reina de la columna 5.,

Para mostrar todas las posibles soluciones básicas o simétricas, básicamente solo hay que cambiar el ciclo while y la bandera usada para cortar al encontrar una solución, por un for que obligatoriamente recorra las ocho filas en cada una de las ocho columnas.

Pregunta **9**

Correcta

Puntúa 1 sobre 1

El cálculo del valor a^n (para simplificar, asumimos $a > 0$ y $n >= 0$ y sabiendo que si $n = 0$ entonces $a^0 = 1$) es igual a multiplicar n veces el número a por sí mismo. Por caso, $5^3 = 5 * 5 * 5 = 125$. Sabiendo esto, ¿cuál de las siguientes sería una *definición recursiva* matemáticamente correcta de la operación *potencia(a, n)*?

Seleccione una:

a.

potencia(a, n) (a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a, n) si n > 0

b.

potencia(a, n) (a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a, n-1) si n > 0

¡Correcto!

c.

potencia(a, n) (a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a-1, n) si n > 0

d.

potencia(a, n) (a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a + potencia(a, n-1) si n > 0

¡Correcto!

La respuesta correcta es:

potencia(a, n) (a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a, n-1) si n > 0

Pregunta **10**

Correcta

Puntúa 1 sobre 1

El producto de dos números *a* y *b* mayores o iguales cero, es en última instancia *una suma*: se puede ver como sumar *b* veces el número *a*, o como sumar *a* veces el número *b*. Por ejemplo: 5 * 3 = 5 + 5 + 5 o bien 5 * 3 = 3 + 3 + 3 + 3 + 3. Sabiendo esto, se puede intentar hacer una definición recursiva de la operación producto(a, b) [a >= 0, b >= 0], que podría ser la que sigue:

$$\textbf{producto(a, b)} = \begin{cases} 0 & \text{si } a == 0 \text{ o } b == 0 \\ a + producto(a, b-1) & \text{si } a > 0 \text{ y } b > 0 \end{cases}$$

[a y b enteros, a >= 0 y b >= 0]

¿Es correcto el siguiente planteo de la función *producto(a, b)*?

```
def producto(a, b):
    if a == 0 or b == 0:
        return 0
    return a + producto(a, b-1)
```

Seleccione una:

- a. Sí. Es correcto. ✔ ¡Correcto!
- b. No. La propia definición previa del concepto de producto es incorrecta: un producto es una multiplicación, y no una suma...
- c. No. La función sugerida siempre retornará 0, sean cuales fuesen los valores de *a* y *b*.
- d. No. La última línea debería decir *return a + producto(a-1, b-1)* en lugar de *return a + producto(a, b-1)*.

¡Correcto!

La respuesta correcta es:
Sí. Es correcto.

Pregunta **11**

Incorrecta

Puntúa 0 sobre 1

¿Cuáles de las siguientes afirmaciones son correcta respecto de las características de la recursividad? (Más de una puede ser cierta, por lo que marque todas las que considere válidas):

Seleccione una o más de una:

- a. La recursividad puede ocupar más memoria que un algoritmo iterativo equivalente.
- b. La recursividad utiliza memoria de stack. ✔ ¡Correcto! Es un proceso de llamadas a funciones, por lo que el control se hace en base al stack segment.
- c. La recursividad utiliza memoria ROM.
- d. La recursividad siempre ocupa más memoria que el algoritmo iterativo equivalente. ✖ Incorrecto... En muchos casos efectivamente ocurre eso, pero en muchos otros los algoritmos recursivos consumen memoria en el mismo orden que los algoritmos no recursivos equivalentes...

Revise sus notas de clase y la Ficha 26, especialmente la sección dedicada al seguimiento de la recursividas.

Las respuestas correctas son:
La recursividad utiliza memoria de stack.,

La recursividad puede ocupar más memoria que un algoritmo iterativo equivalente.

Pregunta **12**

Correcta

Puntúa 1 sobre 1

¿Cuál es la cantidad de niveles del *árbol de invocaciones recursivas* que se genera al ejecutar el *Quicksort* para ordenar un arreglo de *n* elementos, en el ***peor caso***? (Es decir: ¿Cuál es la *altura* de ese árbol en ese peor caso?)

Seleccione una:

a. `Altura = log(n)`

b. `Altura = n`^{2}

c. `Altura = n` ¡Correcto!

d. `Altura = n * log(n)`

¡Correcto!

La respuesta correcta es:
`Altura = n`

Pregunta **13**

Correcta

Puntúa 1 sobre 1

¿Qué hace el siguiente programa (que incluye una función recursiva)?

```
def repetir(numero, rep, res):
    if rep != 0:
        res.append(numero)
        repetir(numero, (rep - 1), res)


v = [1, 3, 3, 7]
res = []
for i in range(len(v)):
   repetir(v[i], 2, res)
print(res)
```

Seleccione una:

a. Produce un error de ejecución, porque la función recursiva no tiene valor de retorno.

b. Genera y muestra un nuevo vector res con los números que en el vector original v eran negativos.

c. Genera y muestra un nuevo vector res con cada elemento del vector original v repetido dos veces. ¡Correcto!

d. Produce un error de ejecución al intentar el print() final, ya que el vector res no es generado en ningún momento.

