Comenzado el	jueves, 17 de diciembre de 2020, 08:26
Estado	Finalizado
Finalizado en	jueves, 17 de diciembre de 2020, 09:00
Tiempo empleado	34 minutos 12 segundos
Puntos	22/25
Calificación	**9** de 10 (**89**%)

Pregunta **1**

Finalizado

Puntúa 1 sobre 1

¿Cuál es la diferencia entre el *peor caso* y el *caso promedio* en el análisis de algoritmos?

Seleccione una:

a.
El *peor caso* es la configuración de datos de entrada más favorable para el algoritmo, mientras que el *caso promedio* describe una configuración de datos pensada para desfavorecer al algoritmo.

b.
El *peor caso* es la configuración de datos de entrada más desfavorable para el algoritmo, mientras que el *caso promedio* describe una configuración de datos pensada para favorecer al algoritmo.

c.
El *peor caso* es la configuración de datos de entrada más desfavorable para el algoritmo, mientras que el *caso promedio* describe una configuración aleatoria de datos (no pensada ni para favorecer ni para desfavorecer al algoritmo).

d.
El *peor caso* es la configuración de datos de entrada más favorable para el algoritmo, mientras que el *caso promedio* describe una configuración aleatoria de datos (no pensada ni para favorecer ni para desfavorecer al algoritmo)

Pregunta **2**

Finalizado

Puntúa 1 sobre 1

Suponga que se desea un programa que cargue dos números, y muestre el producto y el cociente entre ellos ¿Cuál es el problema con el siguiente programa desarollado en base a funciones que manejan variables globales? (ver Ficha 09),

```
__author__ = 'Cátedra de AED'


def cargar():
    global a, b
    a = int(input('A: '))
    b = int(input('B: '))


def calcular():
    c = a * b
    d = a / b

# script principal...
cargar()
calcular()

print('Producto:', c)
print('Cociente:', d)
```

Seleccione una:

- a. Lanza un error de intérprete en la carga de datos: la carga de datos DEBE hacerse en el script principal y no en una función separada.
- b. El programa lanza un error al ser ejecutado, en el *script principal*: no reconoce las variables *c* y *d* ya que fueron definidas en la función *calcular()*, pero como variables locales a ella.
- c. Lanza un error de intérprete en el mismo iniciio: todas las variables debieron ser definidas (asignadas) en el script principal.
- d. No lanza errores de intérprete, pero funciona mal: en el *script principal* están al revés las invocaciones a las funciones *cargar()* y *calcular()*: primero debería invocar a *calcular()* y luego a *cargar()*.

Pregunta **3**

Finalizado

Puntúa 1 sobre 1

Para cada una de las variables predefinidas de uso global de Python de la primera columna, seleccione la definición que le corresponde o que mejor se le ajuste.

__all__	Los elementos a importar desde un paquete.
__doc__	Todos los docstrings que contiene el elemento.
__author__	El nombre del autor del elemento.
__name__	El nombre de un módulo o el valor literal '__main__'.

Pregunta **4**

Finalizado

Puntúa 1 sobre 1

Dada la siguiente función recursiva del factorial:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Seleccione cuál de las siguientes opciones es la correcta, si se le pasa como parámetro actual a esta función el valor -4.

Seleccione una:

a.
La función devuelve como resultado: -24.

b.
La función devuelve como resultado: 24.

c.
La función no tiene definido ningún caso base, que es aquel caso que se resuelve sin recursividad, y por lo tanto no está bien planteada.

d.
Así como está planteada, si entra el -4 (o cualquier otro negativo) como parámetro, la condición de corte nunca es cierta y se producirá un error de stack overflow en algún momento

Pregunta **5**

Finalizado

Puntúa 1 sobre 1

¿Cuál de las siguientes funciones creará una tupla conteniendo exclusivamente los números pares divisibles por 3 y por 7 al mismo tiempo, del intervalo [0, n]?

