Morgan Adams
Couse: Foundations of Algorithms
EN.605.621.85.FA21
Date: Nov 1, 2021

Project 2 Analysis

Overview
This project has us explore the median-of-three and normal partitioning strategies and then comparing their resulting worst case and average case runtimes with Quicksort.

Median-of-Three Partitioning:
The median-of-three partition takes the median of a[i], a[k], and a[j] of an array a[i] to a[j] where k = floor((i + j) / 2) and then uses this median as the pivot in Quicksort. Consider the pseudocode:

```
MedianOfThreePivot(array, start, end)
    mid = Floor((start + end) / 2)
    order = Array(start, mid, end)

    // insertion sort over start, mid, and end based on values
    //      in array
    for index = 1 to order.length
        insertVal = order[index]
        position = index
        while position > 0 && array[order[position - 1]] > array[insertVal ]
            order[position] = order[position - 1]
            position--
        order[position] = insertVal

    // order[1] is the index of array that points to the median of
    // array[start], array[mid], and array[end]
    return order[1]

Swap(array, index1, index2)
    temp = array[index1]
    array[index1] = array[index2]
    array[index2] = temp

MedianOfThreePartition(array, start, end)
    pivot = MedianOfThreePivot(array, start, end)
```

```
// move pivot to end
Swap(array, pivot, end)
pivot = end

lower = start - 1
higher = start
while higher < end
    if array[higher] <= array[pivot]
        lower++
        Swap(array, lower, higher)
    higher++
Swap(array, lower + 1, end)
return lower + 1
```

Runtime of Median-of-Three Partitioning Quicksort:
The recurrence relation doesn't change form with the median-of-three partition strategy. If we let a be the size of one partition, our recurrence looks like:

$T(n) = T(n-a) + T(n - (n-a)) + \Theta(n)$

or

$T(n) = T(n-a) + T(n - (n-a)) + cn.$

Where the median of 3 partitioning fits into this recurrence is in the constant for the cn term. Insertion sort is used to sort the indices of start, mid, and end by the values located in the array: array[start], array[mid], and array[end]. We then take whichever index in the array [start, mid, end] has the median value in the array and then use that as the pivot. If we look at the runtime of calculating the median-of-three and tolerate some sloppiness, the runtime is n^2 or simply 9 for every median calculation. We let d be equal to 9 and rewrite the recurrence factoring in d:

$T(n) = T(n-a) + T(n - (n-a)) + cn + d.$

Since only a constant value is being added at each level the recurrence, it doesn't change the recurrence form and neither does our average runtime change asymptotically since we know from 7.2 in our text (page 175-176) that Quicksort runs in O(n lg n) which is a larger order term than a constant.

In general, on random input we should expect median-of-three and normal Quicksort to perform the same asymptotically although median-of-three may be slightly slower by a constant factor.

Runtime of Quicksort Using Median-of-Three Partitioning on Sorted Input:
Where the median-of-three does help us is with the worst case where the recurrence is split into T(n-1) subproblems on each recursive call. This can happen if the array is input already in order. For example if the numbers 1 through 10 were in order as the input, then 10 would be our pivot on the first run. Then 9

would be the pivot. Then 8 and so on all the way down to 1. However with the median-of-three, 5 would become the pivot which gives us a more even split which is closer to the best case running time.

Looking at the actual running time of this scenario we see the normal Quicksort operate with the following recurrence:
$T(n) = T(n-1) + T(0) + \Theta(n)$

for every recursive call. Each recursive call has 1 less value meaning we get n levels with a cost of n per each giving us a runtime of $\Theta(n^2)$. More precisely we get n levels with a cost that drops by 1 on each level with gives us a summation per Equation A.1 from our text of $n(n+1)/2$ which still gives us our asymptotic runtime of $\Theta(n^2)$.