¡Correcto!

La respuesta correcta es:
Genera y muestra un nuevo vector res con cada elemento del vector original v repetido dos veces.

Pregunta **14**

Correcta

Puntúa 1 sobre 1

¿Cuáles de las siguientes son CIERTAS en relación al sistema de coordenadas de pantalla?

Seleccione una o más de una:

- [x] a. Las filas de pantalla o de ventana con número de orden más bajo, se encuentran más cerca del borde superior que las filas con número de orden más alto. ¡Correcto!
- [ ] b. El origen del sistema de coordenadas se encuentra en el punto inferior izquierdo de la pantalla o de la ventana que se esté usando.
- [x] c. El origen del sistema de coordenadas se encuentra en el punto superior izquierdo de la pantalla o de la ventana que se esté usando. ¡Correcto!
- [ ] d. Las filas de pantalla o de ventana con número de orden más alto, se encuentran más cerca del borde superior que las filas con número de orden más bajo.

¡Correcto!

Las respuestas correctas son:
El origen del sistema de coordenadas se encuentra en el punto superior izquierdo de la pantalla o de la ventana que se esté usando.,

Las filas de pantalla o de ventana con número de orden más bajo, se encuentran más cerca del borde superior que las filas con número de orden más alto.

Pregunta **15**

Correcta

Puntúa 1 sobre 1

Dado un arreglo de *n* componentes... ¿qué significa decir que en el peor caso la cantidad de comparaciones que realiza el algoritmo de *búsqueda secuencial* es O(n) (o sea: del orden de n)?

Seleccione una:

- a. Significa que en el peor caso el algoritmo hará siempre más de n comparaciones.
- b. Significa que en el peor caso el algoritmo no hará ninguna comparación.
- c. Significa que en el peor caso el algoritmo hará n comparaciones. ¡Correcto!
- d. Significa que en el peor caso el algoritmo siempre hará menos de n comparaciones.

¡Correcto!

La respuesta correcta es:
Significa que en el peor caso el algoritmo hará n comparaciones.

Pregunta **16**

Incorrecta

Puntúa 0 sobre 1

Considere la siguiente función (vista en clases) basada en **backtracking** para resolución del Problema de las Ocho Reinas, e indique cuál de las opciones que se muestran es correcta:

```
def intend(col):
    global rc, qr, qid, qnd

    fil, res = -1, False
    while not res and fil != 7:
        res = False
        fil += 1
        di = col + fil
        dn = (col - fil) + 7
        if qr[fil] and qid[di] and qnd[dn]:
            rc[col] = fil
            qr[fil] = qid[di] = qnd[dn] = False

            if col < 7:
                res = intend(col + 1)
                if not res:
                    qr[fil] = qid[di] = qnd[dn] = True
                    rc[col] = -1
            else:
                res = True
    return res
```

Seleccione una:

a. La función es correcta y funciona sin problemas.

b. La función no es correcta porque la diagonal normal debe almacenar valores **(columna + fila)**.

c. La función no es correcta porque **qr** debe señalar las columnas disponibles y no las filas.

d. La función no es correcta porque no debe asignarse a **rc[col]** el valor de -1. ✖ Incorrecto... Esa asignación es válida en el algoritmo...

Revise el funcionamiento y la explicación de este algoritmo en la Ficha 28.

La respuesta correcta es:
La función es correcta y funciona sin problemas.

Pregunta **17**

Incorrecta

Puntúa 0 sobre 1

¿Cuáles de las siguientes son ciertas respecto de la estrategia de resolución de problemas conocida como *Algoritmos Ávidos*?

Seleccione una o más de una:

- a. Normalmente, una ventaja de intentar aplicar un algoritmo ávido es que si bien la validez de la regla local debe ser demostrada formalmente, eso es comúnmente fácil de hacer. ✖ Incorrecto... Justamente, es lo contrario... Esa regla no suele ser sencilla de probar...
- b. Se trata de un planteo basado en explorar y analizar cada camino posible para resolver un problema, de forma que si detecta que algún camino conduce a una solución incorrecta, se abandone ese camnio y se regrese para explorar otros posibles, hasta dar con la solución correcta.
- c. Se trata de un planteo basado en identificar una regla local que intuitivamente parece correcta, y aplicarla una y otra vez sin volver atrás ni analizar caminos alternativos, hasta llegar a la solución del problema global. ✔ ¡Correcto!
- d. Normalmente, una ventaja de un algoritmo ávido es que, si la regla local aplicada es correcta, suele ser sencillo de plantear y de comprender, además de veloz para ejecutar.

