Question

May a set contain duplicates?

Question

May a set contain duplicates?

Answer

No.

Question

May a set contain duplicates?

Answer

No.

Question

Are the elements of a set ordered?

Question

May a set contain duplicates?

Answer

No.

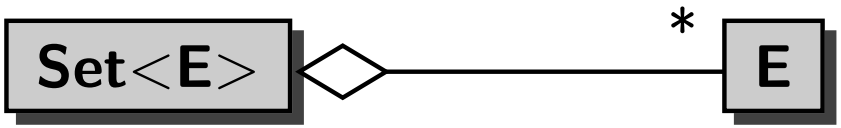
Question

Are the elements of a set ordered?

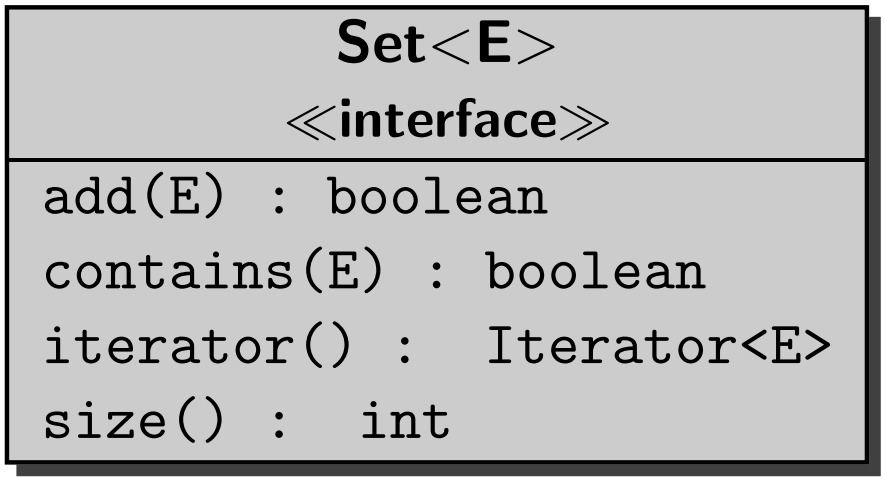
Answer

No.

