Arreglos en C++

Operaciones

Tomás Peiretti

Operaciones con arreglos

¿Qué operaciones debemos saber realizar con los arreglos para aprobar AEDD?

- Recorrer (de izq a derecha, en un rango, de der a izq, etc)
- Invertir un arreglo
- Eliminar/agregar un elemento
- Buscar un elemento
- Ordenar (BubbleSort, MergeSort, SelectionSort, InsertionSort)

Recorrer un arreglo

```
1  void recorrerCompleto(const int arr[], int tam) {
2      for(int i=0; i<tam; i++) {
3          cout << arr[i] << endl;
4      }
5  }
6
7  void recorrerEnRango(const int arr[], int a, int b) {
8      for (int i=a; i<=b; i++) {
9          cout << arr[i] << endl;
10     }
11 }
12
13 void recorrerCompletoAlReves(const int arr[], int tam) {
14     for(int i = tam-1 ; i>=0 ; i--) {
15         cout << arr[i] << endl;
16     }
17 }
18
19 void recorrerHaciaElCentro(const int arr[], int tam) {
20     int izq = 0, der = tam - 1;
21     while (izq < der) {
22         cout << "arr[" << izq << "] = " << arr[izq] << endl;
23         cout << "arr[" << der << "] = " << arr[der] << endl;
24         izq++;
25         der--;
26     }
27     // elemento del medio
28     if (tam%2 != 0)
29         cout << "arr[" << tam/2 << "] = " << arr[tam/2] << endl;
30 }
```

Invertir un arreglo

```
1  void invertirConCopia(int arr[], int tl) {
2
3      int copia[100000];
4      for (int i=0; i<tl; i++) {
5          copia[tl-1 - i] = arr[i];
6      }
7
8      for (int i=0; i<tl; i++) {
9          arr[i] = copia[i];
10     }
11 }
12
13 void invertir(int arr[], int tl) {
14
15     int izq = 0, der = tl-1;
16
17     while(izq <= der) {
18         swap(arr[izq], arr[der]);
19         izq++;
20         der--;
21     }
22 }
```

Eliminar un elemento del arreglo

```
1  // elimina el elemento que
2  // se encuentra en la posicion indicada
3  void eliminarElemento(char arr[], int &tl, int posicion) {
4
5      if (tl == 0)
6          cout << "ERROR! no hay elementos para eliminar";
7      else if (posicion >= tl)
8          cout << "ERROR! posicion invalida";
9      else {
10         // borrar un elemento implica desplazar todos los
11         // elementos que estaban a su derecha
12         // una posicion a la izquierda
13         for (int i=posicion; i<tl-1; i++) {
14             arr[i] = arr[i+1];
15         }
16         // y no hay que olvidarse de actualizar
17         // el tamanioo logico
18         tl--;
19     }
20 }
```

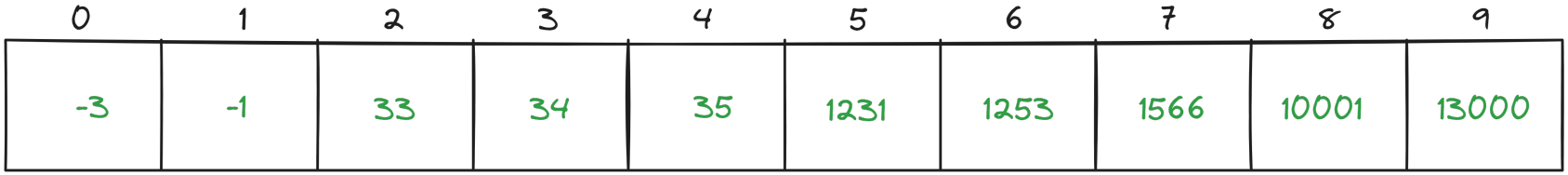
Agregar un elemento al arreglo

```
1  #define TF 1500
2
3  // agrega el elemento en la posicion indicada
4  void agregarElemento(int arr[], int &tl, int posicion, int elemento) {
5
6      if (tl == TF)
7          cout <<"ERROR! ya no queda espacio";
8      else if (posicion > tl)
9          cout << "ERROR! posicion invalida";
10     else {
11         // agregar un elemento implica desplazar todos los
12         // elementos que se encuentran de la posicion en adelante
13         // una posicion a la derecha
14         // (primero hay que darle lugar al nuevo elemento)
15         for (int i=tl; i>=posicion; i--) {
16             arr[i+1] = arr[i];
17         }
18         //luego hay que agregar el elemento
19         arr[posicion] = elemento;
20         // y no hay que olvidarse de actualizar
21         // el tamanioo logico
22         tl++;
23     }
24 }
```

