

<b>Comenzado el</b>	domingo, 27 de agosto de 2023, 15:29
<b>Estado</b>	Finalizado
<b>Finalizado en</b>	domingo, 27 de agosto de 2023, 15:33
<b>Tiempo empleado</b>	4 minutos 15 segundos
<b>Puntos</b>	13/13
<b>Calificación</b>	10 de 10 (96%)


Pregunta **1**

Correcta

Se puntúa 1 sobre 1

¿Cuál es la **idea principal** en la que se basa el algoritmo de ordenamiento de **Selección Directa** que se presentó en clase? (Suponga que se desea ordenar el arreglo de menor a mayor).

Seleccione una:

- ☒ a. Realizar varias pasadas sobre el arreglo, en cada pasada **seleccionar el menor valor** de los que aún no han sido ordenados, y  colocarlo en su casilla definitiva.
- ☐ b. Realizar varias pasadas sobre el arreglo, en cada pasada **comparar a cada elemento con el que le sigue inmediatamente**, intercambiarlos si están desordenados y seguir haciendo pasadas en esta forma hasta que el arreglo quede ordenado.
- ☐ c. Realizar varias pasadas sobre el arreglo. En cada pasada seleccionar un elemento cualquiera **x**. Pasar a la derecha del arreglo todos los valores mayores a x. Pasar a la izquierda del arreglo todos los valores menores a x. Finalmente, aplicar la misma idea a cada una de las dos "mitades" así obtenidas y seguir así hasta que ya no pueda seguir obteniendo nuevas "mitades".
- ☐ d. Realizar **una sola pasada sobre el arreglo**, en esa pasada comparar a cada elemento con el que le sigue e intercambiarlos si están invertidos.

## Pregunta 2

Correcta

Se puntúa 1 sobre 1

La siguiente función implementa el algoritmo de ordenamiento por *Selección Directa*, tal como se analizó en clases para ordenar de menor a mayor un arreglo  $v$  con  $n$  componentes:

```
def selection_sort(v):  
    # ordenamiento por seleccion directa  
    n = len(v)  
    for i in range(n-1):  
        for j in range(i+1, n):  
            if v[i] > v[j]:  
                v[i], v[j] = v[j], v[i]
```

¿Qué efecto se produciría en la función anterior si la instrucción condicional `if v[i] > v[j]:` fuese reemplaza por `if v[i] < v[j]:`?

Seleccione una:

- ☐ a. Provocaría que el arreglo permanezca siempre sin cambio alguno (no será nunca ordenado de forma alguna, ni se modificará nunca su contenido original).
- ☐ b. No causaría ningún efecto particular: el arreglo seguiría siendo ordenado de menor a mayor.
- ☒ c. El arreglo sería ordenado, pero ahora de mayor a menor. ✓
- ☐ d. Provocará que el programa se interrumpa con un mensaje de error en la primera comparación.

## Pregunta 3

Correcta

Se puntúa 1 sobre 1

¿Cuántas comparaciones hace el algoritmo de Búsqueda Secuencial para encontrar un valor  $x$  en un arreglo de  $n$  componentes en el *peor caso posible*? (El peor caso es el que obliga a un algoritmo a hacer la máxima cantidad de trabajo. Obviamente, en el caso de la Búsqueda Secuencial ese peor caso se presenta si el valor buscado está exactamente en la última casilla, o bien, si el valr buscado no está en el arreglo).

Seleccione una:

- ☒ a.  $n$  comparaciones. ✓
- ☐ b.  $n^2$  comparaciones.
- ☐ c. Una sola comparación, siempre.
- ☐ d.  $\log_2(n)$  comparaciones.