Revise la Ficha 28, página 578 y siguientes.

Las respuestas correctas son:
Se trata de un planteo basado en identificar una regla local que intuitivamente parece correcta, y aplicarla una y otra vez sin volver atrás ni analizar caminos alternativos, hasta llegar a la solución del problema global.,

Normalmente, una ventaja de un algoritmo ávido es que, si la regla local aplicada es correcta, suele ser sencillo de plantear y de comprender, además de veloz para ejecutar.

Pregunta **18**

Incorrecta

Puntúa 0 sobre 1

Considere el problema de las *Ocho Reinas* presentado en clases. ¿Cuáles de las siguientes afirmaciones son ***ciertas*** en relación a las ***diagonales inversas del tablero*** en el cual deben colocarse la reinas, suponiendo que el tablero es el normal del ajedrez, de 8 * 8? (Más de una respuesta puede ser cierta, por lo que marque todas las que considere correctas...)

Seleccione una o más de una:

- a. El valor de la suma entre el número de columna y el número de fila de cada componente de una diagonal inversa, es un número constante para cada diagonal, y los posibles valores están en el intervalo [0..14] ✔ ¡Correcto!
- b. El valor de la resta entre el número de columna y el número de fila de cada componente de una diagonal inversa, es un número constante para cada diagonal, y los posibles valores están en el intervalo [-7..7]
- c. Las diagonales inversas pueden representarse con un arreglo *qid* de 15 componentes, en el que cada diagonal cuyos elementos tengan el mismo valor *(col - fil)*, se haga coincidir el casillero *qid[(col - fil) + 7]*. ✖ Incorrecto... esa afirmación es cierta pero sólo para las diagonales normales...
- d. Las diagonales inversas pueden representarse con un arreglo *qid* de 15 componentes, en el que cada diagonal cuyos elementos tengan el mismo valor *(col + fil)*, se haga coincidir el casillero *qid[(col + fil)]*.

Incorrecto... revise la Ficha de clase 28, página 585...

Las respuestas correctas son:
El valor de la suma entre el número de columna y el número de fila de cada componente de una diagonal inversa, es un número constante para cada diagonal, y los posibles valores están en el intervalo [0..14],

Las diagonales inversas pueden representarse con un arreglo *qid* de 15 componentes, en el que cada diagonal cuyos elementos tengan el mismo valor *(col + fil)*, se haga coincidir el casillero *qid[(col + fil)]*.

Pregunta **19**

Correcta

Puntúa 1 sobre 1

¿Qué se entiende (esencialmente) por una *gráfica o figura fractal*?

Seleccione una:

- a. Es aquella que se compone de versiones más simples y pequeñas de la misma figura original. ✔ ¡Correcto!
- b. Es aquella que se compone sólo por cuadrados, círculos y triángulos combinados.
- c. Es aquella que se compone de fragmentos distintos de diversas figuras, conformando una imagen sin un patrón definido ni obvio.
- d. Es aquella que se compone a partir de la combinación de millares de pequeños puntos que al ser mirados desde cierta distancia producen en forma completa la imagen compuesta.

¡Correcto!

La respuesta correcta es: Es aquella que se compone de versiones más simples y pequeñas de la misma figura original.

Pregunta **20**

Correcta

Puntúa 1 sobre 1

¿Cuáles de las siguientes afirmaciones ***son ciertas*** en relación a conceptos asociados con la *recursividad*?

Seleccione una o más de una:

- a. Cada instancia recursiva que es ejecutada, almacena dos grupos de datos en el stack segment: la dirección de retorno (a la que se debe regresar cuando termine la ejecución de esa instancia) y las variables locales que esa instancia de la función haya creado. ✔ ¡Correcto! De hecho, estos datos son almacenados en el stack segment por toda función que sea invocada, haya o no recursividad implicada.
- b. Una función recursiva no puede incluir más de una invocación a si misma en su bloque de acciones.
- c. Si una función es recursiva, entonces no debe incluir ningún ciclo en su bloque de acciones.
- d. A medida que se desarrolla la cascada de invocaciones recursivas, el stack segment se va llenando para darle soporte a cada instancia recursiva, y luego, cuando una instancia logra finalizar y se produce el proceso de vuelta atrás, el stack segment comienza a vaciarse. ✔ ¡Correcto!

¡Correcto!

Las respuestas correctas son:
Cada instancia recursiva que es ejecutada, almacena dos grupos de datos en el stack segment: la dirección de retorno (a la que se debe regresar cuando termine la ejecución de esa instancia) y las variables locales que esa instancia de la función haya creado.,

A medida que se desarrolla la cascada de invocaciones recursivas, el stack segment se va llenando para darle soporte a cada instancia recursiva, y luego, cuando una instancia logra finalizar y se produce el proceso de vuelta atrás, el stack segment comienza a vaciarse.

◄ Planillas de Notas y Condición Final (Provisorias)

Ir a...