Buscar un elemento

Una de las operaciones que más utilizaremos es la búsqueda de cierto elemento dentro de un arreglo.

Supongamos que queremos buscar dentro del siguiente arreglo de enteros el número 1253

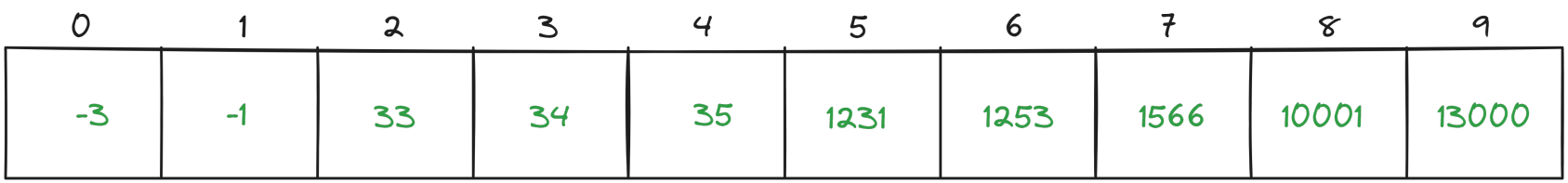


Buscar un elemento: búsqueda lineal

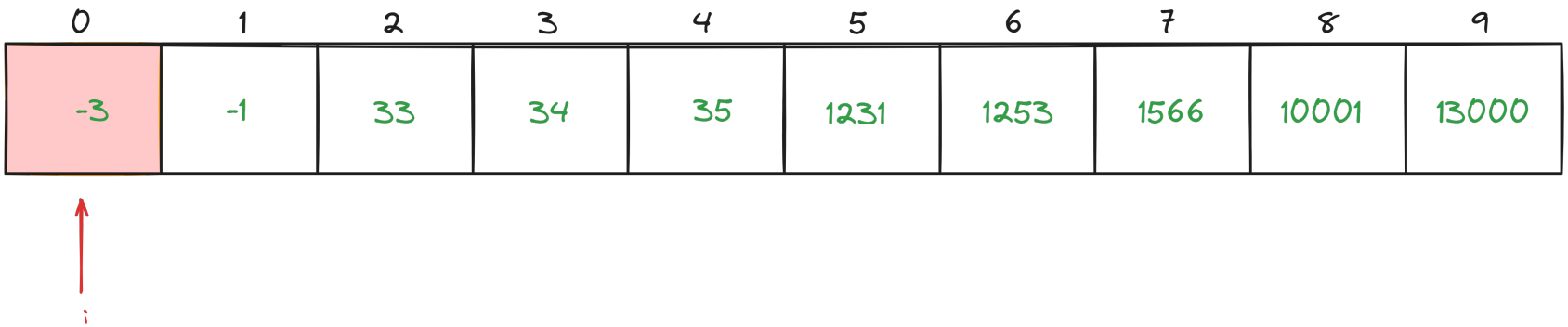
En la mayoría de los casos, cuando necesitemos buscar un elemento dentro de un arreglo implementaremos una búsqueda lineal

```
1  // busqueda lineal
2  void buscarChar(const char arr[], int tam, char c) {
3      int i = 0;
4      bool encontrado = false;
5      while(i<tam && !encontrado) {
6          if (arr[i] == c)
7              encontrado=true;
8          i++;
9      }
10
11     if (encontrado)
12         cout << "Encontre " << c << " en " << i-1 << endl;
13     else
14         cout << "No encontre el char " << c << endl;
15 }
```