As described, when using median-of-three partitioning we better maintain the O(n lg n) runtime. The reason for this is that, once median-of-three finds 5 is the pivot value for an in order array, the resulting recurrence is our ideal scenario:

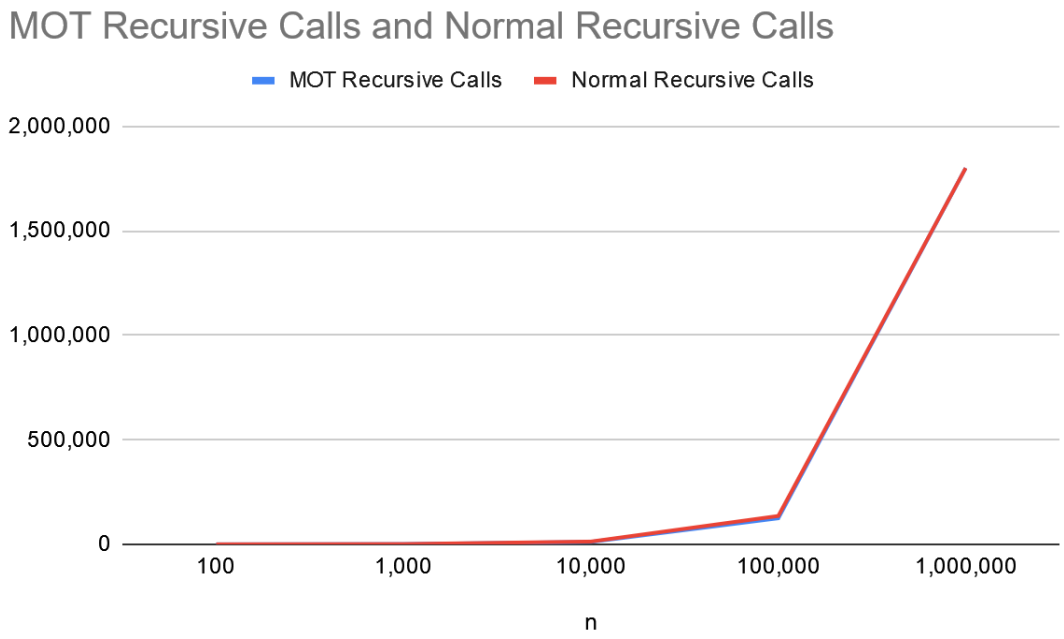
$T(n) = 2T(n/2) + \Theta(n)$

This will result in lg n levels and keep our runtime short.

In general the median-of-three helps us ensure we have better partitioning at each level of the recurrence by calculating the median from a small sample of the values.

