

# Work on project. Stage 2/4: Jumping bubbles

Project: [Phone Book](#)

Medium [?](#)

236 users solved this problem. Latest completion was about 6 hours ago.

## Description

You have to iterate over each element of the number list every time you want to find someone's number. This is the only way to search if your list contains unordered data. Any number can be anywhere on the list, so you have to check every element.

At this stage, you should sort the list of numbers alphabetically by the owner's name. Sort the list using the **bubble sort** algorithm and search in the list using the **jump search** algorithm.

After sorting, search for 500 phone numbers using the list from the previous stage. Note how long it takes to sort the list, and also measure the time the program spends searching. Don't include the sorting time in the searching time, because the list of numbers stays sorted after every search request. If you want to save the sorted list into the file so you don't have to sort it again, do not override the file that contains the unsorted list of phone numbers. This file will also be required in the next stage.

If the sorting process takes too long (more than 10 times longer than all 500 iterations of the linear search), you should stop sorting and use the linear search. Look at the second example to see what you need to output.

9 / 9 Prerequisites

- ✓ [Jump search](#) Stage 2
- ✓ [The sorting problem](#) Stage 2
- ✓ [Bubble sort](#) Stage 2
- ✓ [Floating-point types](#) Stage 2
- ✓ [The Math library](#) Stage 2

[Show all](#)

## Example

Output both approaches one after another and see which one is faster. The output example is shown below. Note that you can get totally different sorting and searching times!

Example 1:

```

1 Start searching (linear search)...
2 Found 500 / 500 entries. Time taken: 1 min. 56 sec. 328 ms.
3
4 Start searching (bubble sort + jump search)...
5 Found 500 / 500 entries. Time taken: 9 min. 15 sec. 291 ms.
6 Sorting time: 8 min. 45 sec. 251 ms.
7 Searching time: 0 min. 30 sec. 40 ms.

```

Example 2:

```

1 Start searching (linear search)...
2 Found 500 / 500 entries. Time taken: 2 min. 01 sec. 134 ms.
3
4 Start searching (bubble sort + jump search)...
5 Found 500 / 500 entries. Time taken: 22 min. 14 sec. 482 ms.
6 Sorting time: 20 min. 12 sec. 251 ms. - STOPPED, moved to linear search
7 Searching time: 2 min. 02 sec. 231 ms.

```

HINT by redstrike

`directory.txt` contains 1.014.130 records. Sorting it using the Bubble Sort algorithm costs about 13.5 hours on my computer. (100 full iterations cost 4 seconds).  $O(n^2)$ .

So, to save time, you should only consider Example 2: stop sorting and move to linear search.

See the next hint

↙ Write a program

[Code Editor](#) [IDE](#)