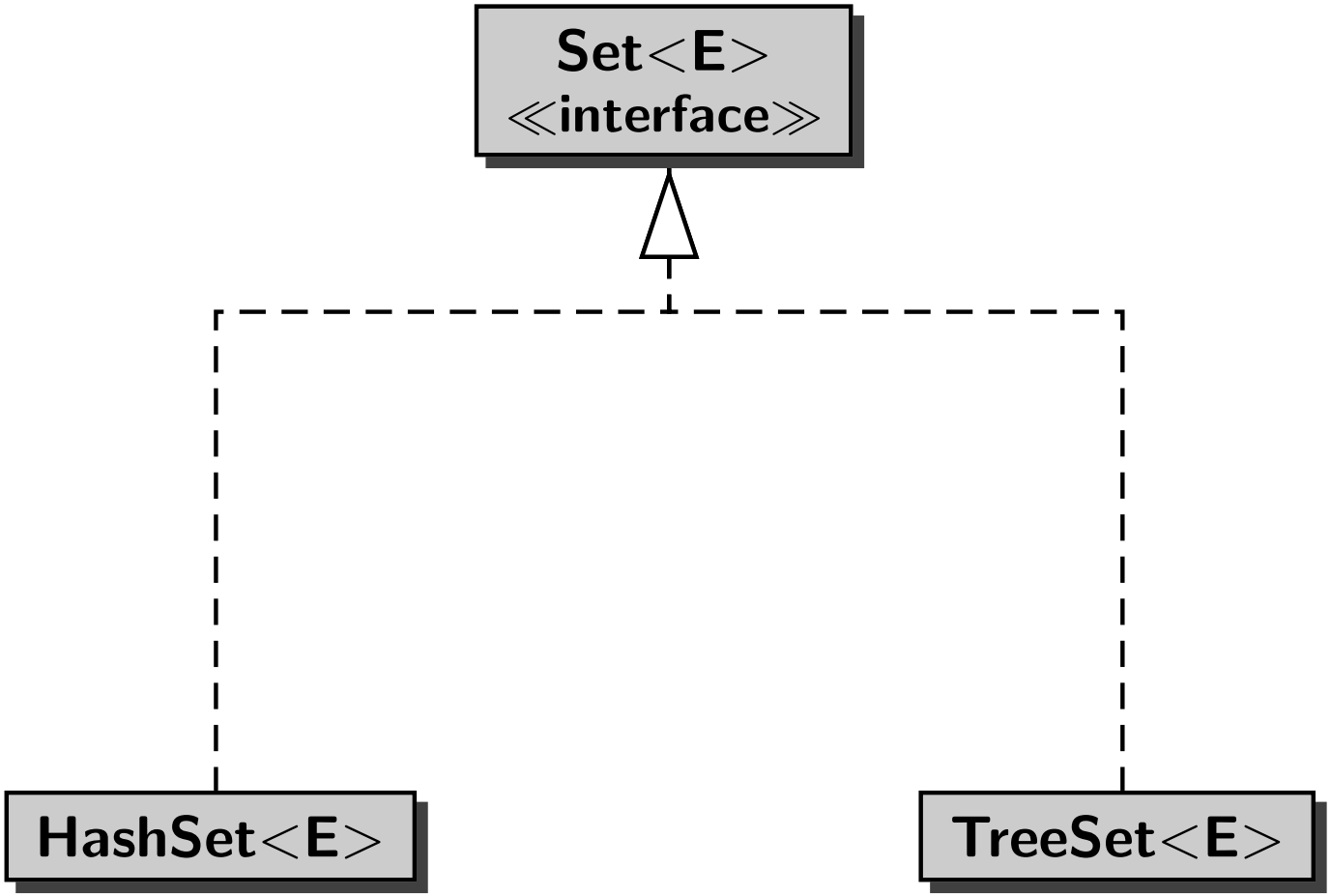
Sets



Methods of Set



Sets



HashSet or TreeSet?

- Adding to or deleting from or searching in a set takes `HashSet` $O(1)$ (expected), whereas it takes `TreeSet` $O(\log(n))$, where n is the size of the set.
- `TreeSet` keeps the elements sorted, but `HashSet` does not.

$O(\log(n))$

Question

What does "adding to a set takes `TreeSet` $O(\log(n))$" mean?

$O(\log(n))$

Question

What does "adding to a set takes `TreeSet` $O(\log(n))$" mean?

Let $a : \mathbb{N} \to \mathbb{N}$ be the function that given the size of the set n returns the number $a(n)$ of basic instructions that need to be executed to add an element to the set.

$O(\log(n))$

Question

What does "adding to a set takes `TreeSet` $O(\log(n))$" mean?

Let $a : \mathbb{N} \to \mathbb{N}$ be the function that given the size of the set n returns the number $a(n)$ of basic instructions that need to be executed to add an element to the set.

Answer

$a \in O(\log(n))$.

$O(\log(n))$

Question

What does $a \in O(\log(n))$ mean?

$O(\log(n))$

Question

What does $a \in O(\log(n))$ mean?

Answer

$$\exists M \in \mathbb{N} : \exists F \in \mathbb{N} : \forall n \geq M : a(n) \leq F \times \log(n).$$

There exist M and F so that adding an element to a set of size n, where $n \geq M$, takes at most $F \times \log(n)$ basic instructions. That is, the number of instructions grows logarithmically in the size of the set.

ITunes library

Problem

Print each song of each playlist of an iTunes library. Each song should appear on a separate line. Playlists should be separated by a blank line.