Seleccione una:

a.
```
def prueba(n):
    r = ()
    for i in range(1, n+1, 21):
        r += i,
    return r
```

b.
```
def prueba(n):
    r = ()
    for i in range(0, n+1, 21):
        r += i,
    return r
```

c.
```
def prueba(n):
    r = ()
    for i in range(2, 3*(n+1), 7):
        r += i,
    return r
```

d.
```
def prueba(n):
    r = ()
    for i in range(0, n+1, 42):
        r += i,
    return r
```

Pregunta **6**

Finalizado

Puntúa 1 sobre 1

El siguiente programa crea y carga un arreglo *temp* con *n* valores de temperaturas medidas en diferentes momentos y luego procesa ese arreglo mediante la función *amplitud()* ¿Qué hace exactamente esa función?

```
__author__ = 'Cátedra de AED'


def read(temp):
    n = len(temp)
    for i in range(n):
        temp[i] = int(input('Temperatura[' + str(i) + ']: '))


def amplitud(temp):
    n = len(temp)
    my = mn = temp[0]
    for i in range(1, n):
        if temp[i] > my:
            my = temp[i]
        elif temp[i] < mn:
            mn = temp[i]

    return my - mn


def test():
    n = int(input('Cantidad de temperaturas a cargar: '))
    temp = n * [0.0]
    read(temp)

    d = amplitud(temp)

    print('Amplitud térmica:', d)


if __name__ == '__main__':
    test()
```

Seleccione una:

a.
Calcula y retorna el promedio entre la mayor y la menor temperatura del arreglo *temp*.

b.
Calcula y retorna la menor temperatura del arreglo *temp*.

c.
Calcula y retorna la diferencia entre la mayor y la menor temperatura del arreglo *temp*.

d.
Calcula y retorna la la mayor temperatura del arreglo *temp*.

Pregunta **7**

Finalizado

Puntúa 1 sobre 1

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

Seleccione una o más de una:

- [ ] a.
  El origen del sistema de coordenadas se encuentra en el punto inferior izquierdo de la pantalla o de la ventana que se esté usando.

- [ ] b.
  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.

- [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.

- [x] d.
  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 **8**

Finalizado

Puntúa 1 sobre 1

¿Cuál de los siguientes scripts ***NO*** creará una tupla conteniendo exclusivamente los caracteres de la palabra 'Python', en ese mismo orden'? (Repetimos: la pregunta es cuál de todos ***NO*** creará la tupla pedida...)

Seleccione una:

- a.
  ```
  t1 = tuple('Python')
  ```

- b.
  ```
  t2 = 'P', 'y', 't', 'h', 'o', 'n'
  ```

- c.
  ```
  t4 = ()
  for c in 'Python':
      t4 += c,
  ```

- d.
  ```
  Python = 'Java'
  t5 = tuple(Python)
  ```

- e.
  ```
  t3 = ('P', 'y', 't', 'h', 'o', 'n')
  ```

Pregunta **9**

Finalizado

Puntúa 1 sobre 1

¿Por qué motivo el algoritmo Bubblesort para ordenamento de un arreglo usa una *bandera de corte* en la versión presentada en las fichas de clase?

Seleccione una:

a.
La bandera de corte se usa para garantizar que el arreglo quede ordenado.

b.
La bandera de corte se usa para determinar si el ordenamento debe hacerse de manor a mayor (bandera = True) o de mayor a menor (bandera = False)

c.
La bandera de corte se usa para terminar el proceso apenas se detecte que en la pasada actual no hubo intercambios, para ahorrar tiempo.

d.
No es cierto que la versión vista en clases use una bandera de corte.

Pregunta **10**

Finalizado

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.

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

Pregunta **11**

Finalizado

Puntúa 1 sobre 1

Suponga que se quiere cargar por teclado el valor una temperatura, validando que la misma esté entre -70 grados y 50 grados, y se plantea un ciclo *while forzado a operar en forma [1, N],* como muestra el esquema, para hacerlo:

```
__author__ = 'Catedra de AED'

t = 51
while ?????? :
    t = int(input('Cargue temperatura (entre -70 y 50, por favor):
'))
```

¿Cuál de las siguientes debería ser la condición a incluir en ese ciclo *while* (en el lugar donde figuran los signos de pregunta de color rojo) para que el *valor vuelva a cargarse* en caso de haber sido mal ingresado?

