Algorithm for creating the worst case median-of-three quicksort.

In order to find the worst case for the median-of-three quicksort, the following assumptions were made:

1. The leftmost point would be the pivot in the median-of-three every time.
2. The middle point would be the only number in the list that was less than the pivot every time.
3. The rightmost point would be the greatest number in the list.

Using these parameters we took a list and "sorted" it using the previous assumptions.

a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24
a12	a0	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a1	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24
		a13	a2	a4	a5	a6	a7	a8	a9	a10	a11	a1	a3	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24
				a14	a4	a6	a7	a8	a9	a10	a11	a1	a3	a5	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24
						a15	a6	a8	a9	a10	a11	a1	a3	a5	a7	a16	a17	a18	a19	a20	a21	a22	a23	a24
								a16	a8	a10	a11	a1	a3	a5	a7	a9	a17	a18	a19	a20	a21	a22	a23	a24
										a17	a10	a1	a3	a5	a7	a9	a11	a18	a19	a20	a21	a22	a23	a24
												a18	a1	a5	a7	a9	a11	a3	a19	a20	a21	a22	a23	a24
														a19	a5	a9	a11	a3	a7	a20	a21	a22	a23	a24
																a20	a9	a3	a7	a11	a21	a22	a23	a24
																		a21	a3	a11	a7	a22	a23	a24
																				a22	a11	a7	a23	a24
																						a23	a7	a24

In the example about of 25 elements, the green cells are the sorted segments, and the yellow cells are the pivots for that step.

Then each was labeled from 0 to 24 across to get the order that the numbers were in originally. It was found that the original order was:

a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24
1	13	3	19	5	15	7	23	9	17	11	21	0	2	4	6	8	10	12	14	16	18	20	22	24

Several patterns were found in the worst case.

1. From the middle index (n/2) to the second to last index (n-2), it counts from 0 by 2 (0, 2, 4, … , n-2).
2. The last index (n-1) is always the largest number in the list.
3. The first half of the list has the following pattern:
   For each iteration i, find the first open space and fill it with the next available odd number while jumping 2^{i+1} spaces until the middle of the list is reached.
   The first open space for each iteration is given by the equation $2^i - 1$, for $i \geq 0, while\ 2^i - 1 < \frac{n-1}{2}$. The number of spaces that is jumped is given by the equations 2^{i+1} for the same $i \geq 0$.
   Below is the code which implements this algorithm.

From these patterns the algorithm for creating a list of worst case for the quicksort was made.

Code snippet for the worst case for the front of the list.

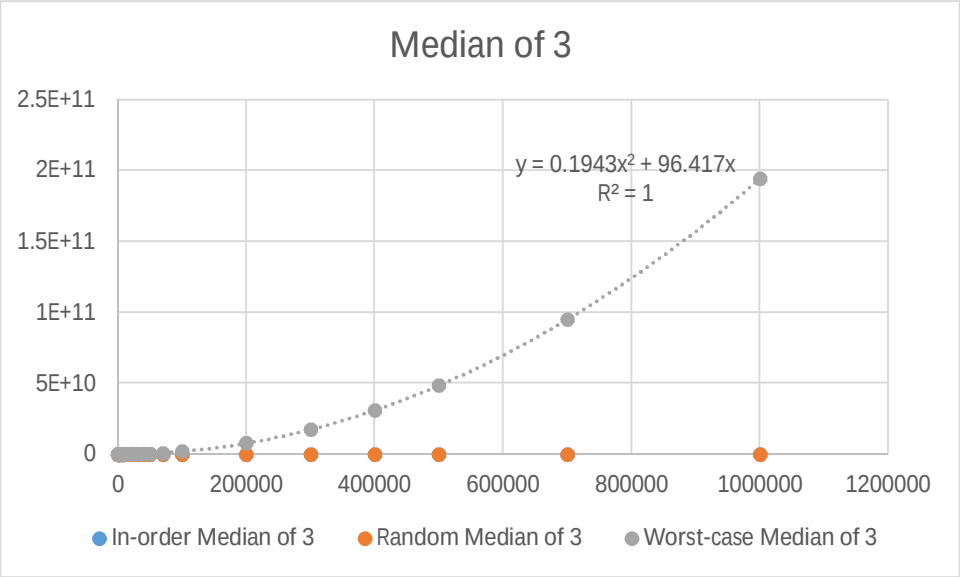
```
int l = 0, start, step;
for (int p = 1; p < n; )
{
    start = pow (2, l) - 1;
    step = pow(2, l+1);

    if( start > (n-1)/2)
      break;

    while (start < (n-1)/2)
    {
        scrambled[start] = sorted[p];
        p = p + 2;
        start = start + step;
    }
    l++;
  }
```