ITunes library

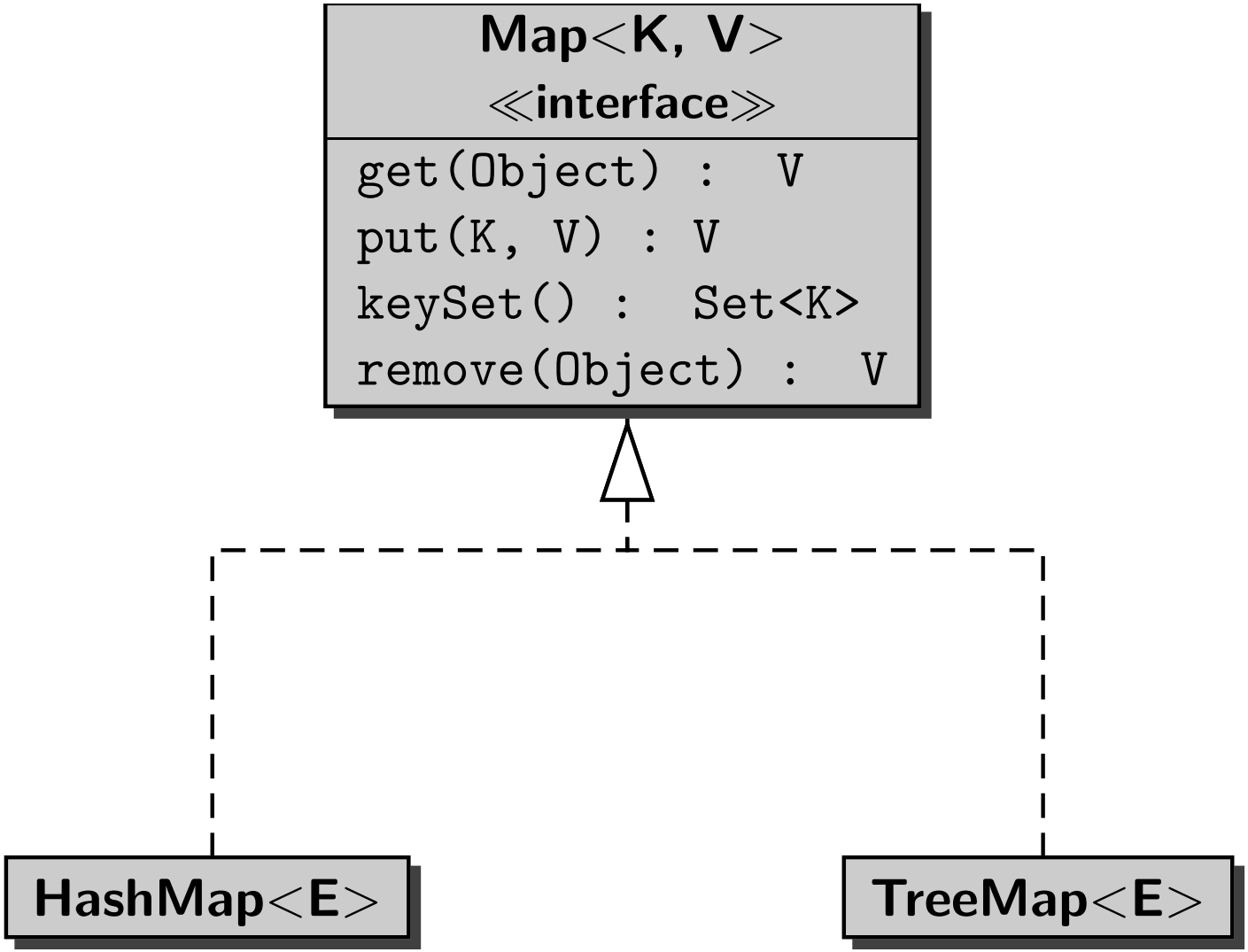
Problem

Print each song of each playlist of an iTunes library. Each song should appear on a separate line. Playlists should be separated by a blank line.

Problem

Determine whether each playlist of an iTunes library contains duplicates.

Map


Map<K, V>
≪interface≫
get(Object) : V
put(K, V) : V
keySet() : Set<K>
remove(Object) : V
HashMap<E>
TreeMap<E>

Searching

Problem

Check whether a given word appears in the book entitled "The Java Language Specification, Java SE 7 Edition." The book is contained in the file `jls7.pdf`.

Question

Consider

```
List<String> words = ...;
String search = ...;
boolean found = false;
for (String word : words)
{
   found = word.equals(seach) || found;
}
```

Given that the list contains n elements, how many times is the method `equals` invoked?

Search

Question

Consider

```
List<String> words = ...;
String search = ...;
boolean found = false;
for (String word : words)
{
   found = word.equals(seach) || found;
}
```

Given that the list contains n elements, how many times is the method `equals` invoked?

Answer

n times.

Binary search

```
int index = Collections.binarySearch(list, element);
```

- The list must be sorted.
- If the element is contained in the list then the method returns the index at which the element can be found.
- If the element is not in the list then the method returns -1.

