CS 325 - Homework Assignment 2

Problem 1: *(3 points)* Give the asymptotic bounds for T(n) in each of the following recurrences. Make your bounds as tight as possible and justify your answers. *Assume the base cases T(0)=1 and/or T(1) = 1.*

a) $T(n) = T(n-2) + n$

T(n) = T(n-2) + n
T(n-1) = T(n-1-2) + (n-1)
…

T(n) = n + n-1 + ... + 1
T(n) = $O(n^2)$

b) $T(n) = 3T(n-1) + 1$

T(n) = 3T(n-1) + 1
T(n-1) = 3T(n-1-1) + 1
…

T(n) = $3^{n-1} + 3^{n-2} + \ldots + 3 + 1$
T(n) = $3^{n-1}/2$
T(n) = $\Theta(3^n)$

c) $T(n) = 2T\left(\frac{n}{8}\right) + 4n^2$

T(n) = 2T(n/8) + $4n^2$
T(n) = 2T($n/8^2$) + $4(n/8)^2$
…

T(n) = $2^k T(n/8^k) + 4n^2(1+1/8^2+1/8^3+\ldots)$
T(n) = $n + n^2 * C$
T(n) = $\Theta(n^2)$

Problem 2: *(6 points)* The quaternary search algorithm is a modification of the binary search algorithm that splits the input not into two sets of almost-equal sizes, but into four sets of sizes approximately one-fourth.

a) Verbally describe and write pseudo-code for the quaternary search algorithm.
The idea of this problem and binary search are similar. We have Left for start point and Right for end point. If it isn't match, we will move Left and Right to next point.

```
quaternary (Array, Left, Right)
  if Left >Right
      return False
  K = Left + (Right – Left) /4
  if A[k] > x
       quaternary (Array, Left, k-1, x)
  if A[k] < x
       quaternary (Array, k-1, Right, x)
 return K
```

b) Give the recurrence for the quaternary search algorithm

T(n) = T(n/4) + 1

c) Solve the recurrence to determine the asymptotic running time of the algorithm. How does the running time of the quaternary search algorithm compare to that of the binary search algorithm.

T(n) = T(n/4) + 1
a = 1, b = 2
$n^{\log_b a} = 0$
compare n^0 with f(n). $n_0 = \Theta(f(n))$
use case 2
$T(n) = \Theta(\log n)$

The running time is same as binary search.

Problem 3: *(6 points)* Design and analyze a **divide and conquer** algorithm that determines the minimum and maximum value in an unsorted list (array).

a) Verbally describe and write pseudo-code for the min_and_max algorithm.

The divide and conquer algorithm we develop for this problem is motivated by the following observation. Suppose we knew the maximum and minimum element in both of the roughly n/2 sized partitions of an n-element (n >= 2) list. Then in order to find the maximum and minimum element of the entire list we simply need to see which of the two maximum elements is the larger, and which of the two minimums is the smaller. We assume that in a 1-element list the sole element is both the maximum and the minimum element. With this in mind we present the following psuedocode for the max/min problem.

```
min_and_max (Array, max, min)
  if (n == 1)
    return (A[1], A[1])
  else if (n == 2)
    if( A[1] < A[2])
      return (A[1], A[2])
    else
      return (A[2], A[1])
  else
    (max_left, min_left) = min_and_max (A[1...(n/2)])
    (max_right, min_right) = min_and_max (A[(n/2 +1)...n])
    if (max_left < max_right)
      max = max_right
    else
      max = max_left
    if (min_left < min_right)
      max = min_left
    else
      min = min_right
    return (min, max)
```

b) Give the recurrence.
$T(n) = 2T(n/2) + 1$

c) Solve the recurrence to determine the asymptotic running time of the algorithm. How does the theoretical running time of the recursive min_and_max algorithm compare to that of an iterative algorithm for finding the minimum and maximum values of an array.
$T(n) = 2^kT(n/2^k) + k$
When k = logn stop
$T(n) = \Theta(logn)$
For inerative algorithm, the fast way is set two pointers on the start and end point. The Running time is $\Theta(n/2)$. When n is infinty, logn is smaller than n/2, so recursive is faster than iterative.

