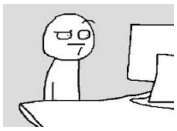


Universidad del País Vasco Euskal Herriko Unibertsitatea

Prog. Básica - Laboratorio 8
Estructuras de datos

Nombre:_____**Fecha**:_____

AVISOS:
1. Cuando uno de los ejercicios que se proponen carece de programa de prueba o de plantilla, significa que lo tienes que hacer desde cero. Además, el hecho de que se proporcionen algunos casos de prueba no significa que no pueda faltar alguno. ¡Tienes que añadir los casos de prueba que falten!
2. Tienes que subir a eGela los ficheros fuente (.adb) comprimidos en un fichero (.zip) que deberá ajustarse a las reglas de nombrado exigidas hasta ahora (por ejemplo, EJauregi_lab7.zip).
3. Se presupone que los ejercicios son correctos, es decir, que no tienen errores de compilación y que funcionan correctamente. Esto significa que las soluciones de los ejercicios no puntúan nada por ser correctas, pero penalizan si tienen errores o no se ajustan a lo que se pide. Una vez que la solución es correcta, lo que se evalúa (lo que puntúa) son:

- **Los casos de prueba: ¿Se han contemplado todos los casos de prueba, desde los generales a los más críticos? Se dan algunos, pero faltan muchos otros.**
- **La eficiencia: ¿Se utilizan los "chivatos" cuando hace falta? ¿No hay asignaciones innecesarias? ¿No hay condicionales que no deberían ejecutarse?, ¿Se definen solamente los parámetros o variables necesarios?, etcétera.**
- **La claridad: ¿El código está tabulado? ¿Los nombres de las variables ayudan a entender el código? ¿Hay un único *return* al final de la función?, etcétera.**
- **Quien quiera hacer los ejercicios en PYTHON puede utilizar la siguiente dirección para implementar los ejercicios:**

https://py3.codeskulptor.org

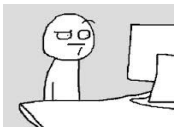
1. Búsqueda en un vector

Crear un subprograma que, dado un valor entero y un vector de N enteros (no necesariamente completo, o sea, con, como mucho, N enteros) cuyos valores pueden no estar ordenados, diga si el entero está o no en el vector.

Plantillas ofrecidas:

esta_en_vector.adb	Para hacer el subprograma en Ada
prueba_esta_en_vector.adb	Para completar y probar el subprograma en Ada

Fichero adicional: datos.ads (No hay que modificarlo).



2. Borrado del tercer elemento en una lista ordenada

Crear un subprograma que, dado una lista de, como máximo N enteros (no necesariamente completo, o sea, con, como mucho, N enteros) cuyos valores están ordenados ascendentemente, lo modifique eliminando su tercer elemento (si existe, si no existe no hará nada). Los elementos de la lista están ordenados antes y deben seguir ordenados después de realizar el borrado del tercer elemento.

Plantillas ofrecidas:

eliminar_tercer_elemento_ordenada.adb	Para hacer el subprograma en Ada
prueba_eliminar_tercer_elemento_ordenada.adb	Para completar y probar el subprograma Ada

Fichero adicional: datos.ads y escribir_lista.adb (No hay que modificarlo).

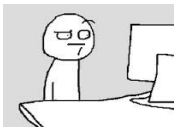
3. Inserción de un elemento en una posición

Crear un subprograma que, dados dos enteros *num* y *pos*, y una lista de, como máximo N-1 enteros (lo que significa que aún hay hueco para uno más), inserte en la lista el número *num* en la posición *pos*, desplazando los elementos que corresponda una posición a la derecha.
Para resolver este problema conviene utilizar el programa *rotar_derecha*, añadiéndole como parámetros de entrada la posición *Pos* y el número a insertar *Num*.

Plantillas ofrecidas:

rotar_derecha.adb	Para hacer el subprograma en Ada
insertar_elemento_en_pos.adb	Para hacer el subprograma en Ada
prueba_insertar_elemento_en_pos.adb	Para completar y probar el subprograma Ada

Fichero adicional: datos.ads y escribir_lista.adb (No hay que modificarlo).



4. Eliminar repetidos

Hacer un subprograma que, dada una lista de enteros, obtenga otra lista equivalente que no contenga elementos repetidos.

Plantillas ofrecidas:

esta.adb	Para hacer el subprograma en Ada
eliminar_repetidos.adb	Para hacer el subprograma en Ada
prueba_eliminar_repetidos.adb	Para completar y probar el subprograma Ada

Fichero adicional: datos.ads y escribir_lista.adb (No hay que modificarlo).

