CS 325 Summer 17
HW 1 – 25 points

1) (1 pt) Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in 64nlgn steps. For what values of n does insertion sort run faster than merge sort?

Note: lg n is log "base 2" of n or $\log_2 n$. There is a review of logarithm definitions on page 56. For most calculators you would use the change of base theorem to numerically calculate lgn.

For this question, that let us to find n when $8n^2 <= 64n\log n$.

So,

$8n^2 <= 64n\log n$

$n^2 <= 8n\log n$

n <= 8logn

n – 8logn = 0

n = 43.411

So for 0 < n <= 43, insertion sort works better than merge sort.

2) (5 pts) For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and explain.

a.	$f(n) = n^{0.25}$;	$g(n) = n^{0.5}$
b.	f(n) = n;	$g(n) = \log^2 n$
c.	f(n) = log n;	g(n) = ln n
d.	$f(n) = 1000n^2$;	$g(n) = 0.0002n^2 – 1000n$
e.	f(n) = nlog n;	$g(n) = n\sqrt{n}$
f.	$f(n) = e^n$;	$g(n) = 3^n$
g.	$f(n) = 2^n$;	$g(n) = 2^{n+1}$
h.	$f(n) = 2^n$;	$g(n) = 2^{2^n}$
i.	$f(n) = 2^n$;	g(n) = n!
j.	f(n) = lgn;	$g(n) = \sqrt{n}$

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{n^{0.25}}{n^{0.5}}\right) = \frac{1}{n^{0.25}} = 0$$

f(n) is O(g(n))

CS 325 Summer 17
HW 1 – 25 points

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{n}{log^2 n}\right) = \lim_{n\to\infty}\left(\frac{n}{(\log n)^2}\right) = \infty$$

f(n) is Ω(g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{\log n}{\ln n}\right) = \frac{1}{\log 2} = C$$

f(n) is Θ(g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{10000n^2}{0.0002n^2 - 1000n}\right) = \lim_{n\to\infty}\left(\frac{10000n^2}{0.0002n^2}\right) = \frac{10000}{0.0002} = C$$

f(n) is Θ(g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{n\log n}{n\sqrt{n}}\right) = \lim_{n\to\infty}\left(\frac{\log n}{\sqrt{n}}\right) = \infty$$

f(n) is Ω(g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{e^n}{3^n}\right) = \lim_{n\to\infty}\left(\frac{e}{3}\right)^n = 0$$

f(n) is O(g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{2^n}{2^{n+1}}\right) = \frac{1}{2} = C$$

f(n) is Θ (g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{2^n}{2^{2n}}\right) = \frac{1}{2^n} = 0$$

f(n) is O(g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{2^n}{n!}\right) = \frac{2*2*2*\ldots..*2}{n*(n-1)*(n-2)*\ldots.*2*1} = 0$$

f(n) is O(g(n))

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{\lg n}{\sqrt{n}}\right) = 0$$

f(n) is O(g(n))

3) (4 pts) Let f_1 and f_2 be asymptotically positive non-decreasing functions. Prove or disprove each of the following conjectures. To disprove give a counter example.

CS 325 Summer 17
HW 1 – 25 points

a. If $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$ then $f_1(n) = \Theta(f_2(n))$.

Because we know $f_1(n) = o(g(n))$ and $f_2(n) = O(g(n))$. And the f_2 can be $f_2 = O(f_2(n))$ and $f_2 = O(g(n))$, and $f_2(n) = \Theta(f_2(n))$, so $O(g(n)) = \Theta(f_2(n))$. Therefore, $f_1 = O(g(n)) = \Theta(f_2(n))$. So $f_1(n) = \Theta(f_2(n))$.

b. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then $f_1(n)+f_2(n) = O(\max\{g_1(n), g_2(n)\})$

Let $f_1(n) = O(g_1(n))$ and $f_1(n) = O(g_1(n))$. This means that there exist constants $c_1, c_2 > 0$ such that $f_1(n) <= c_1g_1(n)$ and $f_2(n) <= c_2g_2(n)$ for all $n > 0$ integers. To prove the claim, we must find some constant c_3 that causes $f_1(n) + f_2(n) <= c_3(g_1(n) + g_2(n))$ for all $n > 0$ integers.
So,

$$\begin{aligned} f_1(n) + f_2(n) &<= c_1g_1(n) + c_2g_2(n) \\ &<= \max(c_1, c_2)g_1(n) + \max(c_1, c_2)g_2(n) \\ &<= \max(c_1, c_2)(g_1(n) + g_2(n)) \\ &= c_3(g_1(n) + g_2(n)) \end{aligned}$$

so $f_1(n)+ f_2(n) = O(\max\{g_1(n), g_2(n)\})$

4) (5 pts) **Merge Sort and Insertion Sort Programs**
Implement merge sort and insertion sort to sort an array/vector of integers. These are very common algorithms and you may modify existing code if you reference it. You may implement the algorithms in the language of your choice, name one program "mergesort" and the other "insertsort". Your programs 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.
Example values for data.txt:

```
4 19 2 5 11
8 1 2 3 4 5 6 1 2
```

