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

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

# Time Complexities Game


O(1)


O(log n)


O(n)


O(n2)

list.sorted()

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"]
```

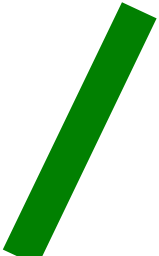
```
linkedList[index]
```

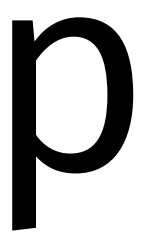
```
list.flatMap { list }
    .forEach { println(it) }
```

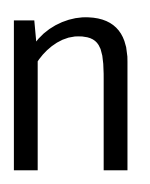
```
list.forEach { println(it) }
```

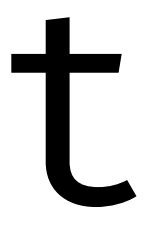
```
hashSet += "a new element"
```

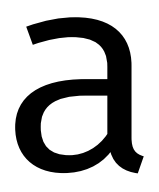
```
treeMap["key"]
```

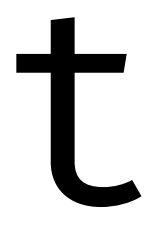


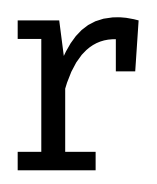


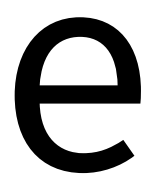


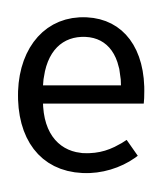




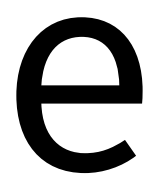


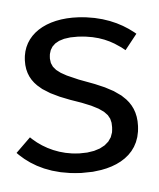


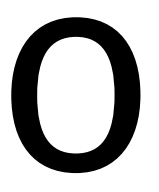


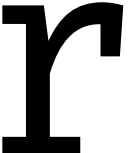


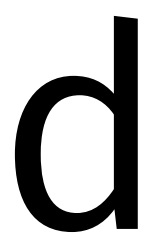
S











```
//print a treeset sorted
```

# 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(log n)

```
treeSet += "a new element"
treeMap["key"]
```

O(n)

```
linkedList[index]
list.forEach { println(it) }
//print a treeset sorted
```

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(log n)

```
treeSet += "a new element"
treeMap["key"]
```

O(n)

```
linkedList[index]
list.forEach { println(it) }
//print a treeset sorted
```

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?