



```
fun printFirst100(ns: List<Int>) {  
    for (i in 0..100)  
        println(ns[i])  
}
```

treeSet += "a new element"

**TimeComplexities Game**





O(1)

$O(\log n)$



$O(n)$

$O(n^2)$

list.sort()

`list.sorted()` has a medium time complexity of  $O(n \log n)$   
why is QuickSort being used and not a  $O(n \log n)$  sorting algorithm?

hashMap["key"]

LinkedList[index]

```
list.flatMap { list }  
      .forEach { println(it) }
```

```
list.forEach{ print(it) }
```



















































hashSet += "a new element"

# Time Complexities Game

$O(1)$

```
fun printFirst100(ns: List<Int>) {  
    for (i in 0..100)  
        println(ns[i])  
}
```

```
hashMap["key"]
```

```
hashSet += "a new element"
```

$O(n)$

```
linkedList[index]
```

```
list.forEach { println(it) }
```

$O(\log n)$

```
treeSet += "a new element"
```

$O(n^2)$

```
list.sorted()
```

```
list.flatMap { list }  
    .forEach { println(it) }
```

`list.sorted()` has a medium time complexity of  $O(n \log n)$   
why is QuickSort being used and not a  $O(n \log n)$  sorting algorithm?



# Time Complexities Game

$O(1)$

```
fun printFirst100(ns: List<Int>) {  
    for (i in 0..100)  
        println(ns[i])  
}
```

```
hashMap["key"]
```

```
hashSet += "a new element"
```

$O(n)$

```
linkedList[index]
```

```
list.forEach { println(it) }
```

$O(\log n)$

```
treeSet += "a new element"
```

$O(n^2)$

```
list.sorted()
```

```
list.flatMap { list }  
    .forEach { println(it) }
```

`list.sorted()` has a medium time complexity of  $O(n \log n)$   
why is QuickSort being used and not a  $O(n \log n)$  sorting algorithm?