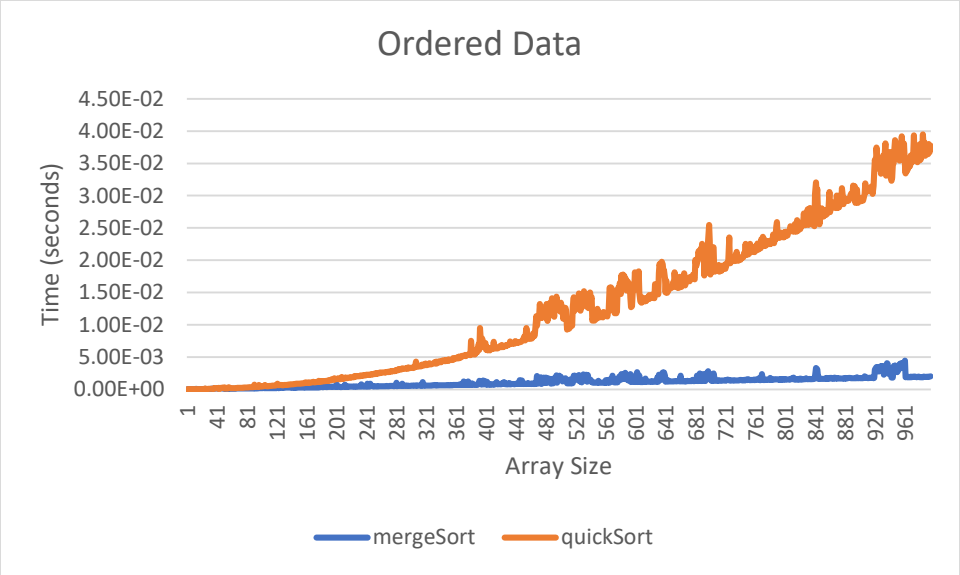
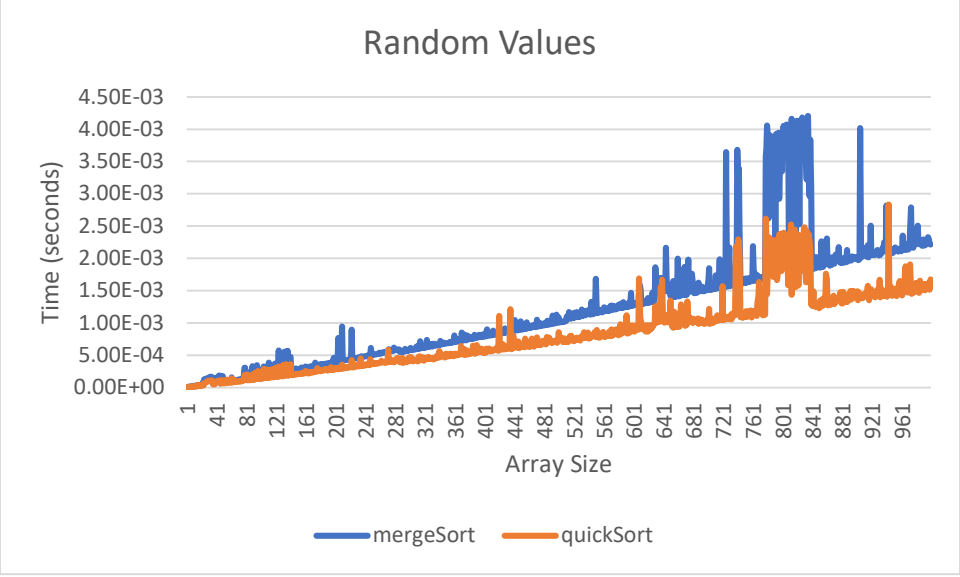
Jonathan Koning

Koningj@oregonstate.edu

CS 514

HW2





For sorted data, O(n^2) best fits quickSort and O(nlogn) best fits MergeSort. This means that for cases where the data is most likely sorted, MergeSort should be used.

For random data, O(nlogn) is still the best fit for MergeSort, but quicksort is also O(nlogn). Although both algorithms seem to be bounded by nlogn, quicksort seems to be slightly faster.

These two scenarios show that when handling data that is most likely sorted or is mostly sorted, MergeSort should be used. However, in cases where the data is guaranteed to be random, quicksort should be used. Unfortunately, in the real world, datasets can reach tremendous size and can be in any order. With this in mind, I would rely on MergeSort to provide a more consistent performance than that of quicksort even if that means leaving some performance on the table in some scenarios.

```
def bubbleSort(a):
    swap,j = 1, len(a)
   while swap == 1:

#  Assertion 1: swap must be 1, j > 0, a is an array of numbers

        swap,j = 0,j-1
        for i in range(j):

#  Assertion 2: i+1 <= len(a)-1, a[j] <= ... <= a[n-1]
            if a[i] > a[i+1]:
                a[i], a[i+1], swap = a[i+1], a[i], 1

# Assertion 3: [after the while-loop ends]. a[0] <= a[1] <= ... <= a[n-1]
```

Assertion 1 is held because the for loop will bring the next largest number to the next last spot in the array. Therefore, the array will be sorted before j gets below 1. The array is also simply manipulated but is never replaced so “a” will always be an array with the same numbers.

Assertion 2 is held because j starts at the last element of a, which is n-1, and is then reduced by 1 every iteration of the while loop. The for loop starts at the beginning of the array and checks each value against the next. If the current value is larger than the next, it is swapped with that value and remains the value of interest. In this way the largest value will always be brought to the end of the list. Because we know the end of the list will have the largest values, we no longer must check that value. Therefore j can be reduced and a[j] <= … a[n-1].

Assertion 3 is held because with every iteration of the while loop, the next largest number was brought to the j-1 position. Therefore, the array will be sorted.

T(n) = O(n^2)

The while loop iterates up to n times. The for loop iterates 1 less each time. Therefore it is the sum of all numbers n*(n +1)/2 = (n^2+n)/2 = O(n^2)