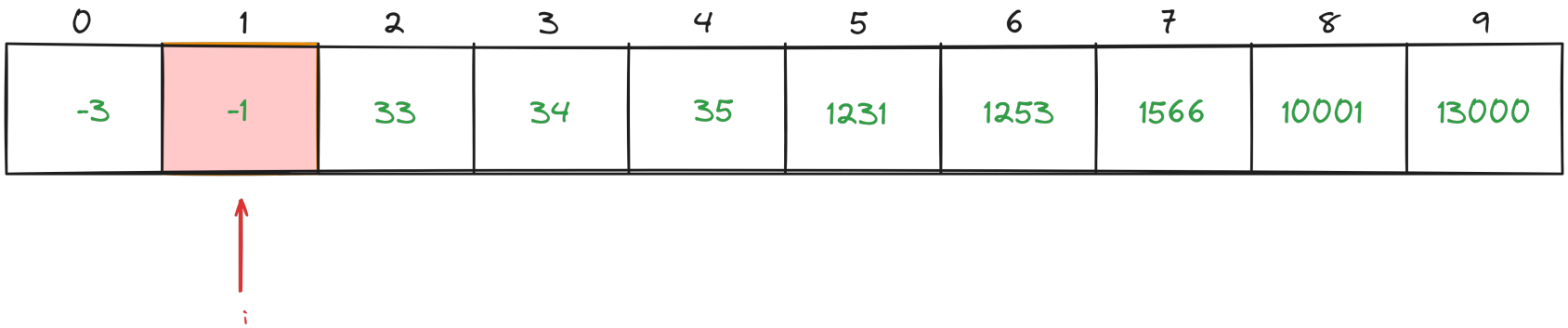
Buscar un elemento: búsqueda lineal



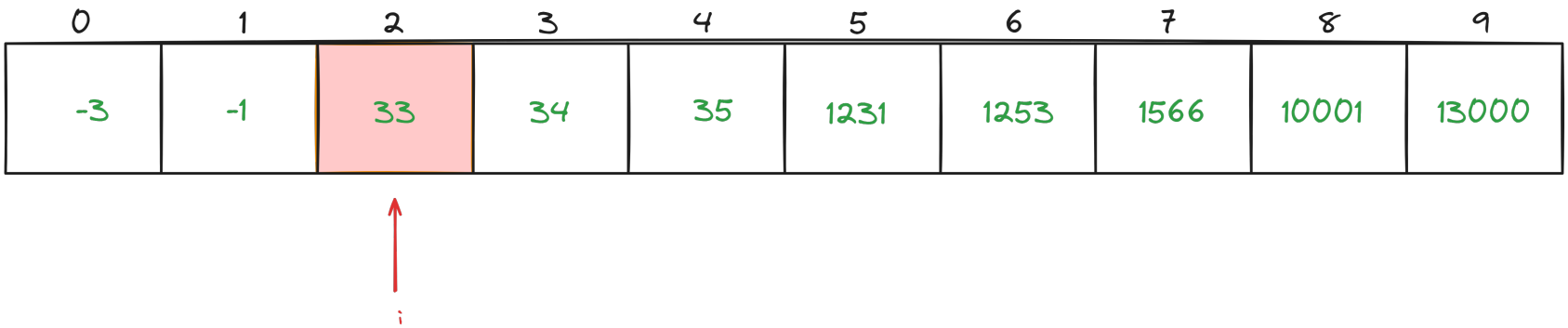
Buscar un elemento: búsqueda lineal



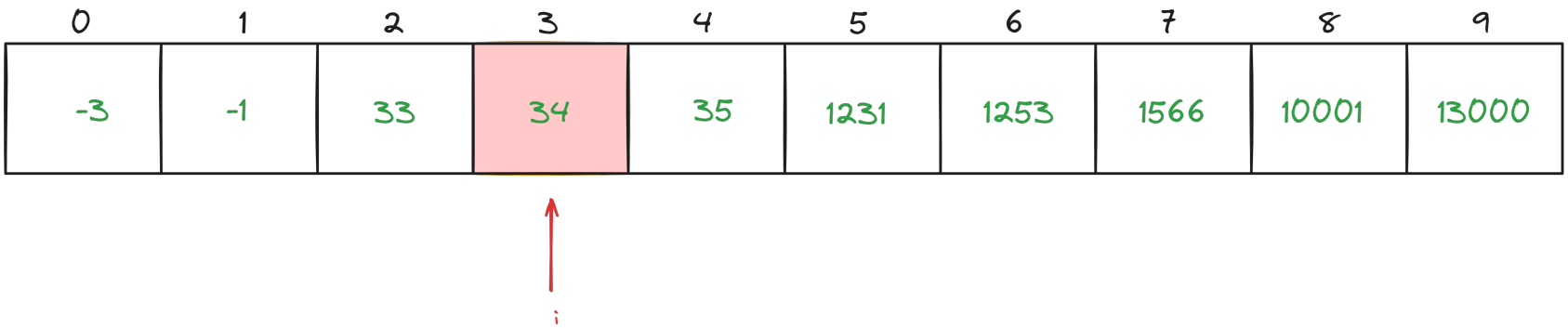
Buscar un elemento: búsqueda lineal



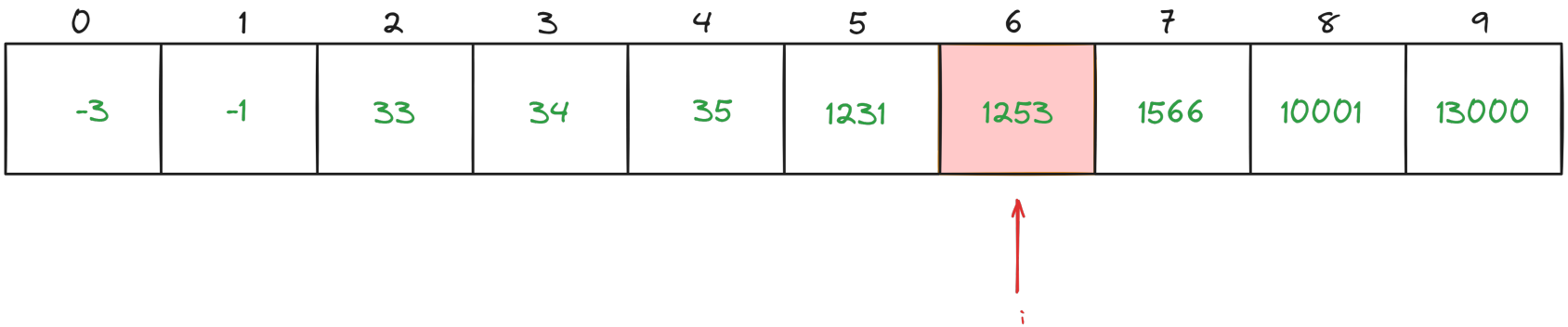
Buscar un elemento: búsqueda lineal