The output will be written to files called "merge.out" and "insert.out".
For the above example the output would be:

```
2 5 11 19
1 1 2 2 3 4 5 6
```

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.

5) (10 pts) **Merge Sort vs Insertion Sort Running time analysis**

The goal of this problem is to compare the experimental running times of the two sorting algorithms.

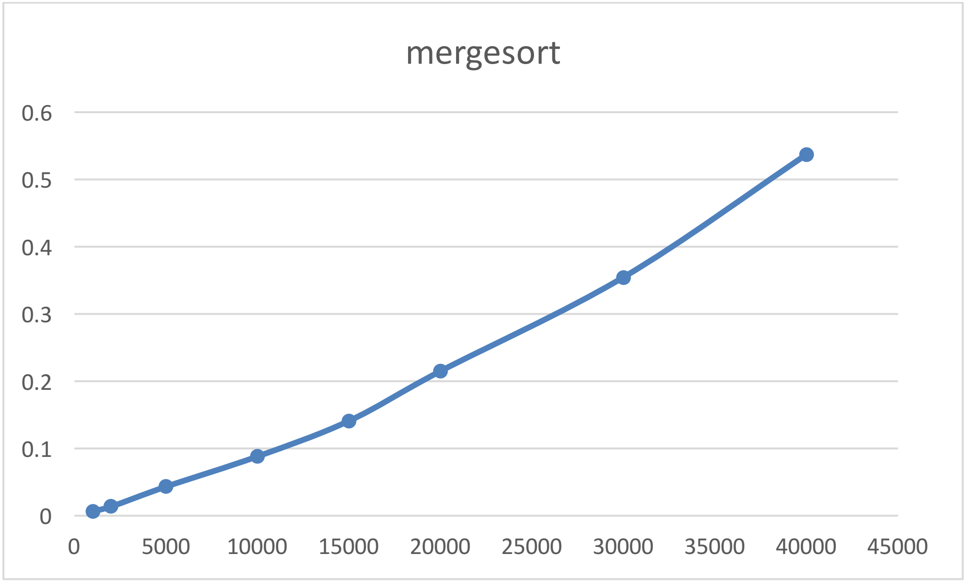
a) Code

```
#!/usr/bin/env python
import sys
import datetime
import random

def mergesort(lst):
    if (len(lst) <= 1): return lst
    left = mergesort(lst[:len(lst) / 2])
    right = mergesort(lst[len(lst) / 2:len(lst)])
    result = []
    while len(left) > 0 and len(right) > 0:
        if (left[0] > right[0]):
            result.append(right.pop(0))
        else:
            result.append(left.pop(0))

    if (len(left) > 0):
        result.extend(mergesort(left))
    else:
        result.extend(mergesort(right))
    return result


# insert_sort
def insertsort(l):
    for i in range(len(l)):
        min_index = i
        for j in range(i + 1, len(l)):
            if l[min_index] > l[j]:
                min_index = j
        tmp = l[i]
        l[i] = l[min_index]
        l[min_index] = tmp
    return str(l)


def main():
    n = 40000
    print n