Seleccione una:

a.
`t < -70 and t > 50`

b.
`t >= -70 or t <= 50`

c.
`t >= -70 and t <= 50`

d.
`t < -70 or t > 50`

Pregunta **12**

Finalizado

Puntúa 1 sobre 1

Suponga que se tiene una pila *p* con capacidad para almacenar números enteros y que las operaciones básicas *pop()*, *peek()* y *push()* están correctamente implementadas en el módulo *stack.py* presentado en clases. Suponga que se han insertado los siguientes valores: [8 - 6 - 5 - 3 - 7] (el número 8 es el valor del frente o tope de la Pila). ¿Cuál de las siguientes secuencias de instrucciones permite retirar el valor 5 de la pila, pero dejando el 6 y 8 nuevamente arriba? (es decir: ¿cuál de las siguientes secuencias, dejaría la pila *p* en el estado [8 - 6 - 3 - 7]?)

Seleccione una:

a.
```
n1 = stack.pop(p)
n2 = stack.pop(p)
x = stack.peek(p)
stack.push(p, n2)
stcak.push(p, n1)
```

b.
```
n1 = stack.pop(p)
n2 = stack.pop(p)
x = stack.pop(p)
```

c.
```
n1 = stack.pop(p)
n2 = stack.pop(p)
x = stack.pop(p)
stack.push(p, n2)
stack.push(p, n1)
```

d.
```
n1 = stack.pop(p)
n2 = stack.pop(p)
x = stack.pop(p)
stack.push(p, n1)
stack.push(p, n2)
```

Pregunta **13**

Finalizado

Puntúa 0 sobre 1

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

Seleccione una:

a.
Es aquella que se compone sólo por cuadrados, círculos y triángulos combinados.

b. Es aquella que se compone de versiones más simples y pequeñas de la misma figura original.

c.
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.

d.
Es aquella que se compone de fragmentos distintos de diversas figuras, conformando una imagen sin un patrón definido ni obvio.

Pregunta **14**

Finalizado

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)

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

Pregunta **15**

Finalizado

Puntúa 1 sobre 1

¿Cuál es el efecto del operador or ("o lógico") en una condición?

Seleccione una:

a.
La condición es verdadera si todas las proposiciones son falsas.

b.
La condición es verdadera si al menos una de las proposiciones es verdadera.

c.
La condición es verdadera si y sólo si todas las proposiciones son verdaderas.

d.
La condición es verdadera si y sólo si una única proposición es verdadera. Si más de una es verdadera, la salida es False.

Pregunta **16**

Finalizado

Puntúa 1 sobre 1

En un archivo binario generado y mantenido por un programa Python, en ocasiones es necesario almacenar únicamente registros de tamaño fijo, es decir, registros que tengan siempre la misma cantidad de bytes. ¿Cuál de las siguientes opciones explica mejor la razón por la que ese tamaño fijo puede ser necesario?:

Seleccione una:

a.
Resulta conveniente que un archivo binario en Python que almacena registros mediante el mecanismo de serialización utilice registros de ancho fijo, para prever la posibilidad de que ante una modificación se sobrescriban datos del registro siguiente o se dejen bytes "huérfanos" impidiendo la correcta lectura serializada de allí en adelante.

b.
Los archivos binarios en Python solo admiten registros de ancho fijo cuando se quieren hacer sobre ellos operaciones de lectura y escritura.

c.
Los archivos binarios en Python requieren registros de ancho fijo cuando se los abre en modo "wb".

d.
Siempre que se utilice el mecanismo de serialización es obligatorio hacerlo mediante registros de ancho fijo, de lo contrario las funciones del módulo "pickle" no pueden realizar las conversiones de tipo correspondientes.

Pregunta **17**

Finalizado

Puntúa 1 sobre 1

¿Cuál de las siguientes expone correctamente la estrategia general que debe llevar a cabo un *proceso de alta de registros en un archivo, suponiendo que NO se admiten registros con clave repetida* en ese archivo?

