Problem Set 2

Problem 2.1

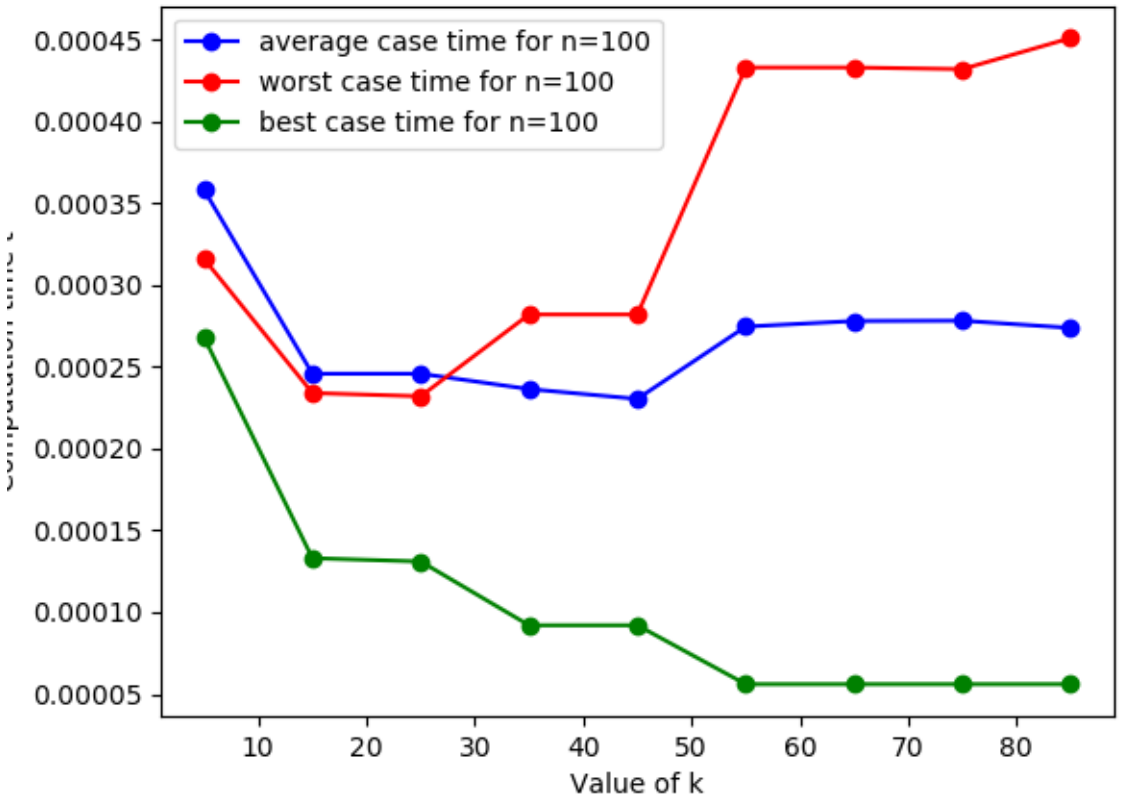
Solution:

(a) See the file merge_insertion.py for implementation

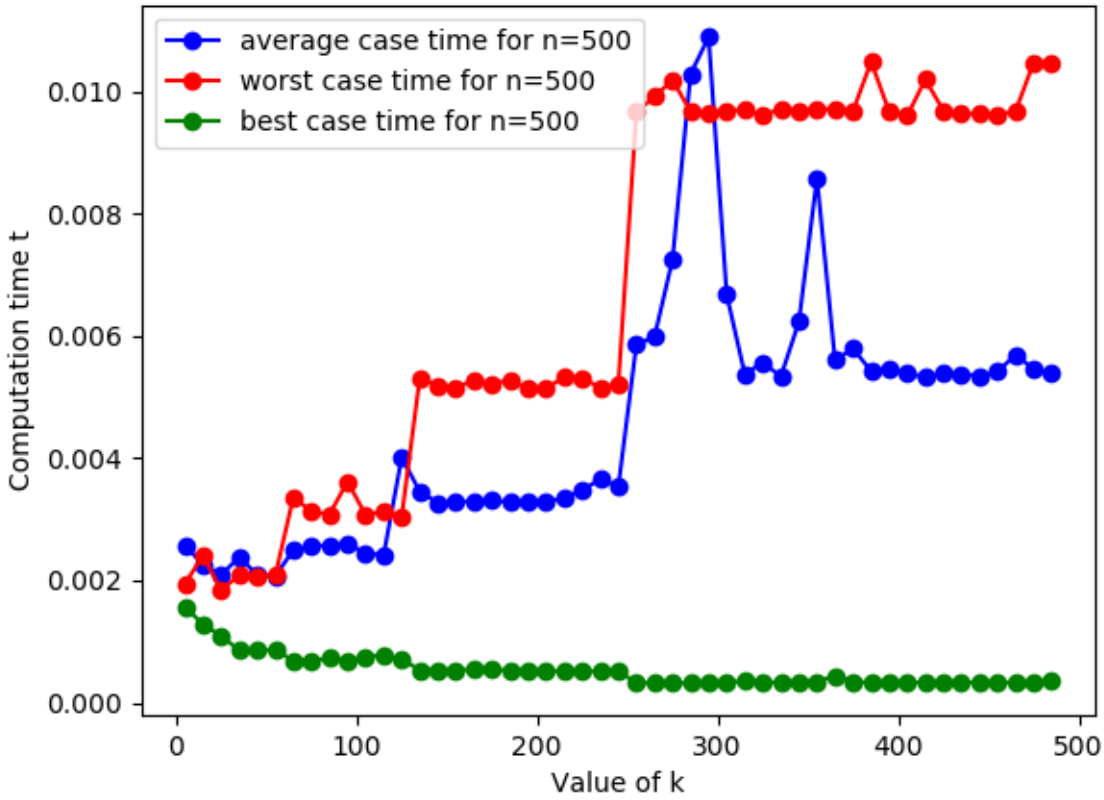
```
def mergeSort(self, A, k):
  t=Tools()
  size=len(A)
  if size<k:
    A=t.insertionSort(A)
    return A
  else:
    mid=size//2
    left=A[0:mid]
    right=A[mid:]
    #sorting left and right arrays recursively
    left=t.mergeSort(left,k)
    right=t.mergeSort(right,k)
    #merging the left and right sorted arrays into single array mergedA
    mergedA=t.merge(A,left,right)
    return mergedA
```