    list = [random.randint(0, 10000) for _ in range(n)]

    starttime = datetime.datetime.now()
    print mergesort(list)
    endtime = datetime.datetime.now()
    print (endtime - starttime)


    starttime = datetime.datetime.now()
    print insertsort(list)
    endtime = datetime.datetime.now()
    print (endtime - starttime)

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

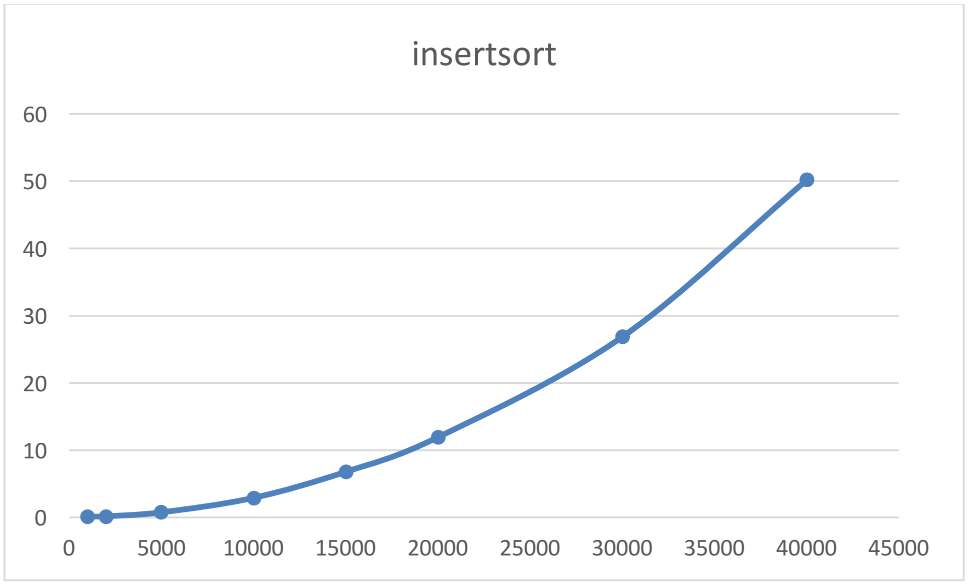
b) Table of running time

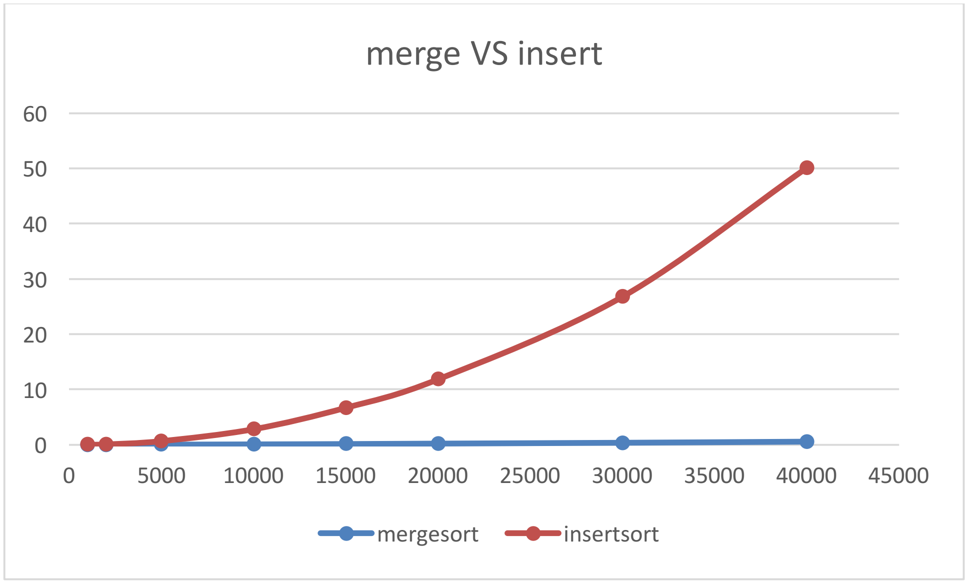
	1000	2000	5000	10000	15000	20000	30000	40000
mergesort	0.007057	0.014035	0.043636	0.08854	0.141296	0.215232	0.354655	0.536859
insertsort	0.026212	0.101341	0.689516	2.841501	6.71479	11.864983	26.794783	50.12328

c)



CS 325 Summer 17
HW 1 – 25 points





d)

For merge sort, the curve is more like linear graph, but insertion sort more like polynomial curve.

e)

For merge sort, the theoretical running time is O(nlogn), in my case, the running time is like kn, because when n goes to infinity, the logn will close to a constant value. So when n is big enough, nlogn will becom kn. So, the experimental running time is consistent with theoretical running time.

CS 325 Summer 17
HW 1 – 25 points