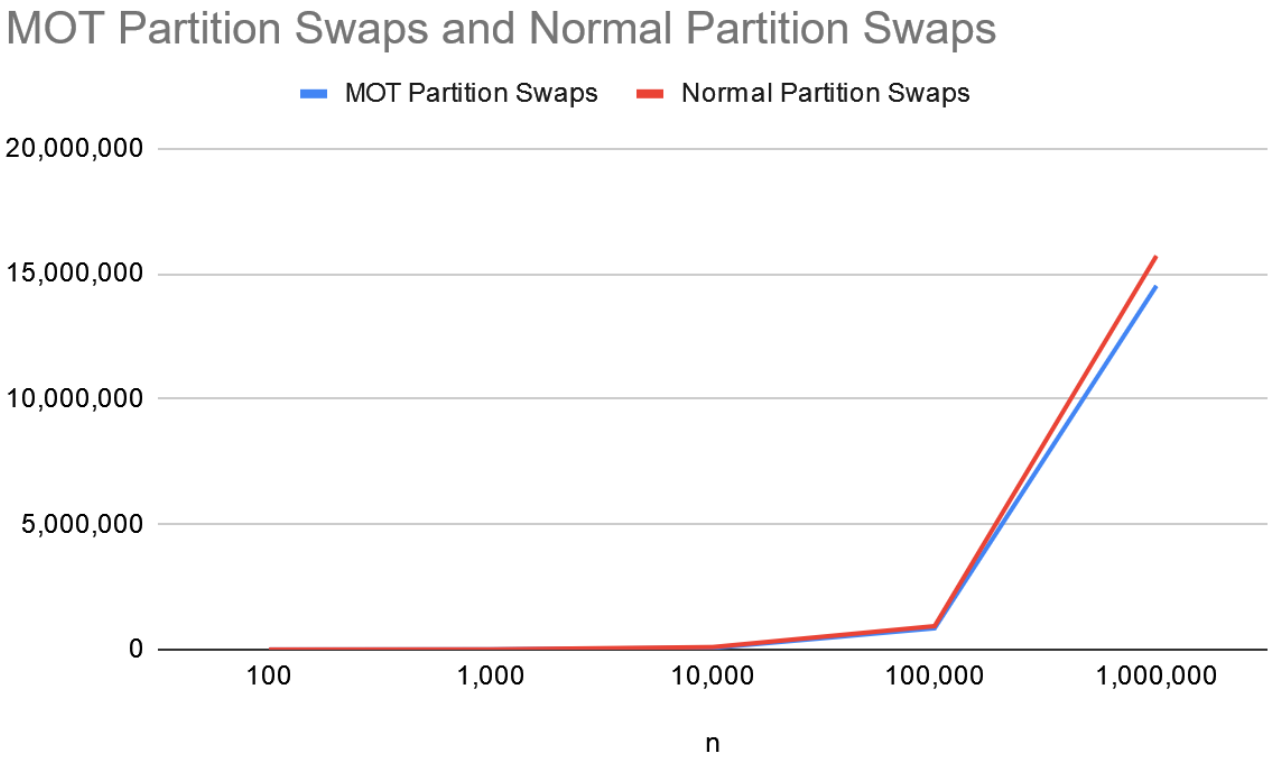
Measured Performance Analysis Between Median-Of-Three And Normal Quicksort:

Data sets were randomly generated with n=100, n=1000, n=10,000, n=100,000, and n=1,000,000. Two performance metrics were used to measure performance. The number of recursive calls made and the number of partition swaps that were made. We first look at the number of recursive calls over the randomized data sets.



n	MOT Recursive Calls	Normal Recursive Calls
100	115	139
1,000	1,133	1,327
10,000	11,585	13,281
100,000	124,677	135,657
1,000,000	1,800,095	1,800,147

As we might expect the median-of-three performs on par with normal quicksort. Intuitively this makes sense since with a randomized data, the selected pivots will likely be reasonable choices in either strategy. The number of recursive calls is effectively equivalent.

