1. *top* = [] // Θ(1)
2. *maxheap*[]= **heapify**(*array*) // Θ(*n*)
3. for i = 0 to *k* // *k* iterations × Θ(lg *n*)⟶ Θ(*k* lg *n*)
4. {
5. *top*[i] = *maxheap*[1] // Θ(1)
6. **DeleteMax**(*maxheap*) // Θ(lg *n*)
7. }
8. return *top* // Θ(1)

The above pseudocode represents an algorithm that could be used to solve the top-*k* search problem. That is, this algorithm takes in an unsorted set of values ***array*** of length ***n*** and returns the ***k*** largest values in the array, here stored in a secondary array ***top***. To accomplish this, the algorithm first calls the **heapify()** function to turn the unsorted array into a sorted maximum-heap, stored in the ***maxheap*** array. According to the lecture slides, this runs in **Θ(*n*)** time for an unsorted array. Then, for ***k*** iterations, the root of the maximum-heap (index 1 of the ***maxheap*** array) is stored in the ***top*** array before that value is removed from the heap by the **DeleteMax()** function. Also according to the lecture slides, this function runs in **Θ(lg *n*)** time for a heap, since that is what we are now dealing with. Since the **for** loop iterates ***k*** times and each iteration has **Θ(lg *n*)** running time, the total running time there is **Θ(*k* × lg *n*)**. Considering the **heapify()** function is only called once during execution, the total running time would be **Θ(*n* + *k* lg *n*)**.