These must be completed and shown to your lab TA either by the end of this lab, or at the start of your next lab. You may work in groups of up to two people.

1. Download `qsortCount.cpp` from the course web page.

   The remaining questions are about how to instrument Quicksort in order to evaluate its average case performance empirically and then to simplify the instrumented code until the recurrence for the average case performance is revealed.

2. Add the global integer variable `comps` to the program and add one line to `quicksort` to count the number of comparisons between array elements that `quicksort` performs.

   Run your program using array size 1000 for 100 repetitions. What is the average number of comparisons?

   Run your program with a few different choices of NN (array size). How does the number of comparisons change with NN?

   Note: Write code in `main` to do this experiment.

3. You don't really need to sort the array in order to calculate `comps` for an input array of size n.

   Write a new recursive function `qc` that takes a single parameter `n` and returns the number of comparisons `quicksort` would perform in sorting an array of size `n`, but doesn't sort anything. (You'll call `randint` to pick a random pivot location.)

4. Instead of calling `randint` in the function `qc`, if we explicitly calculate the average number of comparisons, we might get the following code:

```
float c(int n) {
  if (n <= 1) return 0;
  float sum = 0.0;
  for (int m=1; m <= n; m++)
    sum += n-1 + c(m-1) + c(n-m);
  return sum / n;
}
```

   Write the value, $C(n)$, returned by `c(n)` as a recurrence relation. (It will have a summation and should look similar to the program, but in mathematical notation.)

   Warning: You should not run this code for n much bigger than 20 unless you're very patient (and have a long life). The running time is $\Omega(3^n)$.

5. Answer the following questions in a few short sentences:

   What are the run times for quicksort in A) the average case, and B) the worst case? Why are you unlikely to observe the worst-case run time, in practice?

   Is our implementation of quicksort an in-place sort? Is it a stable sort? Why or why not?

6. Be sure to show your work to your TA before you leave, or at the start of the next lab, or you will not receive credit for the lab!

7. **(Optional)** Use dynamic programming (the bottom-up-tabulate-solutions method of removing duplicate recursive calls) to remove the recursive calls from `c(n)`. How simple can you make the code?

   You could implement this either recursively, storing the results of recursive calls so that each call is never made twice (memoization), or you could do it iteratively, with a nested for-loop. When you complete the iterative version, you can try going even further by removing the inner for-loop.

8. **(Optional)** Solve the recurrence for $C(n)$ to show that $C(n) \in \Theta(n \log n)$. It will help to know that $\sum_{i=1}^{n} 1/i$ is $\Theta(\log n)$.