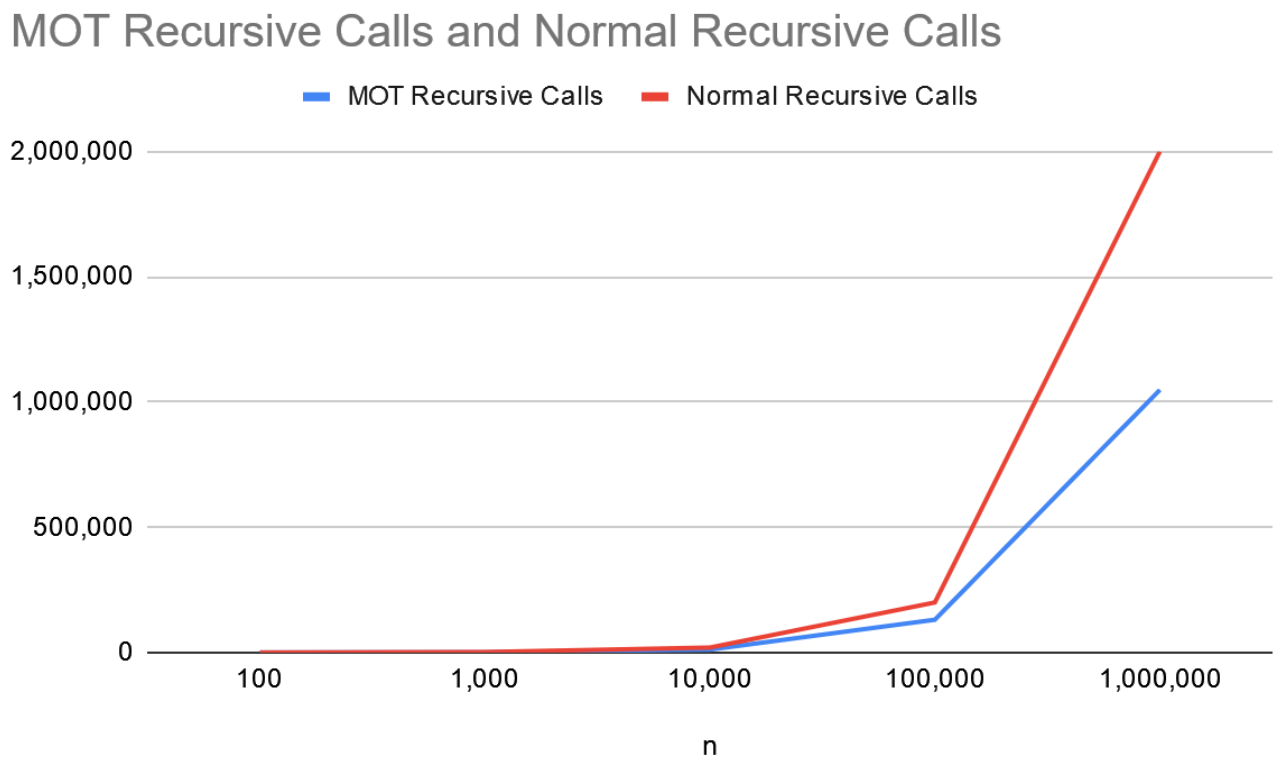


n	MOT Partition Swaps	Normal Partition Swaps
100	258	263
1,000	4,981	6,164
10,000	65,957	99,581
100,000	850,294	936,183
1,000,000	14,522,174	15,715,292

When looking at the number of swaps the median-of-three approach consistently performs better, but not asymptotically better - they're still asymptotically equivalent. The slight improvement is because the

median-of-three has a better chance of making a more reasonable median choice by drawing from a sample of 3 numbers instead of just 1.

The results were drastically different when considering in order data sets. One point of order is that for normal partitioning the results had to be extrapolated because it kept exceeding call stack limitations. The numbers are easily predictable based on the results so this was very feasible. With that in mind, we first consider the number of recursive calls made.

