## 4.5. Algoritmos de Búsqueda en Memoria Principal

En esta sección vamos a estudiar los algoritmos de búsqueda de un elemento en un arreglo (¿dónde está un elemento dado?) y los algoritmos de búsqueda en general de distintas características en un conjunto de valores (por ejemplo, ¿cuántas veces aparece un valor en un arreglo?), los cuales necesitamos para escribir los métodos que implementan los requerimientos funcionales del caso de estudio.

## 4.5.1. Búsqueda de un Elemento

El proceso de búsqueda de un elemento en un arreglo depende básicamente de si éste se encuentra ordenado. En caso de que no haya ningún orden en los valores, la única opción que tenemos es hacer una búsqueda secuencial utilizando para esto el patrón de recorrido parcial estudiado en cursos anteriores.

Tarea 13



**Objetivo:** Escribir el método que busca un elemento en un arreglo.

Para la clase Muestra escriba dos versiones del método que busca un elemento: en el primero, haga una búsqueda secuencial suponiendo que la muestra no está ordenada. En el segundo, haga también una búsqueda secuencial pero suponga que la muestra está ordenada ascendentemente. ¿Qué cambia de un algoritmo a otro?

Si el arreglo en el que queremos buscar el elemento se encuentra ordenado, podemos utilizar una técnica muy eficiente de localización que se denomina la **búsqueda binaria**. La idea de esta técnica, que se ilustra en el ejemplo 12, es localizar el elemento que se encuentra en la mitad del arreglo. Si ese elemento es mayor que el valor que estamos buscando, podemos descartar en el proceso de búsqueda la mitad final del arreglo (¿para qué buscar allí si todos los

25

mobic

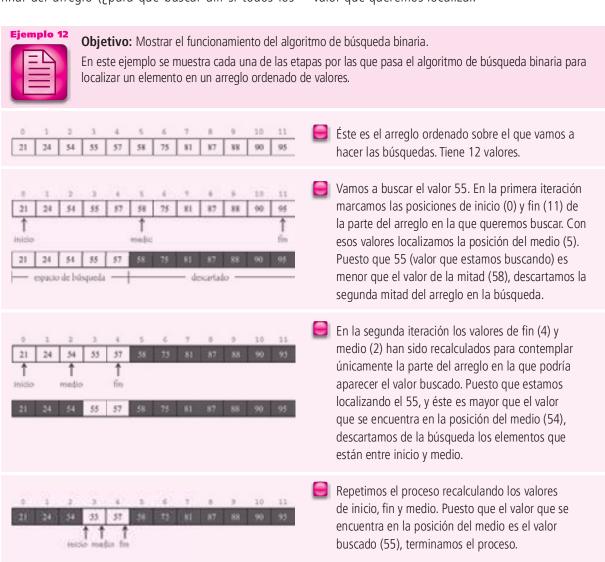
Inicia

valores de esa parte del arreglo son mayores que aquél que estamos buscando?). Si el elemento de la mitad es menor que el valor buscado, descartamos la mitad inicial del arreglo. Piense que sólo con lo anterior ya bajamos el tiempo de ejecución del método a la mitad. Y si volvemos a repetir el mismo proceso anterior con la parte del arreglo que no hemos descartado, iremos avanzando rápidamente hacia el valor que queremos localizar.

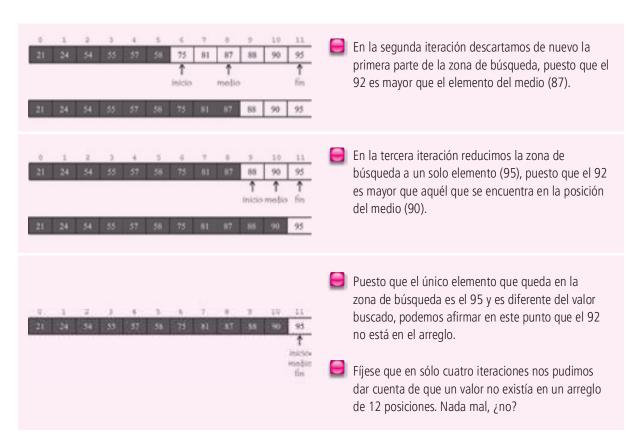
Vamos a buscar ahora un valor inexistente en el

arreglo para ver el comportamiento del algoritmo. Busquemos entonces el 92. En la primera iteración

descartamos los valores de la primera mitad, puesto que el 92 es mayor que el valor del medio (58).