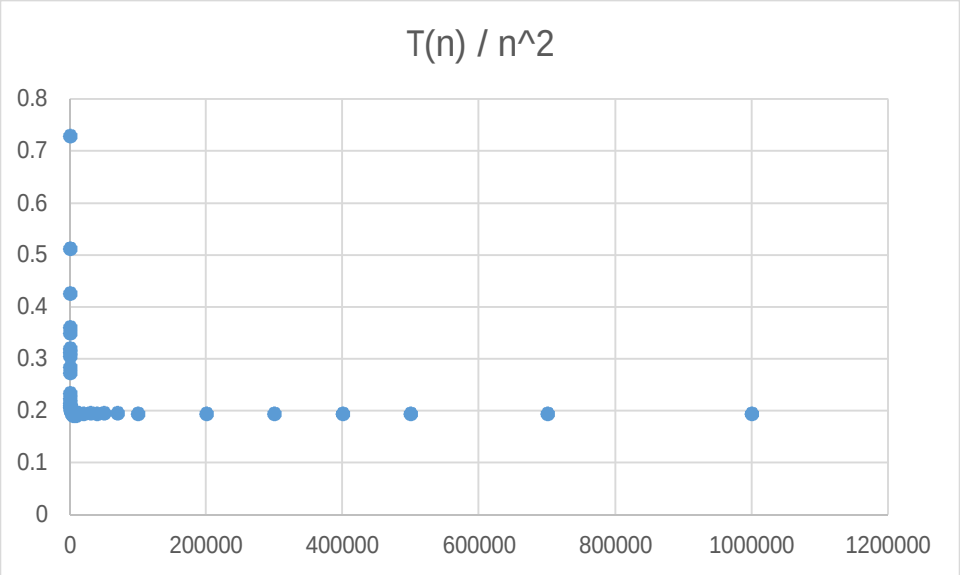
To determine the BigOh time two methods were used. The first was to plot points for different values of n fit a line to it. The second was to take that approximation of the plot function and plot $\frac{number\ of\ comparisons}{n^2}$ as discussed in class earlier this semester.

In order to get a good set of date the number of comparisons was collected for many cases ranging from 10 to 1,000,000. Then this data was plotted.



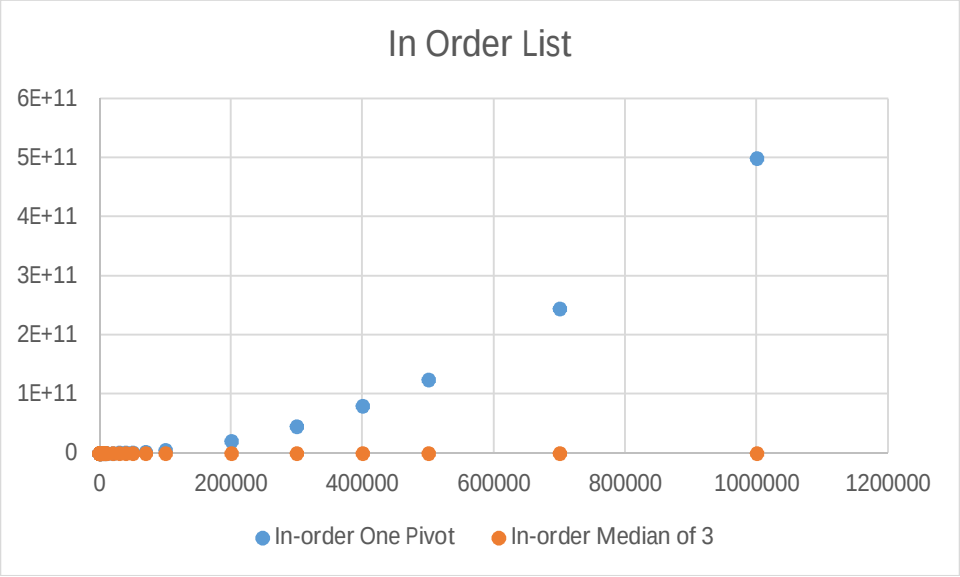
Above is the graph for the median–of-three quicksort texted on three different lists. 1. an in-order list, 2. a random list, and 3. the worst-case median of three. As you can see an in-order list does not show up in comparison to the worst case median-of-three, and the random list appears to be at zero. A trendline was added to the worst case for the median of three to get an approximation for the BigOh time. From this our estimate of worst case time was n^2.

From this it was possible to plot $\frac{number\ of\ comparisions}{n^2}$ on n. This gave the following graph.