Seleccione una:

a.
1) Abrir el archivo *m* en modo 'a+b'. 2) Determinar si el archivo ya contiene o no a un registro con la misma clave que el registro r que se quiere agregar. 3) Si ya existe un registro con esa clave, rechazar el alta. Si no existe, asegurarse de marcar el registro como *eliminado* (*activo = False*), y grabar finalmente el registro en el archivo.

b.
1) Abrir el archivo *m* en modo 'a+b'. 2) Asegurarse de marcar el registro como *no eliminado* (*activo = True*), y grabar finalmente el registro en el archivo.

c.
1) Abrir el archivo *m* en modo 'a+b'. 2) Determinar si el archivo ya contiene o no a un registro con la misma clave que el registro r que se quiere agregar. 3) Si ya existe un registro con esa clave, rechazar el alta. Si no existe, asegurarse de marcar el registro como *no eliminado* (*activo = True*), y grabar finalmente el arrchivo.

d.
1) Abrir el archivo *m* en modo 'ab'. 2) Determinar si el archivo ya contiene o no a un registro con la misma clave que el registro r que se quiere agregar. 3) Si ya existe un registro con esa clave, rechazar el alta. Si no existe, asegurarse de marcar el registro como *no eliminado* (*activo = True*), y grabar finalmente el registro en el archivo.

Pregunta **18**

Finalizado

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 no hará ninguna comparación.

b.
Significa que en el peor caso el algoritmo siempre hará menos de n comparaciones.

c.
Significa que en el peor caso el algoritmo hará n comparaciones.

d.
Significa que en el peor caso el algoritmo hará siempre más de n comparaciones.

Pregunta **19**

Finalizado

Puntúa 1 sobre 1

Suponga que *m* es una variable tipo *file object*, que representa a un archivo ya abierto (o que será abierto cuando corresponda). ¿A qué posición (interna o externa) del archivo quedará apuntando el *file pointer* de ese archivo luego de las siguientes invocaciones al método *seek()*? (Cuando lo necesite, *suponga que el tamaño del archivo es de 20 bytes* y calcule las respuestas a partir de este dato)

`m.seek(5, io.SEEK_END)`	Apuntará al byte número 25.
`m.seek(-5, io.SEEK_SET)`	Provocará un error de runtime (file pointer negativo).
`m.seek(5, io.SEEK_SET)`	Apuntará al byte número 5.
`m.seek(-5, io.SEEK_END)`	Apuntará al byte número 15.
`m.seek(10, io.SEEK_SET)` `m.seek(-3, io.SEEK_CUR)`	Apuntará al byte número 7.

Pregunta **20**

Finalizado

Puntúa 1 sobre 1

Sabemos que en el *algoritmo de Shell* se termina haciendo una última pasada sobre el arreglo con incremento de compración **h = 1** ¿Cuál de las siguientes es cierta respecto de esa última pasada con **h = 1**?

Seleccione una:

a.
La pasada con h = 1 es obligatoria pero no es necesario que sea la última.

b.
Con h = 1 el algoritmo se convierte en un ordenamiento por inserción simple, y sólo entonces garantiza que el arreglo quede ordenado.

c.
No es obligatorio que lo haga, pero favorece un ordenamiento más rápido.

d.
Con h = 1 el algoritmo sólo controla si el arreglo está ya ordenado, y en caso de no estarlo relanza el proceso con otra sucesión de valores h.

Pregunta **21**

Finalizado

Puntúa 1 sobre 1

Para cada uno de los algoritmos o procesos listados en la columna de la izquierda, seleccione la expresión de orden que mejor describe su tiempo de ejecución en el peor caso.

