The binary search algorithm for searching a sorted array is well known:

```python
def bin_search(A, first, last, target):
    # returns index of target in A, if present
    # returns -1 if target is not present in A
    if first > last:
        return -1
    else:
        mid = (first + last) // 2
        if A[mid] == target:
            return mid
        elif A[mid] > target:
            return bin_search(A, first, mid - 1, target)
        else:
            return bin_search(A, mid + 1, last, target)
```

Binary search reduces the number of possible locations for the target value by (about) half each time, which makes it quite efficient. But we could eliminate more locations by looking at two values in the range A[first ... last]. If we look at the value 1/3 of the way from first to last, and the value 2/3 of the way from first to last, we can eliminate about 2/3 of the locations each time. We can call this algorithm trinary search:

```
| ....... | ....... | ....... |
```

To search for value X, compare X to this and this

These two comparisons tell us which third of the array contains X

Pseudo-code for the trinary search algorithm is on the next page:
Your assignment is to empirically evaluate the efficiency of these two search algorithms.
Preliminary Work:

Learn how to use clock functions in the programming language that you plan to use for this assignment. You will need to measure elapsed time in small amounts.

Experiment 1:

For this range of values of n: n = 1000, n = 2000, n = 4000, n = 8000, n = 16000 complete the following steps. (Note that for your particular language/hardware combination, you may need to let n be much larger than this to get useful results from your clock functions. It is also possible that you may not be able to complete the experiment for the larger data sets. If so, that’s ok – just do the largest cases that you can):

Step 1: Generate an array (or list, in Python) of n randomly selected integers. Sort the integers into ascending order. You should be able to write your own sorting function.

Step 2: Generate a set of 10*n search values, consisting of each value in the list repeated 10 times.

Step 3: Use binary search to search the array for the values in this set. Measure the total time required to conduct the binary searches.

Step 4. Use trinary search to search the array for the values in this set. Measure the total time required to conduct the trinary searches.

Experiment 2:

Repeat Experiment 1, but this time search only for values that are not present in the array. (One easy way to do this is to fill your array with even values, then search for odd values.) In this experiment you should randomly generate the set of 10*n search values.

Create tables or graphs for the results of the two algorithms within the two experiments.
Based on the results of your experiments, answer the following questions:

1. Binary search and trinary search both fall into the $O(\log n)$ complexity class. Do your experiments show growth in execution time that is consistent with this?

2. Compare the total time for the two search algorithms:
   - Do they ever differ by more than 10%, or are they always within 10% of each other?
   - Under what conditions (if any) is binary search at least 10% faster and under what conditions (if any) is trinary search at least 10% faster?

Logistics:

You may complete the programming part of this assignment in Python, Java, C or C++.

You must submit your source code, properly documented according to standards established in CISC-121 and CISC-124. You must also submit a PDF file containing your answers to the questions. Both files must contain your name and student number, and must contain the following statement: “I confirm that this submission is my own work and is consistent with the Queen’s regulations on Academic Integrity.”

You are required to work individually on this assignment. You may discuss the problem in general terms with others and brainstorm ideas, but you may not share code. This includes letting others read your code or your written conclusions.

The due date for this assignment is midnight, January 29, 2017. Submission will be through onQ.