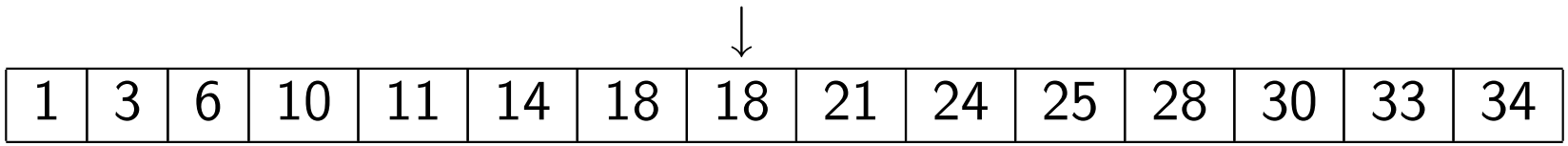
Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```

1	3	6	10	11	14	18	18	21	24	25	28	30	33	34

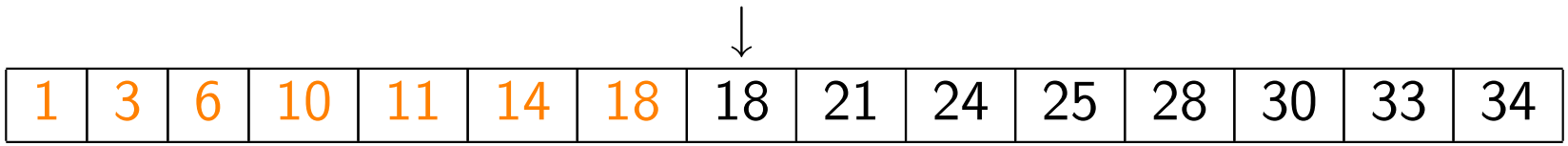
Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```



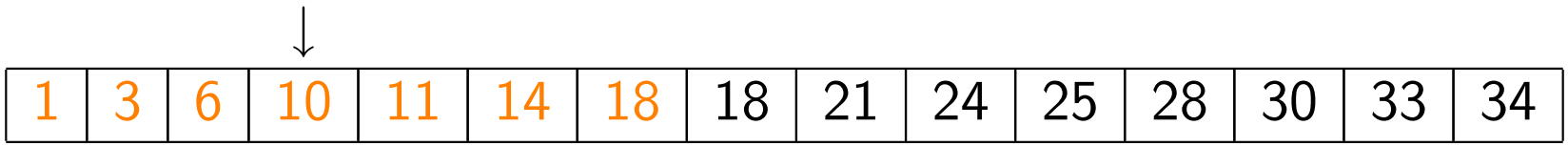
Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```



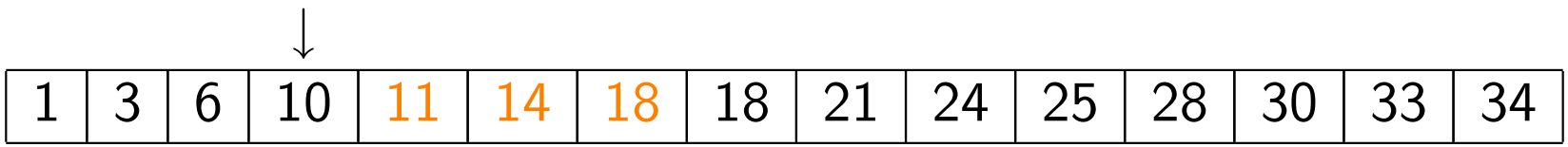
Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```



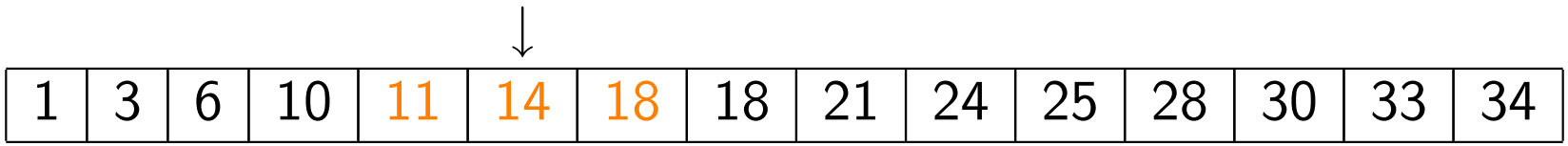
Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```