Dado un arreglo ya ordenado v con n componentes, buscar un valor x aplicando búsqueda binaria.	O(log(n))
Dado un arreglo v con n componentes, ordenarlo de menor a mayor mediante el algoritmo de selección directa.	O(n^2) (léase: "orden n al cuadrado")
Dadas dos matrices de orden n*n, obtener la matriz producto aplicando el método tradicional.	O(n^3) (léase: "orden n al cubo")
Dado un arreglo v con n componentes, buscar un valor x aplicando búsqueda secuencial.	O(n)
Dado un arreglo v con n componentes, acceder y cambiar el valor del casillero v[k] (con 0 <= k <= n-1).	O(1)

Pregunta **22**

Finalizado

Puntúa 1 sobre 1

Para cada uno de los algoritmos básicos y/o técnicas de procesamiento generales que se indican en la columna de la izquierda, seleccione la expresión en notación *Big O* que mejor expresa el tiempo de ejecución de ese algoritmo en el peor caso:

Búsqueda secuencial en un arreglo (ordenado o desordenado).	O(n)
Ordenamiento rápido (Quicksort) (Considere aquí el tiempo para el caso promedio).	O(n*log(n))
Ordenamiento por selección directa.	O(n^2) (n al cuadrado)
Búsqueda binaria en un arreglo ya ordenado.	O(log(n))
Multiplicación de matrices cuadradas de tamaño n*n.	O(n^3) (n al cubo)
Acceso directo a un casillero de un vector.	O(1)

Pregunta **23**

Finalizado

Puntúa 0 sobre 1

¿Cuáles de las siguientes afirmaciones son correctas respecto de las características de los algoritmos ávidos? (Más de una puede ser cierta, por lo que marque todas las que considere válidas):

Seleccione una o más de una:

- a. Aplican una regla formal global o general en cada paso proceso, buscando lograr una solución óptima local.
- b. La generación de gráficos fractales es un ejemplo de este tipo de algoritmos.
- c. Los algoritmos para obtener el camino más corto entre dos nodos de un grafo, o para obtener el árbol de expansión mínimo de un grafo, son ejemplo de este tipo de algoritmos.
- d. El Quicksort es un ejemplo de este tipo de algoritmos.
- e. Aplican una regla intuitivamente válida en cada paso local del proceso, buscando lograr una solución global óptima.

Pregunta **24**

Finalizado

Puntúa 0 sobre 1

Suponga que para un problema *p* dado, cuyo volumen de entrada es *n*, se conocen tres algoritmos diferentes *a1*, *a2* y *a3* para resolverlo. Suponga que los tiempos de ejecución de estos tres algoritmos son t(a1) = $O(2^n)$, t(a2) = $O(n^4)$ y t(a3) = $O(n^2*\log(n))$ *¿Cuáles de las siguientes afirmaciones son ciertas respecto del problema p desde el punto de vista de la Teoría de la Complejidad Computacional*? (Más de una respuesta puede ser cierta... marque todas las que considere correctas...)

Seleccione una o más de una:

- a. Los algoritmos a2 y a3 tienen tiempos de ejecución aceptables para la teoría, por lo tanto p no es un problema intratable.
- b. El problema p es intratable, ya que el algoritmo a1 ejecuta en tiempo exponencial (considerado impráctico por la teoría).
- c. El problema p no es intratable, ya que al menos uno de los algoritmos que se conocen para resolverlo ejecuta en tiempo polinomial o subpolinomial.
- d. No hay forma de decidir si p es intratable o no, ya que algunos de sus algoritmos ejecutan en tiempo exponencial y otros lo hacen en tiempo polinomial o subpolinomial.

Pregunta **25**

Finalizado

Puntúa 1 sobre 1

¿Cuál es la mejora esencial que el algoritmo *Quicksort* realiza sobre el algoritmo *Bubblesort* o *Burbuja*?

Seleccione una:

a.
En las versiones analizadas en clases, *Quicksort* sólo usa un ciclo para recorrer el arreglo, mientras que *Bubblesort* usa dos.

b.
Quicksort no implementa ninguna mejora sustancial sobre *Bubblesort*.

c.
Quicksort primero determina qué tan desordenado está el arreglo, mientras que *Bubblesort* procede directamente a ordenarlo

d.
Quicksort acelera el cambio de posición tanto de los elementos menores como de los mayores, mientras que *Bubblesort* solo acelera a los mayores (o a los menores, dependiendo de la forma de implementación).

◄ Avisos

Ir a...