5. Ordenación por inserción

Dado una lista de N enteros (N>0), hacer un subprograma que lo ordene mediante el método de inserción.

- Video ilustrativo: http://www.youtube.com/watch?v=gTxFxgvZmQs&feature=related

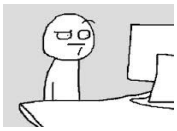
Los elementos se toman de uno en uno siguiendo el orden en el que están inicialmente y se insertan en la lista de forma ordenada de manera similar a como se hacía en el ejercicio de *inserción en un vector de un elemento en una posición*. Al comienzo de la *i*-ésima iteración, los primeros *i*-1 elementos de la lista están ya ordenados; se toma el elemento actual (el de la posición *i*-1) y se compara con los elementos ya ordenados para determinar cuál es el lugar que le corresponde. Como consecuencia, en la mayoría de las iteraciones habrá que hacer un hueco en la lista, desplazando algunos elementos hacia la derecha, para poder insertar el elemento actual en la posición correspondiente.

Plantillas ofrecidas:

buscar_posicion_de_insercion.adb	Para hacer el subprograma en Ada
desplazar_una_posicion_a_la_derecha.adb	Para hacer el subprograma en Ada
ordenar_por_insercion.adb	Para hacer el subprograma en Ada
prueba_ordenar_por_insercion.adb	Para completar y probar el subprograma Ada

Fichero adicional: datos.ads y escribir_lista.adb (No hay que modificarlo).

- *buscar_posicion_de_insercion*. Recibe como parámetros la *lista L*, la *posición actual* (esto es, la que delimita los elementos que hay ya ordenados en las primeras posiciones de la lista) y el *elemento actual* (es decir, el número que se quiere insertar); y devuelve la posición que le correspondería en la lista al elemento actual (entre 0 y *posición actual*-1).
- *desplazar_una_posicion_a_la_derecha.* Recibe como parámetros la *lista L* y dos enteros que representan sendas posiciones: la *posición actual* (que igual que antes se corresponde con la cantidad de elementos que hay ya ordenados en las primeras posiciones de la lista) y la *posición de comienzo* (que será la posición en la que después se insertará el elemento actual). Devuelve la lista con todos los elementos que hay entre la *posición de comienzo* y la *posición actual*-1 desplazados una posición a la derecha. Esto implica que el valor de la *posición actual* (que es precisamente el elemento actual) se sobrescribe, por lo que antes de llamar a esta función conviene haberlo guardado en una variable auxiliar.



Ejemplo:

Lista inicial que se quiere ordenar (10 elementos):

0	1	2	3	4	5	6	7	8	9
9	5	3	4	10	8	13	24	15	11

Inserción 1: se inserta el primer elemento (9) en la posición 0; la lista no cambia (posActual=1):

0	1	2	3	4	5	6	7	8	9
9	5	3	4	10	8	13	24	15	11

Inserción 2: se inserta el segundo elemento (5) en la posición 0, desplazando el 9 (posActual=2)

0	**1**	2	3	4	5	6	7	8	9
5	**9**	3	4	10	8	13	24	15	11

Inserción 3: se inserta el tercer elemento (3) en la posición 0, desplazando el 5 y el 9 (posActual=3):

0	**1**	**2**	3	4	5	6	7	8	9
3	**5**	**9**	4	10	8	13	24	15	11

Inserción 4: se inserta el cuarto elemento (4) en la posición 1, desplazando el 5 y el 9 (posActual=4):

0	**1**	**2**	**3**	4	5	6	7	8	9
3	**4**	**5**	**9**	10	8	13	24	15	11

Inserción 5: se inserta el quinto elemento (10) en la posición 4, sin desplazar nada (posActual= 5):

0	**1**	**2**	**3**	**4**	5	6	7	8	9
3	**4**	**5**	**9**	**10**	8	13	24	15	11

Etcétera...

NOTA: Es imprescindible que la solución funcione para cualquier número de elementos (N>0).

6. Ordenación por selección

Dado una lista de N enteros (N>0), hacer un subprograma que lo ordene mediante el método de selección.

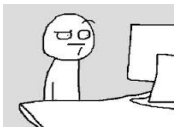
- Video ilustrativo: http://www.youtube.com/watch?v=boOwArDShLU

Se trata de buscar el menor elemento de la lista e intercambiarlo con el elemento que ocupa la primera posición. Después se busca el menor del resto de la lista y se intercambia con el que esté ocupando la segunda posición. En general, al comienzo de la *i*-ésima iteración, los primeros *i*-1 elementos de la lista están ya ordenados en su posición definitiva; se busca el elemento menor entre la posición actual (es decir, la posición *i*-1) y el final de la lista, y se intercambia con el que actualmente ocupa la posición *i*-1.

