Program #2
Student: Alice Xiong
Course: Foundations of Algorithms
Date: Nov03, 2015
Cellphone: 312-646-9280

1. (a) Pseudocode for median-of-three partitioning

#select out median among three values via the steps below, it takes three comparisons to sort it and get median value.

def medianThree(A, i1, i2, i3):

```
    findMedian = [i1, i2, i3]
    for i in range(0,2):
        for j in range(i+1,3):
            if A[findMedian[i]]>A[findMedian[j]]:
                count +=1
                temp = findMedian[i]
                findMedian[i] = findMedian[j]
                findMedian[j] = temp
    return findMedian[1] #sorted and pick up median
```

#Partition utilize the pivot element returned back by the **medianThree** function. The running time of mPartition on the subarray A[p .. r] is $\theta(n)$, where n=r-p+1.

def mPartition(A, p, r):

    #return back the pivot index
    i = **medianTree**(A, p, r, $\left\lfloor \frac{p+r}{2} \right\rfloor$)
    exchange A[r] with A[i]
below is the function called from textbook, which did partition based on the last element A[r]
    return **PARTITION**(A, p, r)

(b) [15 points] What is the running time of median-of-three partitioning? Justify your answer.

- The running time of **PARTITION** on the subarray A[p .. r] is $\boldsymbol{O(n)}$, where n=r-p+1.
- the running time of **medianThree** on the array with length n=3 has an expected running time $\theta(3)$

To sum up, the running time of median-of-three partitioning is **O(n).**

Additionally, we know that if the pivot point for the quick sort is selected via the method "median-of-three", it can guarantee that it won't occur the worst-case running time of $\theta(n^2)$.

(c) [20 points] What is the running time of QUICKSORT if you use median-of-three partitioning on an input set that is already sorted? Justify your answer.

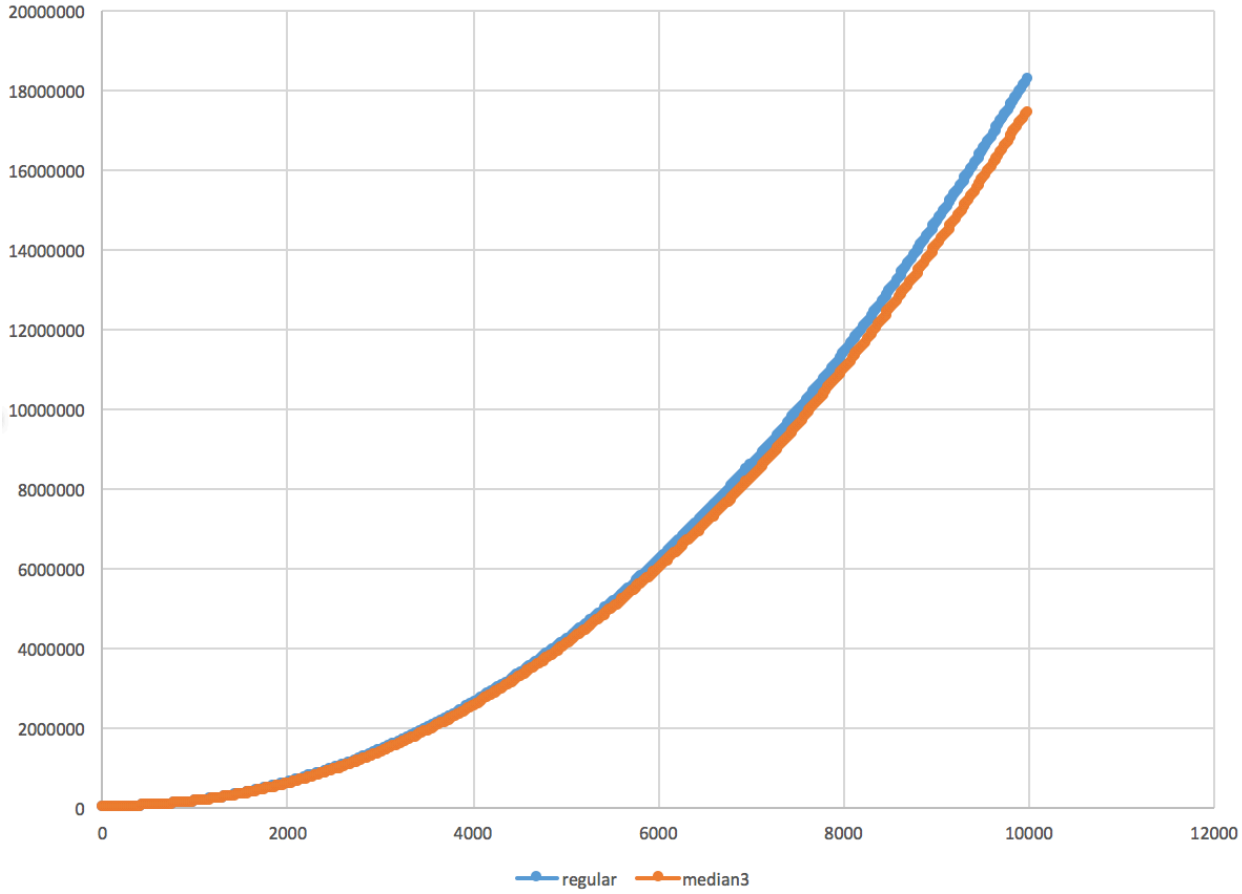
Assuming that the input is a sorted array but we don't know about it. Then the pivot value selected out each time will be the median to balance split the array. We can justify the running time via the master theory below:

T(n) = a T(n/b) + f(n) = 2T($\lfloor n/2 \rfloor$) + n if n > 1
T(n) = $\theta(1)$ if n = 1

- a = 2 sub-problems
- the sub-problems are solved recursively, each in time T(n/b) = T($\lfloor n/2 \rfloor$)
- the function f(n) = n encompasses the cost of dividing the problem and combining the results of the sub-problems.

According to the master theory of case2: if f(n)=n=O(n^{lg2^2})=n, then $T(n) = \theta(nlgn)$.

2. (a), (b), (c): see attached python code.
(d) We simulated n with size increased from 10 to 9990, and obtain the figure below. (You can see the details within the excel, the horizontal axis stands for n.)



It looks like median of three quick sort works slightly better than the regular quick sort. Probably by using the median of three partitioning, it split the array more evenly and then decrease the constant for the nlgn term.