Buscar un elemento: búsqueda lineal



Buscar un elemento: búsqueda lineal

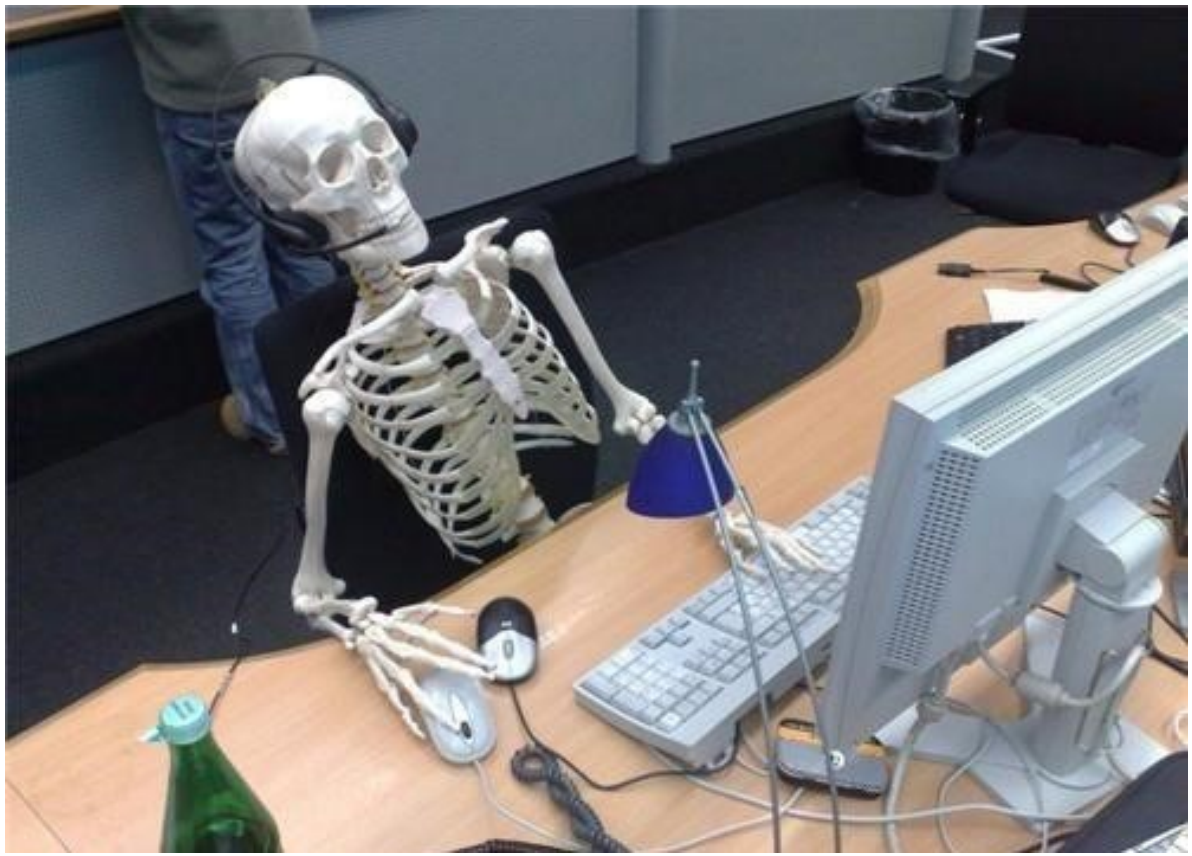


Buscar un elemento: búsqueda lineal

La búsqueda lineal es un algoritmo simple de implementar y que aplica para gran variedad de casos. Pero, qué ocurre si el arreglo sobre el que estamos realizando la búsqueda tiene 100 millones de elementos?

Buscar un elemento: búsqueda lineal

La búsqueda lineal es un algoritmo simple de implementar y que aplica para gran variedad de casos. Pero, qué ocurre si el arreglo sobre el que estamos realizando la búsqueda tiene 100 millones de elementos?