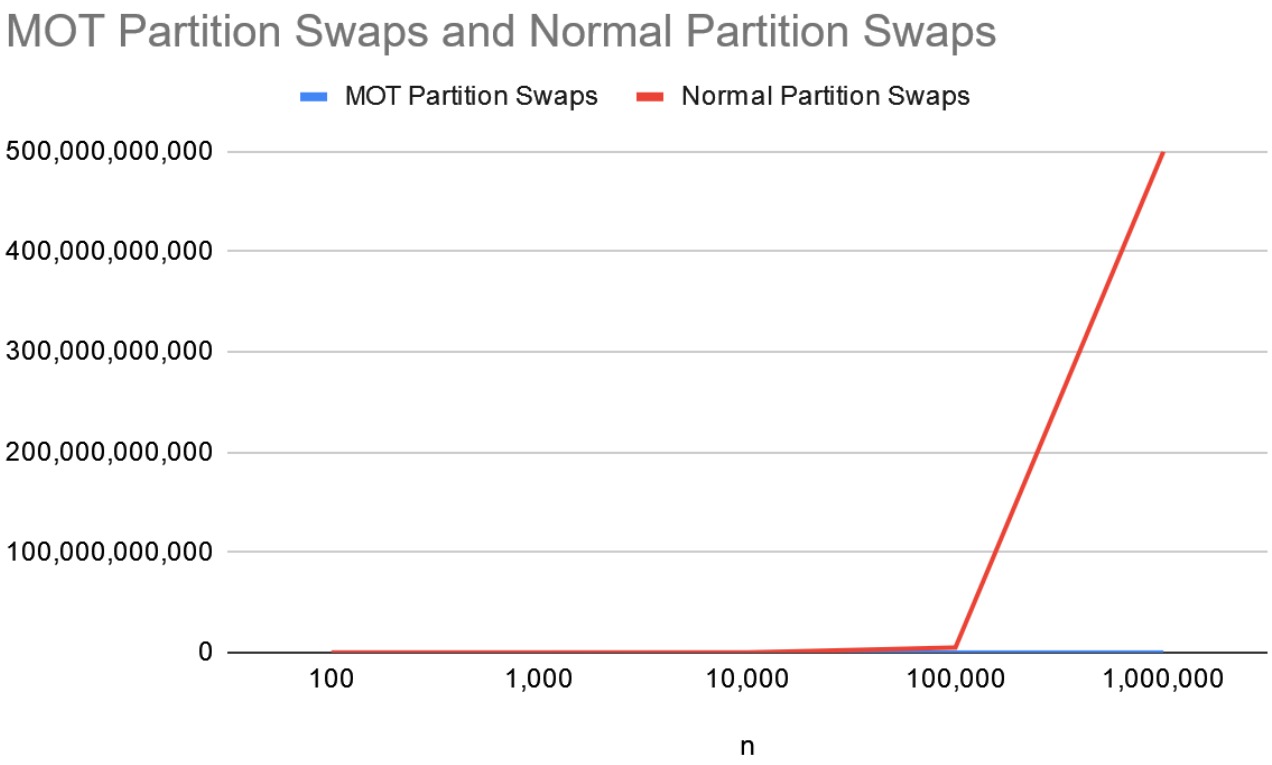


n	MOT Recursive Calls	Normal Recursive Calls
100	127	199
1,000	1,023	1,999
10,000	11,809	19,999
100,000	131,071	(extrapolated) 199,999
1,000,000	1,048,575	(extrapolated) 1,999,999

The number of recursive calls effectively doubles for in order data sets. This makes sense when we refer back to the derived recurrences for in order data sets:

Median-of-Three: $T(n) = 2T(n/2) + \Theta(n)$
Normal Partition: $T(n) = T(n-1) + T(0) + \Theta(n)$

In median-of-three we split the array in half for each subsequent subproblem whereas the normal partitioning reduces in size by n-1 for each subsequent problem and a T(0) call gets made n times so with that context, the almost doubling of recursive calls for normal partitioning makes sense, but the recursive calls in and of themselves don't asymptotically impact the runtime - that becomes more clear when we consider the number of swaps.



n	MOT Partition Swaps	Normal Partition Swaps
100	219	4,950
1,000	3,938	499,500
10,000	54,613	49,995,000
100,000	715,030	(extrapolated) 4,999,500,000
1,000,000	8,884,999	(extrapolated) 499,995,000,000

The results here are stark in comparison to executing against a randomized data set. The number of swaps that occur within a normal data set grow quadratically. In fact if we use the summation formula as suggested earlier for n=1,000,000 we get 500,000,500,000 which is only higher than the actual number of swaps by a relatively small constant and thus confirming that normal partitioning is a worst case for quicksort and runs at $\Theta(n^2)$.

Again we can intuitively grasp this by referring to the recurrence for normal partitioning on an ordered data set:

$T(n) = T(n-1) + T(0) + \Theta(n)$.

Since each recursive call is on T(n-1) and each subproblem only decreases by 1 we get a summation that's quadratic.

Looking at the n=1,000,000 for the median-of-three partition, we calculate the previously estimated O(n lg n) runtime to estimate 20 million. This is only off from the actual number of swaps by a factor of 2 which tells us that our estimate was accurate and median-of-three easily outperforms normal partitioning on ordered data.