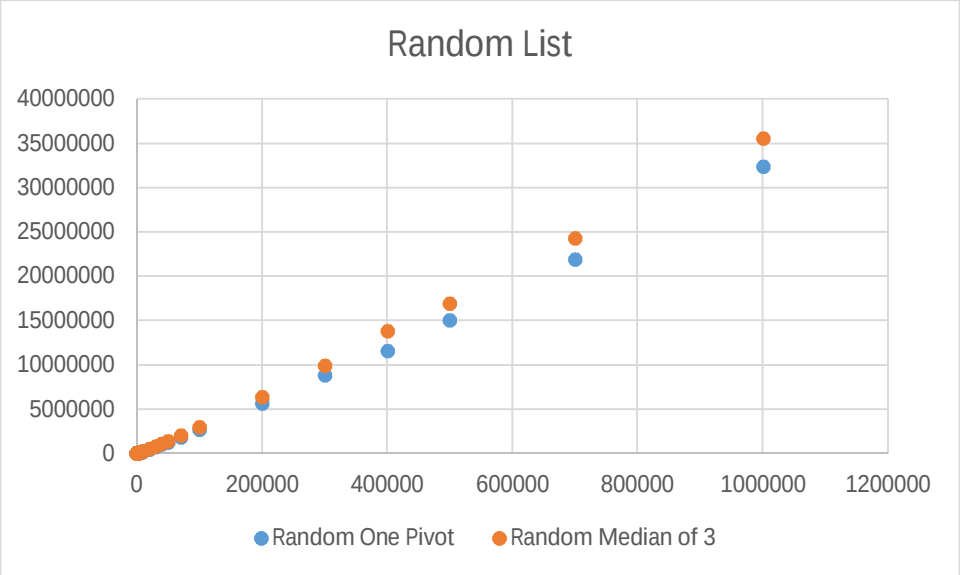
Since for large cases of n the graph flattens out, and doesn't trend upwards or downwards, it is a safe assessment to believe that this is BigOh n^2.

As extra assessment, the one-pivot quicksort was compared to the median-of-three quicksort. On an in-order list, the median-of-three was much quicker than the one-pivot quicksort as expected.

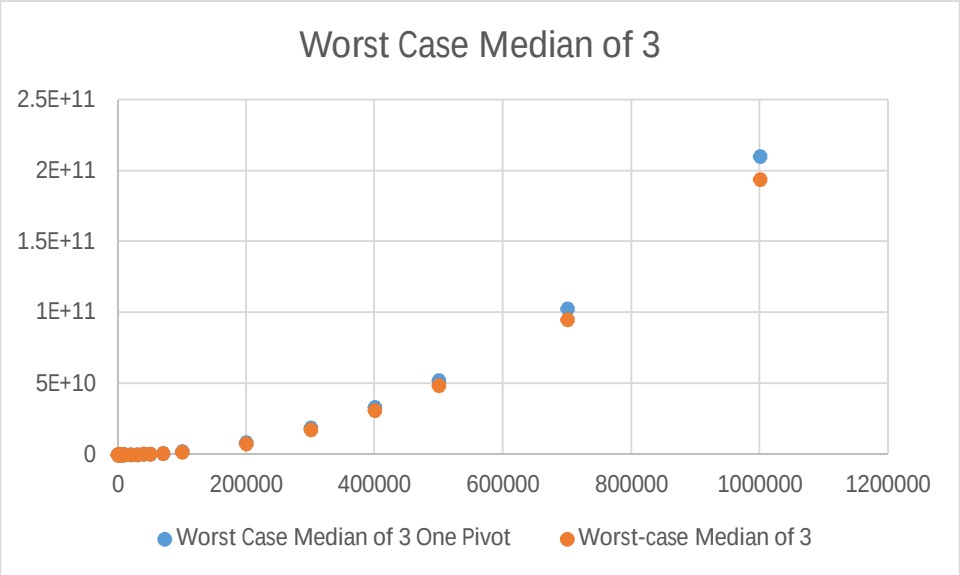


On a random list, something interesting was found. The median of three quicksort was slightly slower than the one pivot quicksort, but only slightly. Although the graph appear linear in nature, when the

same analysis is done as in the worst case, it is found that it is not quite linear as expected from Data Structures.



In the worst case for the median of three list, it was found that the one-pivot quicksort is slightly slower than the worst-case median of three. Looking further into this, the worst case median of three has a list that is partially in order, and so the one-pivot quicksort will not do well on this. Also the median-of-three quicksort is always guaranteed to move one piece past the pivot meaning it loses a size of two every time, where the one-pivot quicksort does not.



Many of these comparisons were made on the timings with similar results.

In conclusion, the worst case median-of-three algorithm found is BigOh n^2. Also it is found that on a random list the one-pivot quicksort is better than the median-of-three quicksort slightly. Although, if there is a risk of having an in-order list, it is wise to use the median-of-three quicksort.