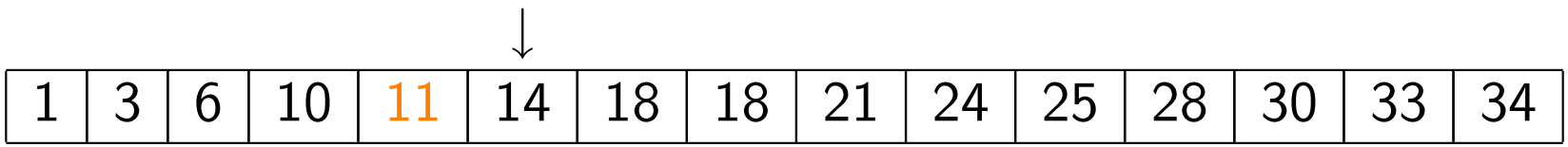
Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```



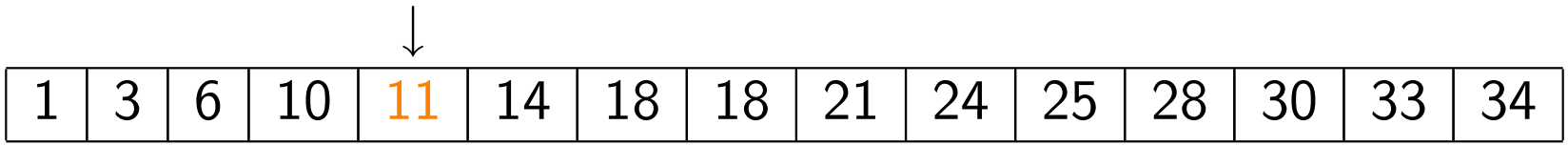
Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```



Binary search

```
final int ELEMENT = 11;
int index = Collections.binarySearch(list, ELEMENT);
```



`index` gets assigned the value 4.

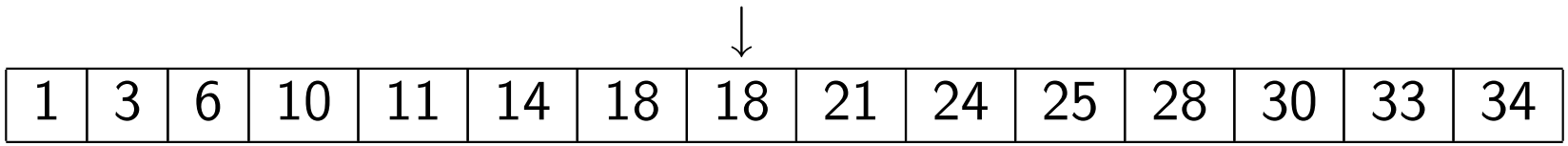
Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```

1	3	6	10	11	14	18	18	21	24	25	28	30	33	34

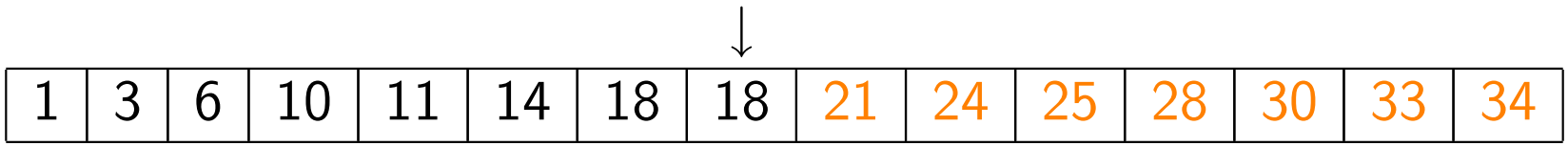
Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```



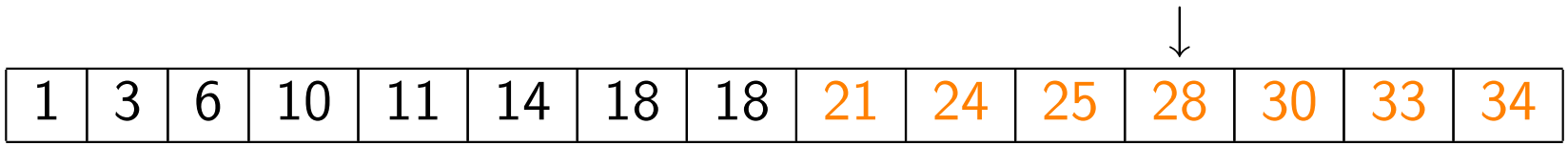
Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```