### CONNECTION STATUS

IDE / Checking the plugin's status



Solve in IDE

src/phonebook/Main.kt

```
1 package phonebook
2
3 import java.io.File
4 import kotlin.math.sqrt
5
6 fun main() {
7     val directory = File("/Users/skubatko/Downloads/directory.txt").readLines()
8     val find = File("/Users/skubatko/Downloads/find.txt").readLines()
9
10    println("Start searching (linear search)...")
11
12    val start = System.currentTimeMillis()
13
14    val result = linearSearch(find, directory)
15
16    val linearSearchTime = System.currentTimeMillis() - start
17
18    val timeTaken = String.format("%1$tM min. %1$tS sec. %1$tL ms.", linearSearchTime)
19
20    println("Found ${result.size} / ${find.size} entries. Time taken: $timeTaken")
21
22    println()
23
24
25    bubbleSortJumpSearch(find, directory, linearSearchTime)
26
27}
28
29
30
31 private fun linearSearch(
32     find: List<String>,
33     directory: List<String>
34 ): Map<String, String> {
35
36     val result = mutableMapOf<String, String>()
37
38     for (person in find) {
39
40         for (line in directory) {
41
42             if (line.contains(person)) {
43
44                 result[person] = line.split(" ")[0]
45
46                 break
47             }
48         }
49     }
50
51     return result
52 }
53
54
55 private fun bubbleSortJumpSearch(
56     find: List<String>,
57     directory: List<String>,
58     linearSearchTime: Long
59 ) {
60
61     println("Start searching (bubble sort + jump search)...")
62
63     val start = System.currentTimeMillis()
```

```
4  
5  
6  
7     val (sorted, isSuccessfull) = bubbleSort(directory, linearSearchTime)  
8  
9  
0     var result = mutableMapOf<String, String>()  
1  
2         if (isSuccessfull) {  
3             for (person in find) {  
4                 result[person] = jumpSearch(sorted, person)  
5             }  
6         } else {  
7             val data = linearSearch(find, directory)  
8             result = data.toMutableMap()  
9         }  
10  
11     val totalTime = System.currentTimeMillis() - start  
12  
13     val totalTimeTaken = String.format("%1$tM min. %1$tS sec. %1$tL ms.", totalTime)  
14     println("Found ${result.size} / ${find.size} entries. Time taken: $totalTimeTaken")  
15  
16     val sortingTimeTaken = String.format("%1$tM min. %1$tS sec. %1$tL ms.", sortingTime)  
17     if (isSuccessfull) {  
18         println("Sorting time: $sortingTimeTaken")  
19     } else {  
20         println("Sorting time: $sortingTimeTaken - STOPPED, moved to linear search")  
21     }  
22  
23     val searchingTimeTaken = String.format("%1$tM min. %1$tS sec. %1$tL ms.", totalTime - sortingTime)  
24     println("Searching time: $searchingTimeTaken")  
25 }  
26  
27 fun jumpSearch(sorted: List<String>, person: String): String {  
28     val step = sqrt(sorted.size.toDouble()).toInt()  
29     for (i in 0..sorted.lastIndex step step) {  
30         val record = sorted[i]  
31  
32         if (record.contains(person)) {  
33             return record.split(" ")[0]  
34         }  
35     }  
36 }
```

```
8
4     if (person < personRecordName(record)) {
8
5         for (j in (i - 1) downTo (i - step)) {
8
6             if (sorted[j].contains(person)) {
8
7                 return sorted[j].split(" ")[0]
8
8             }
8
9         }
9
0     return ""
1
2 }
9
3
4
5
6 private fun bubbleSort(
9
7     directory: List<String>,
9
8     linearSearchTime: Long
9
9 ): Pair<List<String>, Boolean> {
1
0
0     val timeLimit = linearSearchTime * 10
1
0
1
1
0
2     val result = directory.toMutableList()
1
0
3
1
0
4     val start = System.currentTimeMillis()
1
0
5     for (pos in result.lastIndex downTo 1) {
1
0
6         for (i in 0 until pos) {
1
0
7             if (personRecordCompare(result[i], result[i + 1]) > 0) {
1
0
8                 switch(result, i)
1
0
9                 val elapsed = System.currentTimeMillis() - start
1
1
0                 if (timeLimit < elapsed) {
1
1
1                     return Pair(directory, false)
1
1
2                 }
1
1
3             }
1
1
4 }
```

```
1
1
5     }
1
1
1
6
1
1
7     return Pair(result.toList(), true)
1
1
8 }
1
1
9
1
1
2
0 fun personRecordCompare(record1: String, record2: String): Int {
1
2
1     return personRecordName(record1).compareTo(personRecordName(record2))
1
2
2 }
1
2
3
1
1
2
4 fun personRecordName(record: String): String {
1
2
5     val idx = record.indexOf(" ")
1
2
6     return record.substring(idx + 1)
1
2
7 }
1
2
8
1
1
2
9 fun switch(list: MutableList<String>, idx: Int) {
1
3
0     val value = list[idx]
1
3
1     list[idx] = list[idx + 1]
1
3
2     list[idx + 1] = value
1
3
3 }
```

✓ Correct.

Your practice is really paying off. Well done!

11 users liked this problem. 11 didn't like it. What about you?



[Continue](#)

[Solutions \(26\)](#)

[Comments \(12\)](#)

[Hints \(3\)](#)

[Useful links \(2\)](#)

[Solutions \(26\)](#)

[Show discussion](#)