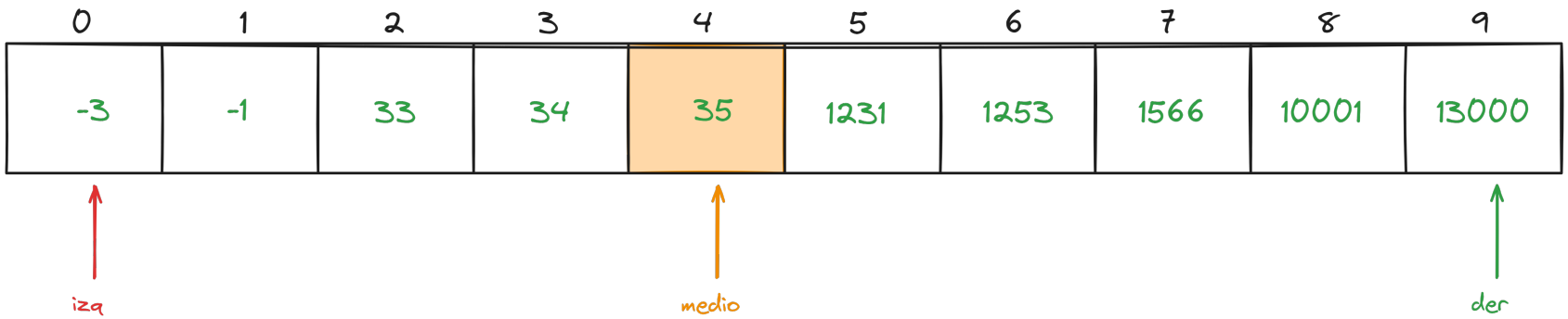
Buscar un elemento: búsqueda binaria

Para los casos en donde el arreglo se encuentre ordenado, podemos aplicar una búsqueda binaria

```
1  // buscar de una manera mas eficiente, rapida.....
2  // -> busqueda binaria
3  // (solo se puede aplicar si el arreglo se encuentra ordenado)
4  // retorna la posicion en la que se encuentra el numero buscado
5  int buscarNumero(const int arr[], int tl, int num) {
6    int izq = 0, der = tl-1;
7    int medio;
8    bool encontrado = false;
9    while(izq<=der && !encontrado) {
10
11     medio = (izq+der)/2;
12
13     if (arr[medio] == num) {
14       encontrado = true;
15     }
16     else if (arr[medio] > num) {
17       der = medio - 1;
18     }
19     else {
20       izq = medio + 1;
21     }
22   }
23
24   return encontrado ? medio : -1;
25 }
```

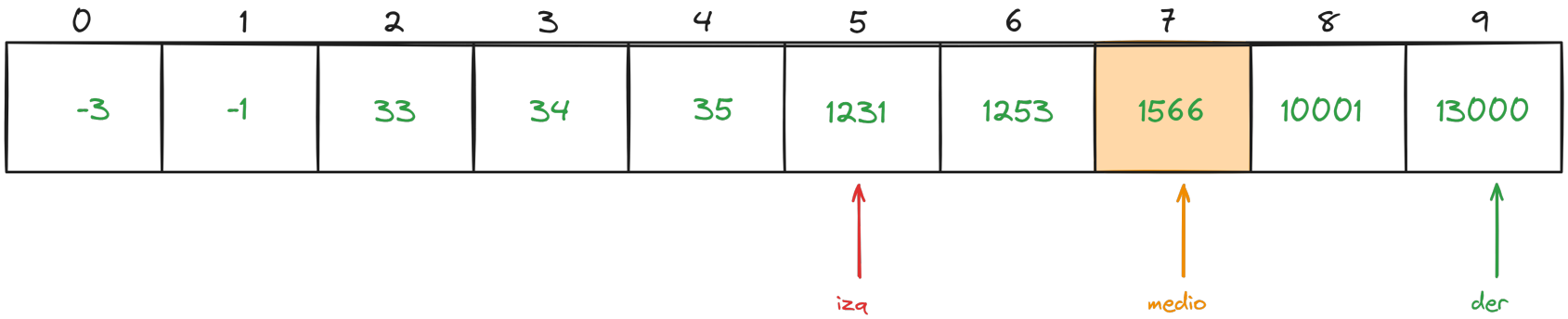
Buscar un elemento: búsqueda binaria

```
1  int buscarNumero(const int arr[], int tl, int num) {
2      int izq = 0, der = tl-1;
3      int medio;
4      bool encontrado = false;
5      while(izq<=der && !encontrado) {
6
7          medio = (izq+der)/2;
8
9          if (arr[medio] == num) {
10             encontrado = true;
11         }
12         else if (arr[medio] > num) {
13             der = medio - 1;
14         }
15         else {
16             izq = medio + 1;
17         }
18     }
19
20     return encontrado ? medio : -1;
21 }
```