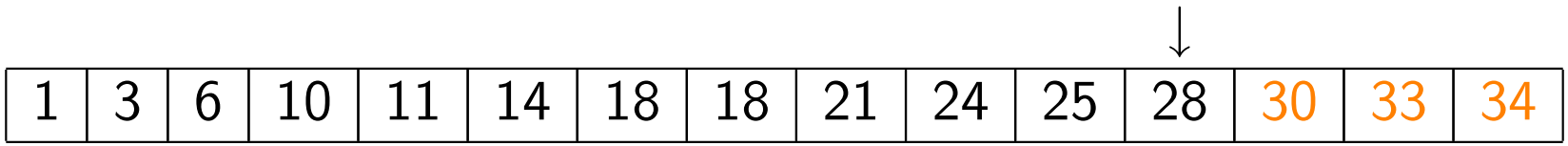
Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```



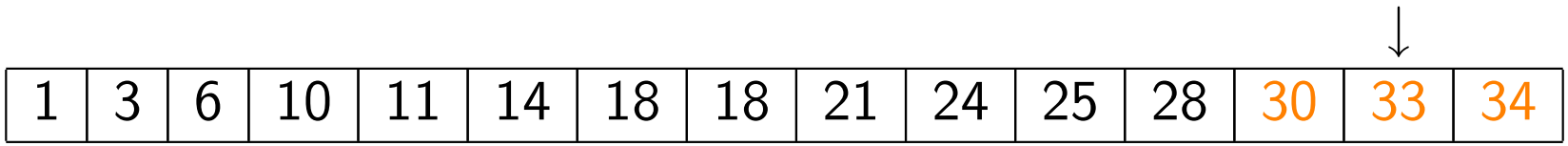
Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```



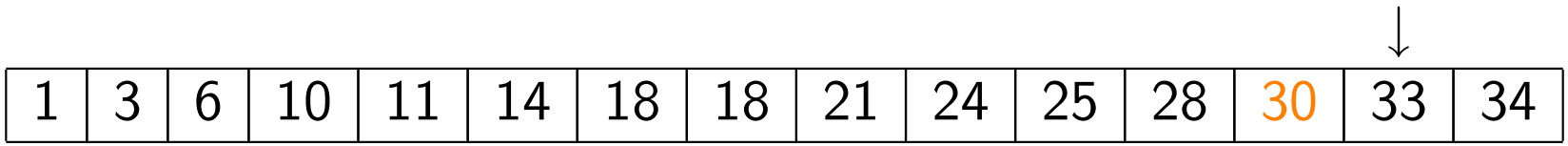
Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```



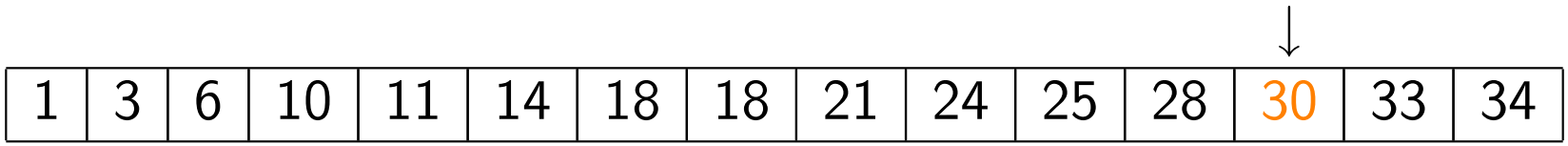
Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```



Binary search

```
final int ELEMENT = 32;
int index = Collections.binarySearch(list, ELEMENT);
```



`index` gets assigned the value -1.

System.nanoTime()

This method returns the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds.

```
long start = System.nanoTime();
...
long stop = System.nanoTime();
// stop - start is an estimate of the number of
// nanoseconds it took to execute ...
```

To do

- Study the remainder of Chapter 10.