## Pregunta 4

Correcta

Se puntúa 2 sobre 2

Oportunamente se presentó en clases el algoritmo de **búsqueda secuencial**, el cual toma un arreglo, busca un valor  $x$  en el mismo casilla por casilla, y retorna el índice de la casilla que lo contiene (si  $x$  está en el arreglo) o retorna -1 si  $x$  no está en el arreglo. Suponga que propone la siguiente variante para el algoritmo de búsqueda secuencial:

```
def linear_search(v, x):  
    r = -1  
    for i in range(len(v)):  
        if x == v[i]:  
            r = i  
  
    return r
```

¿Hay algún inconveniente o problema con esta variante?

Seleccione una:

- ☐ a. La variante propuesta funciona solamente si el valor  $x$  está en el arreglo (falla si  $x$  no está).
- ☐ b. La variante propuesta funciona correctamente.
- ☒ c. Hay un inconveniente: la variante propuesta encuentra el valor buscado (si existía) pero no corta el ciclo de búsqueda al encontrarlo con lo que siempre se hacen  $n$  comparaciones. ✓
- ☐ d. La variante propuesta no funciona correctamente: sólo llega a analizar el contenido de la primera casilla.

## Pregunta 5

Correcta

Se puntúa 2 sobre 2

Oportunamente se presentó en clases el algoritmo de **búsqueda secuencial**, el cual toma un arreglo, busca un valor  $x$  en el mismo casilla por casilla, y retorna el índice de la casilla que lo contiene (si  $x$  está en el arreglo) o retorna -1 si  $x$  no está en el arreglo. Suponga que propone la siguiente variante para el algoritmo de búsqueda secuencial:

```
def linear_search(v, x):  
    n = len(v)  
    for i in range(n):  
        if x == v[i]:  
            return i  
    else:  
        return -1
```

¿Funciona correctamente esta variante? Si no funciona, ¿cuál es el problema?

Seleccione una:

- ☒ a. La variante propuesta no funciona correctamente: sólo llega a analizar el contenido de la primera casilla. ✓
- ☐ b. La variante propuesta funciona solamente si el valor  $x$  está en el arreglo (falla si  $x$  no está).
- ☐ c. La variante propuesta funciona correctamente.
- ☐ d. La variante propuesta provoca un error de intérprete: un **if** no puede tener una instrucción **return** en cada una de sus ramas.

Pregunta **6**

Parcialmente correcta

Se puntúa 1 sobre 1

¿En cuáles de los siguientes casos es aplicable el algoritmo de **búsqueda binaria** en un arreglo? (Seleccione todas las respuestas que considere correctas)

Seleccione una o más de una:

- ☐ a. El arreglo en el cual se debe realizar la búsqueda está desordenado y no se nos permite ordenarlo.
- ☒ b. El arreglo en el cual se debe realizar la búsqueda está desordenado, se nos permite ordenarlo, luego podrá alterarse el **✗** contenido (quedando eventualmente desordenado) y debemos realizar varias búsquedas.
- ☐ c. El arreglo en el cual se debe realizar la búsqueda está desordenado, se nos permite ordenarlo, luego permanecerá sin cambios y debemos realizar muchas búsquedas.
- ☒ d. El arreglo en el cual se debe realizar la búsqueda está ordenado y permanecerá sin cambios. **✓**

Pregunta **7**

Correcta

Se puntúa 2 sobre 2

Oportunamente se presentó en clases el algoritmo de **búsqueda binaria**, el cual toma un arreglo ordenado, busca un valor  $x$  en el mismo, y retorna el índice de la casilla que lo contiene (si  $x$  está en el arreglo) o retorna -1 si  $x$  no está en el arreglo. Suponga que propone la siguiente variante para el algoritmo de búsqueda binaria:

```
def binary_search(v, x):
    # busqueda binaria... asume arreglo ordenado...
    izq, der = 0, len(v) - 1
    while izq <= der:
        c = (izq + der) // 2
        if x == v[c]:
            return c
        else:
            return -1

        if x < v[c]:
            der = c - 1
        else:
            izq = c + 1

    return -1
```

¿Funciona correctamente esta variante? Si no funciona, ¿cuál es el problema?

