1. Suppose we run quicksort, as presented in the lectures and in the class notes, to sort the following array:

\[62, 98, 33, 21, 60, 96, 75, 46]\n
(a) Show the contents of the array after the split step at the top level is complete, immediately before the first recursive call.

(b) Draw the binary tree of lists for this invocation of quicksort, as described in the lectures and the class notes in the analysis of quicksort (slide 2-37).

2. If \( S \) be a set of integers and \( x \) is an integer, the predecessor of \( x \) in \( S \) is defined to be the largest element of \( S \) that is \( \leq x \) if such an integer exists, and \( -\infty \) otherwise.

Consider the following problem:

We are given a set \( S \) of integers, represented as a sorted list. We are also given a list \( Q \) of query values, which is also a sorted list. We want to output a list of pairs, where each query value is paired with its predecessor in \( S \).

As an example, if \( S = \{2, 3, 5, 7, 11\} \) and \( Q = [1, 3, 8, 9, 20] \), the output would be \([(1, -\infty), (3, 3), (8, 7), (9, 7), (20, 11)]\).

Describe an algorithm that solves this problem in \( O(n + k) \) worst-case time, where \( n \) is the number of items in the list \( Q \) and \( k \) is the number of elements in the set \( S \).
3. Suppose you have an array of \( n \) items (the “member items”) and a separate list of \( k \) items (the “test items”). The member items are not stored in any particular order. You want to know which of the \( k \) test items belong to the set of member items. Here are two different ways to solve the problem:

- **Algorithm 1:** Look up each test item in the array of member items, using sequential search.
- **Algorithm 2:** Sort the array of member items, using an optimal comparison-based sorting algorithm, and then look up each test item in the array of member items using binary search.

Answer the following questions, and briefly justify your answer to each part.

(a) What is the worst-case running time of Algorithm 1 (asymptotically, in terms of \( n \) and \( k \))?  
(b) What is the worst-case running time of Algorithm 2 (asymptotically, in terms of \( n \) and \( k \))?  
(c) For a given value of \( n \), for what values of \( k \) is Algorithm 2 at least as efficient as Algorithm 1? (Express your answer using asymptotic notation, in terms of \( n \)).

4. Define a sorting algorithm to be *parsimonious* if it never compares the same pair of input values twice. (Assume that all the values being sorted are distinct.) For example, it was shown in the notes that quicksort is parsimonious.

(a) Is insertion sort parsimonious? Justify your answer with either a counterexample or a brief argument.  
(b) Is merge sort parsimonious? Justify your answer with either a counterexample or a brief argument.  
(c) Is heap sort parsimonious? Justify your answer with either a counterexample or a brief argument.

5. In the lectures and in the notes, we examined the exact worst-case cost of sorting five items using two different sorting algorithms: MergeSort, and an *ad hoc* optimal algorithm for sorting exactly five items. In this problem, you are being asked to give the exact cost of sorting five items using other approaches. In all parts of this problem, “cost” means the number of comparisons performed between two input items.

(a) What is the exact worst-case cost of sorting five items using insertion sort?  
(b) What is the exact worst-case cost of sorting five items using QuickSort?  
(c) What is the exact expected cost of sorting five items using QuickSort, assuming all 5! permutations are equally likely. (For this part, it is fine to use a calculator or write and run a computer program.)
6. In the lectures and the lecture notes, we presented Counting Sort. We noted that because the final pass processes the input array from right to left, counting sort is a stable sorting algorithm.

Some implementations of Counting Sort that you might find on the internet process the input array from left to right in the final pass, rather than right to left. In essence, rather than using the control line on the final for loop as given in the lecture notes, which is as follows

```
for i = n downto 1
```

they use the following control line on the final for loop

```
for i = 1 to n
```

Give an example to show that if CountingSort is modified in this way, it is no longer a stable sorting algorithm. Explain why your example shows this.