1.
    a. First sort the numbers using a comparison-based sorting algorithm, and then return the k smallest numbers.

Since we are using comparison-based sorting algorithms, from Decision Tree (L3 – stableSorting), we know that best worst-case running time for these type of sorting algorithms is Omega (n lgn). Thus, algorithms with this time complexity for their upper-bounds (O (n lgn)) such as HeapSort and MergeSort are asymptotically optimal comparison sorting algorithms. Quicksort do not satisfy this constraint since it has O(n^2) time complexity for its worst-case scenario, which is 'already sorted' or 'reverse sorted' lists as input. However, using randomization expected time complexity can be reduced to Theta (n lgn), but since expected time complexity is not the same as best-worst case time complexity, we should not consider this as a solution.

Comparing HeapSort and MergeSort, they both have their benefits as well as drawbacks. Most importantly, although their best worst-case time complexity are the same for sorting purposes, when retrieving the *k* smallest elements, HeapSort requires O(k lgn) operations as it has to satisfy heap property. Other trade-offs are as follows:

HeapSort: In-place, no auxiliary storage, requires 'build', selecting requires (k lgn), not stable
MergeSort: external sorting, O(n) space complexity, recursion, stable

For different systems, better algorithm may differ between the two. For example, if external storage is an issue, we should opt for HeapSort as it's in-place, thus, not require extra space. On the other hand, if we were to know that *k* will be large, we should opt for MergeSort.

Since we are only considering the best asymptotic worst-case running time, and not given and constraints such as space complexity, MergeSort will be better than HeapSort since:

MergeSort: Dividing (O(lgn)) * Merge(O(n)) + Select(O(k)) => O(n lgn + k)
HeapSort: Build(O(n)) + Sorting (n lgn) + Select(O (k lgn)) => O (n + nlgn + k lgn)

Since, O(n lgn + k) is better than O(n lgn + k lgn), MergeSort is my preferred algorithm.

[aside:
However, if the question did not specify that we should first sort, then return k smallest numbers, but instead only asked for returning k smallest number, then heapsort would be better choice as it only requires:

Build (O (n)) + SelectMin(k lgn) => O (n + k lgn), and for cases of k < n, it would be preferable.
]

Analysis is given above

b. First use an order-statistics algorithm to find the k'th smallest number, then partition around that number to get the k smallest numbers, and then sort these k smallest numbers using a comparison-based sorting algorithm.

Since we are asked to find k smallest numbers with an order-statistic algorithm, we can use SELECT to solve this in O(n) time complexity.

1. SELECT (k, n), gives k'th smallest element = Theta(n)
2. Partition = O(n), now I have k smallest numbers
3. Sort with Merge sort = O(k lgk)

If k is fixed => O(n)
Else if both approach to infinity:
=> Theta(n) + O(n) + O(k lgk) = O(k lgk)

Problem 2 (Linear-time sorting) (a) How can you modify the radix sort algo- rithm for integers, to sort strings? Please explain the modifications.

2. (b) Illustrate how your algorithm sorts the following list of strings ["BATURAY", "GORKEM", "GIRAY", "TAHIR", "BARIS"]. Please show every step of your algorithm.
3. (c) Analyze the running time of the modified algorithm.

ALGO

```
// find longest name len

Int maxNameLen = 0

For each name:

    İf len(name) > maxNameLen

        maxNameLen = len(name)


// modify short names to NAME[AAAA..]

For each name:

    İf len(name) < maxNameLen

        toAdd = maxNameLen – len(Name)

        Name = Name + [A]*toAdd


// change char to asciiNo

For name in NameList:

    For i in range(maxNameLen):

        Char c = ASCII(name[i])

        İnt asciNo = ‘A’ – c

        Name[i] = asciNo


// apply radixsort

    // instead of comparing digits, use asciiNo’s
```

EXAMPLE: BATURAY , BARISAA , BARISAY

1. BARISAA – BARISAY – BATURAY
2. BARISAA – BARISAY – BATURAY
3. BATURAY – BARISAA -BARISAY
4. BARISAA – BARISAY – BATURAY
5. BARISAA – BARISAY – BATURAY
6. BARISAA – BARISAY – BATURAY
7. BARISAA – BARISAY – BATURAY

RUNNING TIME:

Modifications are:

Find longest name = O(n) where n = # of names

Modify short names = O(n)

Char to int = O(n*k) where k = maxNameLen

Apply radix sort:

ZORT