Seleccione una:

- ☐ a. La variante propuesta provoca un error de intérprete y no llega a arrancar: el segundo **if** incluido dentro del ciclo nunca puede ejecutarse, ya que las dos ramas del **if** anterior terminan con una instrucción **return**.
- ☐ b. La variante propuesta funciona correctamente.
- ☒ c. La variante propuesta no funciona correctamente: sólo llega a analizar el contenido de la casilla del centro del arreglo. **✓**
- ☐ d. La variante propuesta funciona solamente si el valor  $x$  está en el arreglo (falla si  $x$  no está).

## Pregunta 8

Correcta

Se puntúa 1 sobre 1

Oportunamente se presentó en clases el algoritmo de [búsqueda binaria](#), el cual toma un arreglo ordenado, busca un valor  $x$  en el mismo, y retorna el índice de la casilla que lo contiene (si  $x$  está en el arreglo) o retorna -1 si  $x$  no está en el arreglo. Mostramos el algoritmo para dar mejor contexto a la pregunta:

```
def binary_search(v, x):  
    # busqueda binaria... asume arreglo ordenado...  
    izq, der = 0, len(v) - 1  
    while izq <= der:  
        c = (izq + der) // 2  
        if x == v[c]:  
            return c  
        if x < v[c]:  
            der = c - 1  
        else:  
            izq = c + 1  
  
    return -1
```

¿Qué pasaría con la función anterior si el arreglo  $v$  contuviese *cadena de caracteres* en cada casillero (en lugar de números), y el parámetro  $x$  contuviese también una cadena de caracteres?

Seleccione una:

- ☐ a. La función mostrada funciona solamente si el valor  $x$  está en el arreglo (falla si  $x$  no está, interrumpiendo el programa con un mensaje de error).
- ☒ b. La función mostrada funcionaría correctamente de todos modos. ✓
- ☐ c. La función mostrada provocaría un error en tiempo de ejecución y se interrumpiría el programa al intentar determinar si  $x$  menor o mayor que  $v[i]$ .
- ☐ d. La función mostrada no provocaría que el programa se interrumpa, pero funcionaría en forma incorrecta y retornaría siempre -1 (nunca encontraría la cadena  $x$  buscada).

## Pregunta 9

Correcta

Se puntúa 1 sobre 1

¿Qué pasaría con el algoritmo de [Búsqueda Binaria](#) si el valor buscado  $x$  estuviese repetido más de una vez en el arreglo?

Seleccione una:

- ☒ a. El algoritmo funcionaría correctamente de todos modos: retornaría el índice de la primera casilla encontrada que contenga a  $x$ . ✓
- ☐ b. El algoritmo produciría un error en tiempo de ejecución y se interrumpiría el programa.
- ☐ c. El algoritmo no provocaría que el programa se interrumpa, pero funcionaría en forma incorrecta y retornaría siempre -1 (nunca encontraría *el valor*  $x$  buscado).
- ☐ d. El algoritmo funciona solamente si el valor  $x$  está en el arreglo (falla si  $x$  no está, interrumpiendo el programa con un mensaje de error).

Pregunta **10**

Correcta

Se puntúa 1 sobre 1

¿En cuáles de los siguientes casos es aplicable el algoritmo de *Fusión de Arreglos* que se describió en las fichas de clases para producir un tercer arreglo ordenado? (Seleccione todas las respuestas que considere correctas)

Seleccione una o más de una:

- ☐ a. Los arreglos originales deben estar desordenados.
- ☒ b. Los arreglos originales deben estar ordenados de menor a mayor si se quiere producir un tercer arreglo ordenado de menor a mayor. ✓
- ☐ c. El proceso es aplicable sin importar si los dos arreglos originales están ordenados o no, ya que el algoritmo de fusión analizado primero ordena esos dos arreglos, y luego procede a la fusión.
- ☐ d. Uno de los arreglos originales debe estar ordenado y el otro arreglo puede estar desordenado.

◀ [Materiales Adicionales para la Ficha 16](#)

Ir a...



[Parcial 2 - Archivo de Texto de Entrada](#) ▶