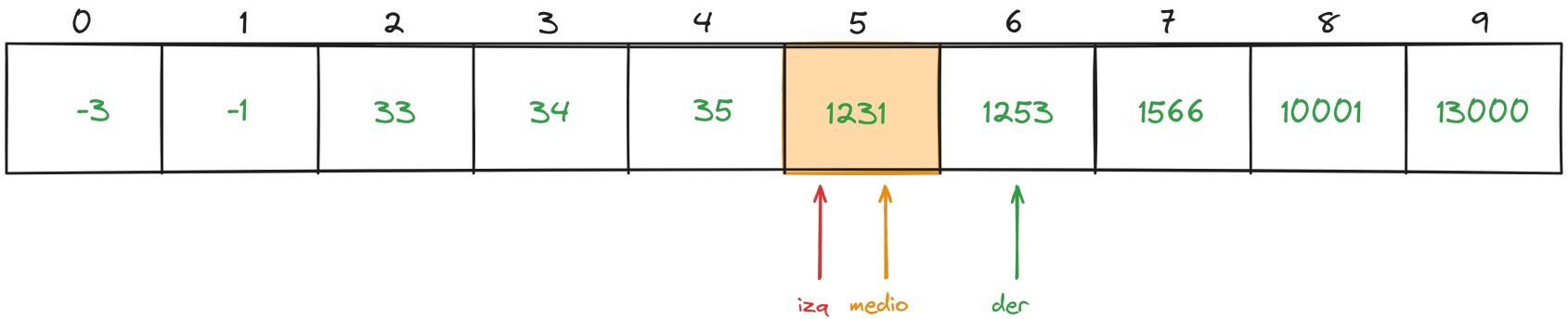
Buscar un elemento: búsqueda binaria

```
1  int buscarNumero(const int arr[], int tl, int num) {
2      int izq = 0, der = tl-1;
3      int medio;
4      bool encontrado = false;
5      while(izq<=der && !encontrado) {
6
7          medio = (izq+der)/2;
8
9          if (arr[medio] == num) {
10             encontrado = true;
11         }
12         else if (arr[medio] > num) {
13             der = medio - 1;
14         }
15         else {
16             izq = medio + 1;
17         }
18     }
19
20     return encontrado ? medio : -1;
21 }
```



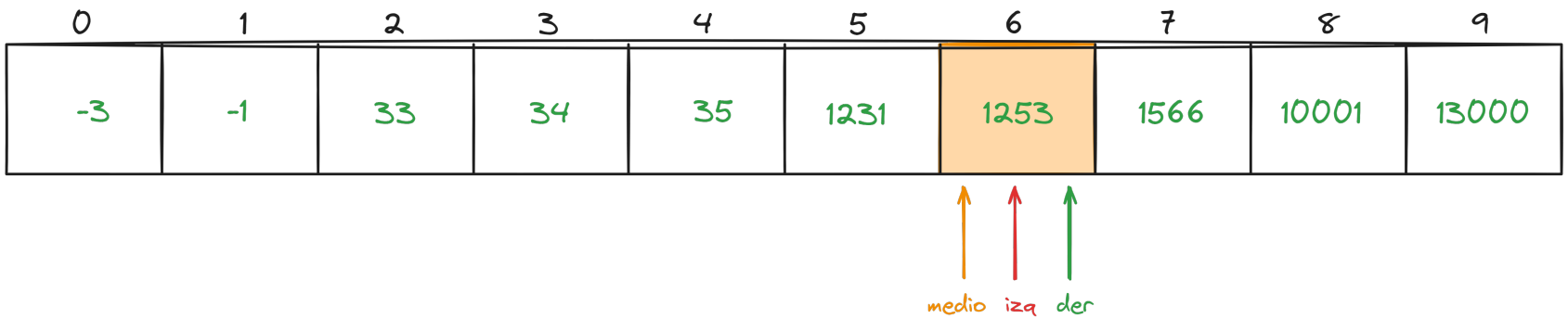
Buscar un elemento: búsqueda binaria

```
1  int buscarNumero(const int arr[], int tl, int num) {
2      int izq = 0, der = tl-1;
3      int medio;
4      bool encontrado = false;
5      while(izq<=der && !encontrado) {
6
7          medio = (izq+der)/2;
8
9          if (arr[medio] == num) {
10             encontrado = true;
11         }
12         else if (arr[medio] > num) {
13             der = medio - 1;
14         }
15         else {
16             izq = medio + 1;
17         }
18     }
19
20     return encontrado ? medio : -1;
21 }
```