For insertion sort, the theoretical running time is O(n^2), in my case, the running time is like kn^2 + cn. Because of the n is infinity, so we just care about n^2. So, the experimental running time is consistent with theoretical running time.

EXTRA CREDIT: *It was the best of times, it was the worst of times…*

Generate best case and worst case input for both algorithms and repeat the analysis in parts b) to d) above. Discuss your results.

For best case, the array should a sorted array, just like [1,2,3,4,5,6,7,8,9,10,….,n-1,n]

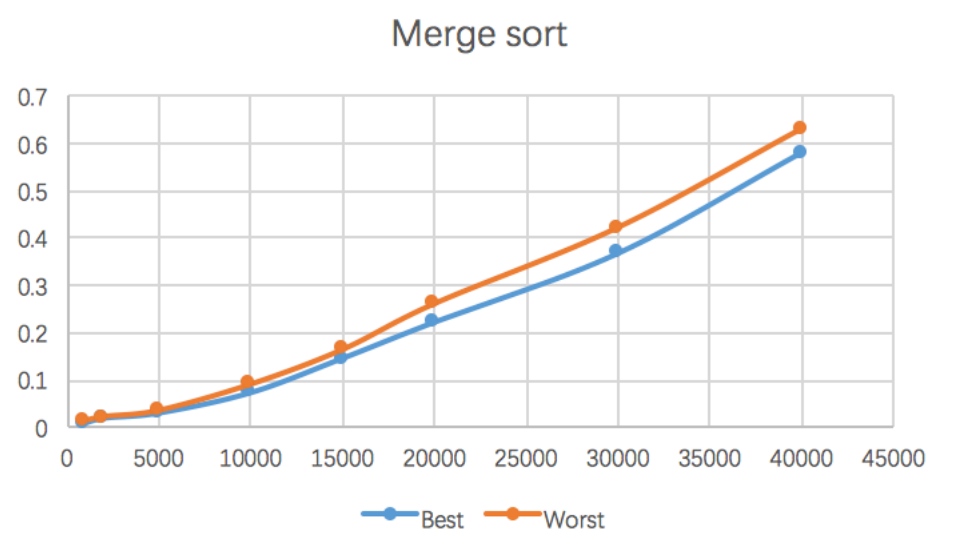
For worst case, the array should a reversed sorted array, just like [n,n-1,….10,9,8,7,6,5,4,3,2,1]

Merge sort

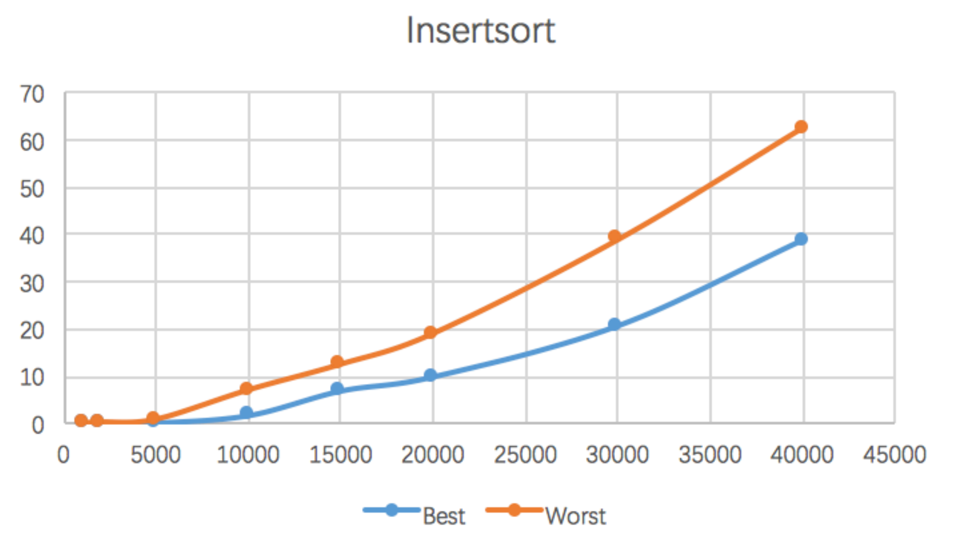
	1000	2000	5000	10000	15000	20000	30000	40000
Best	0.008232	0.018185	0.029177	0.072298	0.144984	0.221502	0.367475	0.580795
Worst	0.012336	0.020344	0.033452	0.088891	0.162445	0.259652	0.421392	0.630894

Insertion sort

	1000	2000	5000	10000	15000	20000	30000	40000
Best	0.025942	0.080783	0.227832	1.786369	6.967886	9.954441	20.723837	38.778623
Worst	0.029883	0.116278	0.731343	6.964082	12.423018	18.957572	38.882936	62.623159



CS 325 Summer 17
HW 1 – 25 points



For merge sort, the two curves are similar, they all like linear graph. For insertion sort, the best case look like linear graph, and worst case like polynomial curve.