Problem 4: *(4 points)* Consider the following algorithm for sorting.

```
StoogeSort(A[0 ... n - 1])
        if n = 2 and A[0] > A[1]
                swap A[0] and A[1]
        else if n > 2
                m = ceiling(2n/3)
                StoogeSort(A[0 ... m - 1])
                StoogeSort(A[n - m ... n - 1])
                Stoogesort(A[0 ... m - 1])
```

a) Explain why the STOOGESORT algorithm sorts its input. (This is not a formal proof).

For this sort style, we can think that the inputs are divided in 3 equal or similar parts(part 1, 2, 3). Frist time, we choose part 1 and 2 to sort it. So the all elements in part 2 are bigger than all elements in part 1. Then we choose part 2 and 3, and sort it. So all elements in part 3 are bigger than part 2 and part 1 because before this sort, the part 2 already bigger than part 1, so if the original part 3 are small than part 2, we can swap all elements in part 2 to part 3. Finally, we choose part 1 and 2 to sort it. We can get a sorted list.

b) Would STOOGESORT still sort correctly if we replaced k = ceiling(2n/3) with k = floor(2n/3)? If yes prove if no give a counterexample. (Hint: what happens when n = 4?)
It cannot work. Because for this sort function, we want the elements in their cross section are more than half of the elements we choose to sort. For example, when n =4. The k = floor(2n/3) = 2. The first part will be A[0,1] and second part is A[2,3]. So they don't have cross section that we cannot compare the whole list. We just sort two separate list A1 in A[0,1] and A2 in A[2,3], not A[].

c) State a recurrence for the number of comparisons executed by STOOGESORT.
$T(n) = 2T(2n/3) + n$

d) Solve the recurrence to determine the asymptotic running time. (Hint: Ignore the ceilings)
$T(n) = 2T(2n/3) + n$

$a = 3$, $b = 2/3$, $\log_b a = \log_{2/3} 3 \approx 2.7$
compare $\log_b a$ with $f(n) = n$
use case 1,
$T(n) = \Theta(n^{\log 3/\log 2/3})$

CS 325 - Homework Assignment 2

Problem 5: *(6 points)*

a) Implement STOOGESORT from Problem 4 to sort an array/vector of integers. Implement the algorithm in the same language you used for the sorting algorithms in HW 1. Your program should be able to read inputs from a file called "data.txt" where the first value of each line is the number of integers that need to be sorted, followed by the integers (like in HW 1). The output will be written to a file called "stooge.out".

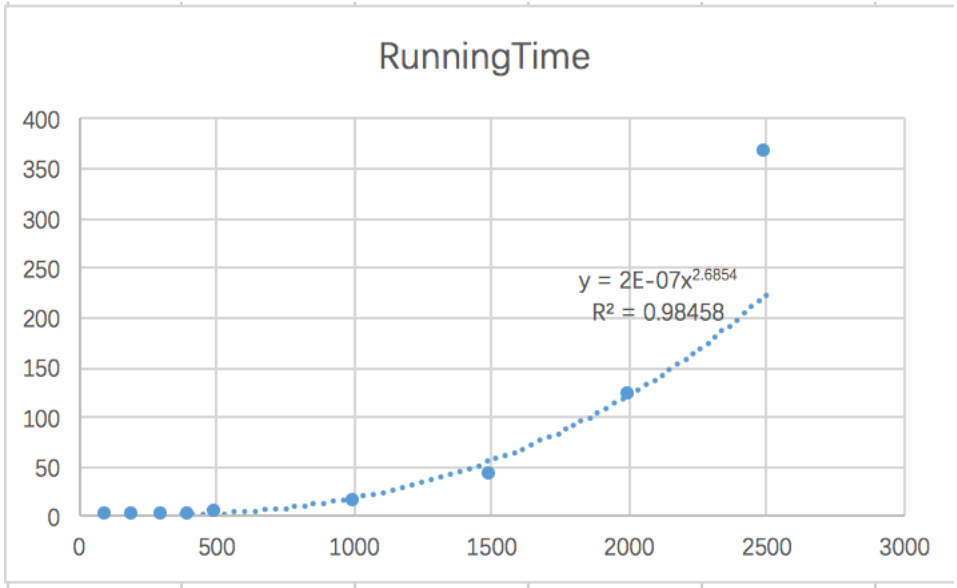
Submit a copy of all your code files and a README file that explains how to compile and run your code in a ZIP file to TEACH. We will only test execution with an input file named data.txt.