95



A continuación se muestra el código del método de la clase MuestraOrdenada encargado de buscar un elemento usando la técnica de búsqueda binaria:

```
public boolean buscarBinario( int valor )
{
   boolean encontre = false;
   int inicio = 0;
   int fin = valores.length - 1;
   while( inicio <= fin && !encontre )
   {
      int medio = ( inicio + fin ) / 2;
      if( valores[ medio ] == valor )
      {
        encontre = true;
      }
      else if( valores[ medio ] > valor )
      {
        fin = medio - 1;
      }
      else
      {
        inicio = medio + 1;
      }
   }
   return encontre;
}
```

- Vamos a utilizar tres variables enteras (inicio, medio y fin) para indicar en cada iteración la posición donde comienza la zona de búsqueda, la posición media y la posición en la que termina.
- Inicialmente "inicio" comienza en 0 y "fin" en la última posición del arreglo.
- La posición media se calcula sumando las posiciones de inicio y fin, y dividiendo el resultado entre 2.
- Luego, hay tres casos: (1) si el valor es igual al de la mitad, ya lo encontramos; (2) si el valor es menor que el de la mitad, reposicionamos la marca final de la zona de búsqueda, y (3) si el valor es mayor, movemos el inicio de la zona de búsqueda.
- Este proceso se repite mientras no hayamos encontrado el elemento y mientras exista algún elemento en el interior de la zona de búsqueda.



**Objetivo:** Medir el tiempo de ejecución de los algoritmos de búsqueda e intentar identificar diferencias entre ellos.

Localice en el CD el ejemplo n7\_muestra, cópielo al disco duro y ejecútelo. Siga luego las instrucciones que aparecen a continuación.

Algoritmo	5000	10000	20000	40000	60000	80000	100000	200000		Llene la tabla de tiempos de ejecución de los dos algoritmos de búsqueda, para muestras de distintos tamaños. Utilice una muestra entre 1 y 200.000.
Secuencial										
Binaria										

¿Qué se puede concluir de la tabla anterior?

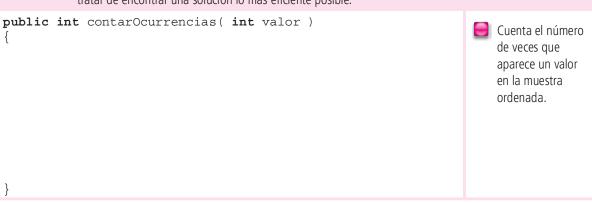
## 4.5.2. Búsquedas en Estructuras Ordenadas

En esta sección planteamos, en términos de tareas, los métodos de búsqueda necesarios para completar el programa del caso de estudio.



**Objetivo:** Escribir los métodos que se necesitan en el caso de estudio, para calcular algún valor sobre la muestra ordenada.

Escriba los métodos que se piden a continuación en la clase MuestraOrdenada. Asegúrese de utilizar en su algoritmo el hecho de que los valores de la muestra se encuentran ordenados, para así tratar de encontrar una solución lo más eficiente posible.



```
public int contarValoresDistintos( )
                                                                               Cuenta el número
                                                                                  de valores distintos
                                                                                  que hay en la
                                                                                  muestra ordenada.
public int contarElementosEnRango( int inf, int sup )
                                                                               Cuenta el número
                                                                                  de elementos que
                                                                                  hay en un rango de
                                                                                  valores (incluidos
                                                                                  los extremos).
public int darValorMasFrecuente( )
                                                                               Retorna el valor
                                                                                  más frecuente
                                                                                  en la muestra
                                                                                  ordenada. Si hay
                                                                                  varios números
                                                                                  con la misma
                                                                                  frecuencia, retorna
                                                                                  el menor de ellos.
```