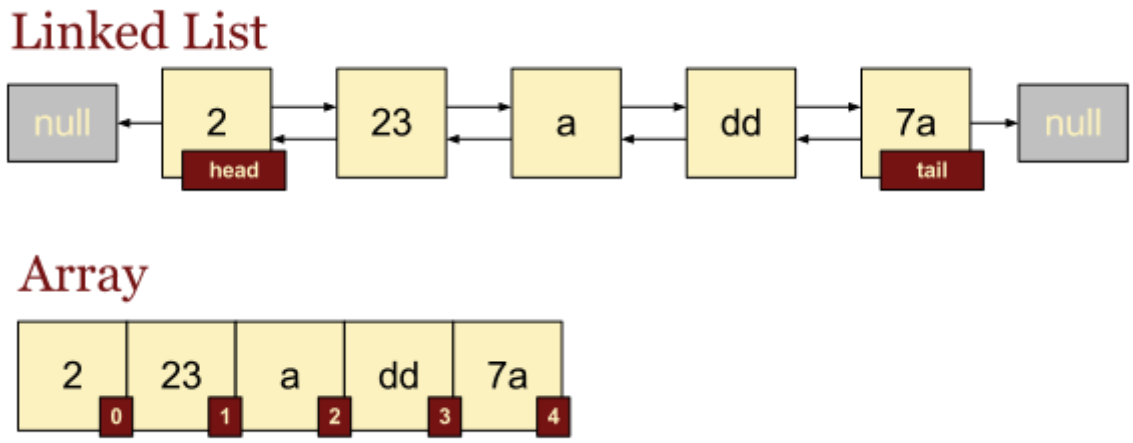
ArrayList vs **LinkedList**

LinkedList y ArrayList son dos implementaciones diferentes de la interfaz List. LinkedList usa internamente una lista doblemente enlazada, mientras que ArrayList usa un array que cambia de tamaño dinámicamente.

Gráficamente podemos representarlas de la siguiente forma:



LinkedList permite **eliminar e insertar elementos en tiempo constante** usando iteradores, pero el acceso es secuencial por lo que el tiempo que se tarda en encontrar un elemento es proporcional al tamaño de la lista.

Normalmente, la complejidad de las operaciones mencionadas anteriormente promedio sería O(n/2), sin embargo, si usamos una lista doblemente enlazada, el recorrido puede ser desde el principio o desde el final de la lista reduciendo la complejidad a O(n/4).

Por otro lado, ArrayList ofrece **acceso en tiempo constante** O(1), pero para añadir o eliminar un elemento en cualquier posición, que no sea la última, es necesario mover elementos. Además si el array ya está lleno es necesario crear uno nuevo con mayor capacidad y copiar los elementos existentes.

Complejidad

LinkedList

Operación	Complejidad Promedio
get	O(n/4)
add(E element)	O(1)
add(int index, E element)	O(n/4)
remove(int index)	O(n/4)
Iterator.remove()	**O(1)**
ListIterator.add(E element)	**O(1)**

ArrayList

Operación	Complejidad Promedio
get	**O(1)**
add(E element)	**O(1)**
add(int index, E element)	O(n/2)
remove(int index)	O(n/2)
Iterator.remove()	O(n/2)
remove(int index)	O(n/2)

Estos métodos son aplicables a las clases genéricas de Java **LinkedList** y **ArrayList**, sin embargo, no son exclusivos de Java, es posible encontrar implementaciones en otros lenguajes de programación, por ejemplo en C# existen LinkedList y List.

Ventajas y desventajas

LinkedList

Ventajas	Desventajas
Añadir y borrar elementos con un iterador	Uso de memoria adicional por las referencias a los elementos anterior y siguiente
Añadir y borrar elementos al final de la lista	El acceso a los elementos depende del tamaño de la lista

ArrayList

Ventajas	Desventajas
Añadir elementos	Coste adicional al añadir o eliminar elementos
Acceso a elementos	La cantidad de memoria considera la capacidad definida para el ArrayList, aunque no contenga elementos

Nota: Si especificas el tamaño de un ArrayList en el momento de crearlo se reduce la cantidad de memoria usada ya que no necesitará aumentar de tamaño y copiar los elementos del array.

Conclusión…

En la práctica, en la mayoría de los casos es mejor optar por un ArrayList porque el tiempo de las operaciones y uso de memoria es menor que en LinkedList, resumiendo: **si no estás seguro de cuál usar, inclínate por un ArrayList**.

Lo anterior no quiere decir que LinkedList nunca se use, existen algunos casos muy específicos donde es la mejor opción, por ejemplo la pila de llamadas del lenguaje C está implementada usando una estructura de datos con estas características.