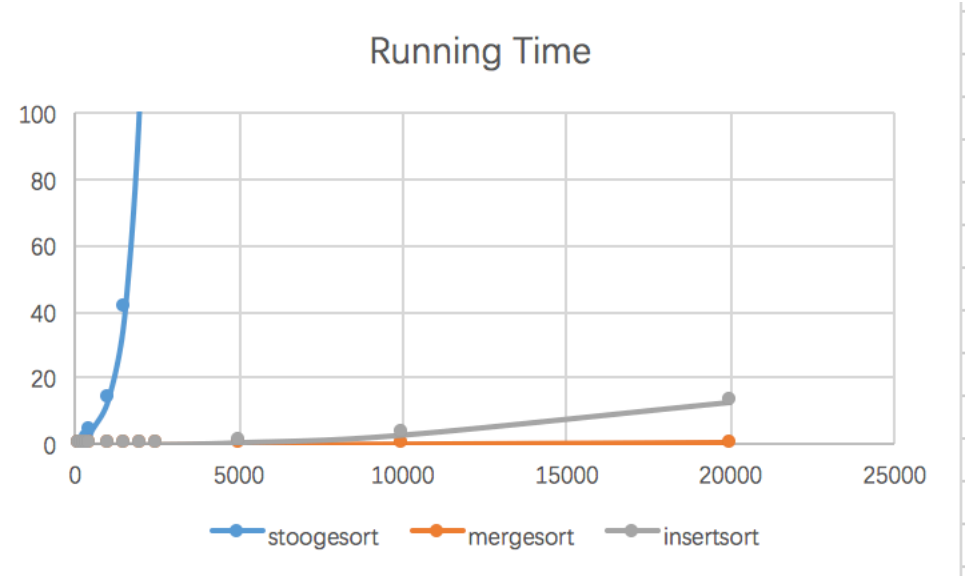
b) Now that you have proven that your code runs correctly using the data.txt input file, you can modify the code to collect running time data. Instead of reading arrays from a file to sort, you will now generate arrays of size n containing random integer values and then time how long it takes to sort the arrays. We will not be executing the code that generates the running time data so it does not have to be submitted to TEACH or even execute on flip. Include a "text" copy of the modified code in the written HW submitted in Canvas. You will need at least seven values of t (time) greater than 0. If there is variability in the times between runs of the algorithm you may want to take the average time of several runs for each value of n.

```
import datetime
import random

def stoogesort(L, i=0, j=None):
    if j is None:
        j = len(L) - 1
    if L[j] < L[i]:
        L[i], L[j] = L[j], L[i]
    if j - i > 1:
        t = (j - i + 1) // 3
        stoogesort(L, i, j - t)
        stoogesort(L, i + t, j)
        stoogesort(L, i, j - t)
    return L

def main():
    n = 400
    print n
    lst = [random.randint(0, 10000) for _ in range(n)]


    starttime = datetime.datetime.now()
    print stoogesort(lst)
    endtime = datetime.datetime.now()
    print (endtime - starttime)


if __name__ == "__main__":
    main()
```

	100	200	300	400	500	1000	1500	2000	2500
RunningTime	0.062819	0.174811	0.533722	1.570456	4.518089	14.049104	41.172818	122.337894	366.605666

c) Plot the running time data you collected on an individual graph with n on the x-axis and time on the y-axis. You may use Excel, Matlab, R or any other software. Also plot the data from Stooge algorithm together on a combined graph with your results for merge and insertion sort from HW1.





d) What type of curve best fits the StoogeSort data set? Give the equation of the curve that best "fits" the data and draw that curve on the graphs of created in part c).
For stoogesort, we can see in graph that is a curve of power function.

e) How does your experimental running time compare to the theoretical running time of the algorithm?
For the theoretical running time, the stoogesort should take $n^{2.7}$. My experimental running time is about $n^{2.68}$.