Plantillas ofrecidas:

buscar_posicion_del_minimo.adb	Para hacer el subprograma en Ada
intercambiar.adb	Para hacer el subprograma en Ada
ordenar_por_seleccion.adb	Para hacer el subprograma en Ada
prueba_ordenar_por_seleccion.adb	Para completar y probar el subprograma Ada

Fichero adicional: datos.ads y escribir_lista.adb (No hay que modificarlo).



- *buscar_posicion_del_minimo*. Recibe como parámetros la *lista L* y la *posición de comienzo* (que corresponderá con la posición actual, esto es, la que delimita los elementos que hay ya ordenados en las primeras posiciones de la lista). Devuelve la posición en la que se encuentra el valor mínimo comprendido entre la *posición de comienzo* y el final de la lista L.
- *intercambiar*. Recibe como parámetros la *lista L* y dos enteros que representan sendas posiciones: *posA* y *posB*. Devuelve la lista L con los valores de las posiciones *posA* y *posB* intercambiados.

Ejemplo:

Lista inicial que se quiere ordenar (10 elementos):

0	1	**2**	3	4	5	6	7	8	9
9	5	**3**	4	10	8	13	24	15	11

El menor entre la posición 0 y la última es 3 (posición 2); se intercambian las posiciones 0 y 2:

0	1	2	**3**	4	5	6	7	8	9
3	5	9	**4**	10	8	13	24	15	11

El menor entre la posición 1 y la última es 4 (posición 3); se intercambian las posiciones 1 y 3:

0	**1**	2	**3**	4	5	6	7	8	9
3	**4**	9	**5**	10	8	13	24	15	11

El menor entre la posición 2 y la última es 5 (posición 3); se intercambian las posiciones 2 y 3:

0	**1**	**2**	3	4	**5**	6	7	8	9
3	**4**	**5**	9	10	**8**	13	24	15	11

El menor entre la posición 3 y la última es 8 (posición 5); se intercambian las posiciones 3 y 5:

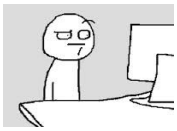
0	**1**	**2**	**3**	4	**5**	6	7	8	9
3	**4**	**5**	**8**	10	**9**	13	24	15	11

El menor entre la posición 4 y la última es 9 (posición 5); se intercambian las posiciones 4 y 5:

0	**1**	**2**	**3**	**4**	5	6	7	8	9
3	**4**	**5**	**8**	**9**	10	13	24	15	11

Etcétera…

NOTA: Es imprescindible que la solución funcione para cualquier número de elementos (N>0).



7. Ordenación por el método de la burbuja

Dado una lista de N enteros (N>0), hacer un subprograma que lo ordene mediante el método de la burbuja de manera eficiente.

- Video ilustrativo: http://www.youtube.com/watch?v=1JvYAXT_064&feature=related

El algoritmo puede ser el siguiente:

```
num_recorridos:=1;
repetir salir si num_recorridos > num_elementos-1;
     i:=1;
     repetir salir si i> num_elementos-1;
          si L(i) > L(i+1) entonces ---intercambiarlos
               aux:=L(i);
               L(i):=L(i+1);
               L(i+1):=aux;
          fin_si;
          i:=i+1;
     fin_repetir;
     num_recorridos:= num_recorridos+1;
fin_repetir;
```

Ejemplo:

Lista inicial que se quiere ordenar (4 elementos):

7	5	1	2

Despues de la primera iteración, el último elemento ya está ordenado.

5	1	2	7

Después de dos iteraciones hay dos elementos ordenados.

1	2	5	7

Y después del tercer y último recorrido la lista quedaría igual.

1	2	5	7

En la última iteración no se ha hecho nada, porque la lista ya estaba ordenada. Sin embargo, en el peor de los casos (a saber, en el caso en el que la lista inicial hubiese estado TOTALMENTE desordenada) hubiera hecho falta esta vuelta. Como de antemano no se puede saber si la lista está o no totalmente desordenada, el algoritmo realiza el proceso completo, esto es, con todas las vueltas necesarias para el caso peor.

Se pide que vuestra solución mejore la eficiencia del algoritmo anterior, **añadiendo un booleano** tal que si no se ha hecho ningún intercambio en la iteración actual se salga del bucle, dado que esto significará que la lista ya está ordenado.

NOTA: Es imprescindible que la solución funcione para cualquier número de elementos (N>0).