Buscar un elemento: búsqueda binaria

```
1  int buscarNumero(const int arr[], int tl, int num) {
2      int izq = 0, der = tl-1;
3      int medio;
4      bool encontrado = false;
5      while(izq<=der && !encontrado) {
6
7          medio = (izq+der)/2;
8
9          if (arr[medio] == num) {
10             encontrado = true;
11         }
12         else if (arr[medio] > num) {
13             der = medio - 1;
14         }
15         else {
16             izq = medio + 1;
17         }
18     }
19
20     return encontrado ? medio : -1;
21 }
```



Ordenamiento

Existen múltiples algoritmos de ordenamiento que pueden aplicarse sobre un arreglo, entre ellos se encuentran:

- BubbleSort (ordenamiento burbuja)
- SelectionSort (ordenamiento por selección)
- MergeSort
- Y más.... (que no los veremos)

De todas formas, para la resolución de ejercicios basta con conocer bien uno. ¿Cuál? el que les resulte más sencillo de entender e implementar.

`https://visualgo.net/en/sorting`

Ordenamiento: BubbleSort

```
1  // ordenar de menor a mayor
2  void bubbleSort(char arr[], int tam) {
3      for (int i = 0; i < tam; i++) {
4          for (int j = 0; j < tam-1; j++) {
5              if (arr[j] > arr[j + 1])
6                  swap(arr[j], arr[j + 1]);
7              }
8      }
9  }
10
11 // ordenar de mayor a menor
12 void bubbleSort2(char arr[], int tam) {
13     for (int i = 0; i < tam; i++) {
14         for (int j = 0; j < tam-1; j++) {
15             if (arr[j] < arr[j + 1])
16                 swap(arr[j], arr[j + 1]);
17             }
18     }
19 }
```

Ordenamiento: SelectionSort

```
1  // ordenar de menor a mayor
2  void selectionSort(int arr[], int tam) {
3      for (int i = 0; i < tam; i++) {
4          int indiceMenor=-1;
5          for (int j = i; j < tam; j++) {
6              if (indiceMenor == -1 || arr[j] < arr[indiceMenor])
7                  indiceMenor = j;
8          }
9          swap(arr[i], arr[indiceMenor]);
10     }
11 }
12
13 // ordenar de mayor a menor
14 void selectionSort2(int arr[], int tam) {
15     for (int i = 0; i < tam; i++) {
16         int indiceMayor=-1;
17         for (int j = i; j < tam; j++) {
18             if (indiceMayor == -1 || arr[j] > arr[indiceMayor])
19                 indiceMayor = j;
20         }
21         swap(arr[i], arr[indiceMayor]);
22     }
23 }
```

Ordenamiento: MergeSort

```
1  // divide el arreglo en dos subarreglos, los ordena y los une
2  void mergeSort(int arr[], int l, int r) {
3      if (l < r) {
4          // m es el punto donde el array se divide en dos
5          int m = l + (r - l) / 2;
6
7          mergeSort(arr, l, m);
8          mergeSort(arr, m + 1, r);
9
10         // Unir los sub-arreglos ordenados
11         merge(arr, l, m, r);
12     }
13 }
```

https://www.programiz.com/dsa/merge-sort

Ordenamiento: MergeSort

```
1  void merge(int arr[], int p, int q, int r) {
2      // Crear arreglos L = A[p..q] and M = A[q+1..r]
3      int n1 = q - p + 1, n2 = r - q;
4      int L[10000], M[10000];
5      for (int i = 0; i < n1; i++)
6          L[i] = arr[p + i];
7      for (int j = 0; j < n2; j++)
8          M[j] = arr[q + 1 + j];
9
10     int i=0, j=0, k=p;
11     // Hasta que no alcancemos el final de L or M procesamos ambos juntos
12     while (i < n1 && j < n2) {
13         if (L[i] <= M[j]) {
14             arr[k] = L[i];
15             i++;
16         } else {
17             arr[k] = M[j];
18             j++;
19         }
20         k++;
21     }
22     // insertamos los elementos restantes en A[p..r]
23     while (i < n1) {
24         arr[k] = L[i];
25         i++; k++;
26     }
27     while (j < n2) {
28         arr[k] = M[j];
29         j++; k++;
30     }
31 }
```