(b) See the file merge_insertion.py for implementation and see plots below

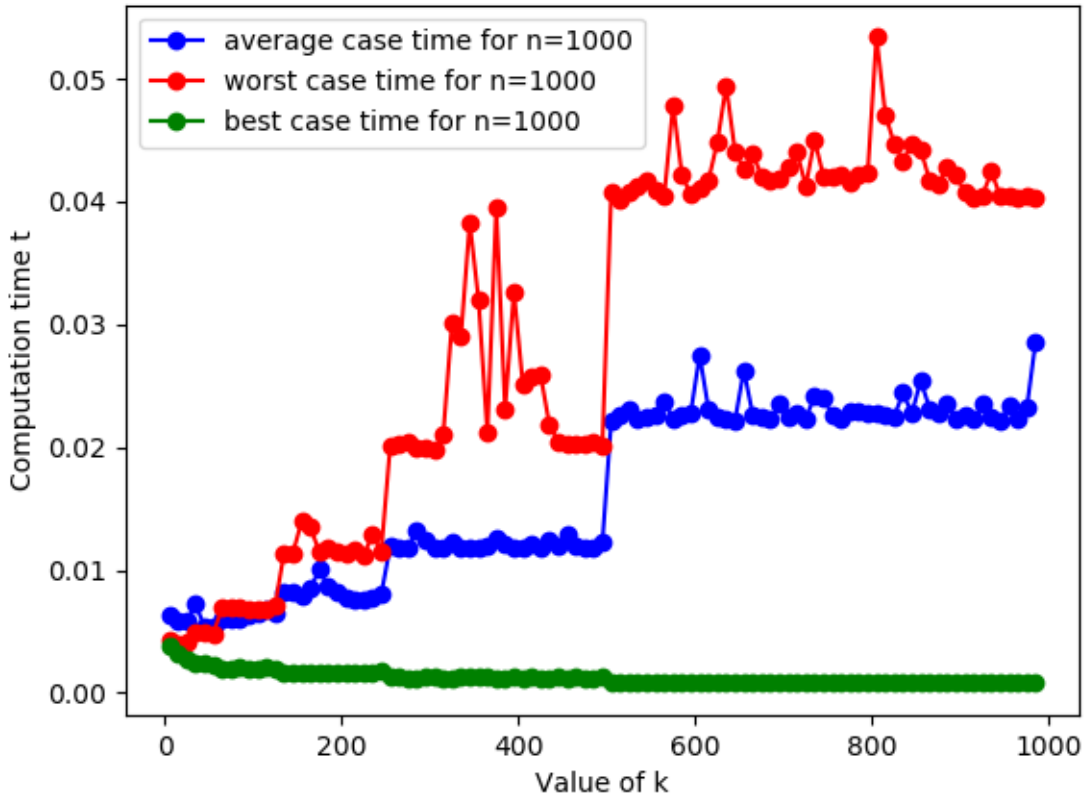
For n=100:



For n=500:



For n=1000:



(c) Insertion sort has worst case and average case running time of $O(n^2)$ while Merge Sort has worst and average case running time of $O(n * log(n))$. The effect of combining the two can be seen in the above graphs.
For large values of k, we observe that as the value of k increases, the computation time (worst and average case) increases because we are performing more insertion sort operations and less

merging operations, so the load of the insertion sorting is higher.
For very small values of k, we are performing more merge operations than insertion operations, but the merging turns out to be more expensive because merging has more constant operations than insertion sort. This is why we observe a dip in the running times after the first few values of k.
Trivially, the best case running time decreases with increasing values of k.

(d) In theory, one would minimize the expression $n * k + n * log(n/k)$ to find the optimal value of k. However, in practice, it would be better to conduct a series of experiments to determine the optimal value of k. This is because the asymptotic bounds do not account for all the constants. Generally, we would choose a small value for k.

Problem 2.2

Solution:
(a) Using master method, $a = 6, b = 36$. $n^(log_b a) = n^2$,$f(n) = n$
Case 1: $f(n) = O^{2-e}$ for $e = 1$. Therefore, $T(n) = \Theta(n^2)$

(b)Using master method, $a = 5, b = 3$. $n^(log_b a) = n^{1.46}$,$f(n) = 17n^{1.2}$
Case 1: $f(n) = O^{1.46-e}$ for $e = 0.26$. Therefore, $T(n) = \Theta(n^{log_3 5})$

(c)Using master method, $a = 12, b = 2$. $n^(log_b a) = n^{3.58}$,$f(n) = n^2 logn$
Therefore, $T(n) = \Theta(n^2(log_3 5)^2)$.

(d)$T(n) = \Theta(2^n)$

(e)$T(n) = \Omega(nlogn)$