CS 325 –Analysis of algorithms

Name: Shou-Cheng Wu

ID: 933 920 148

Email: wusho@oregonstate.edu

---

a) Collect running time (second):

(The program runs in OSU sever)

	insertsort	mergesort
2000	0.734812	0.019949
6000	4.093061	0.066172
10000	11.38595	0.115587
14000	22.32527	0.167455
18000	36.85999	0.221842
22000	55.03628	0.278253

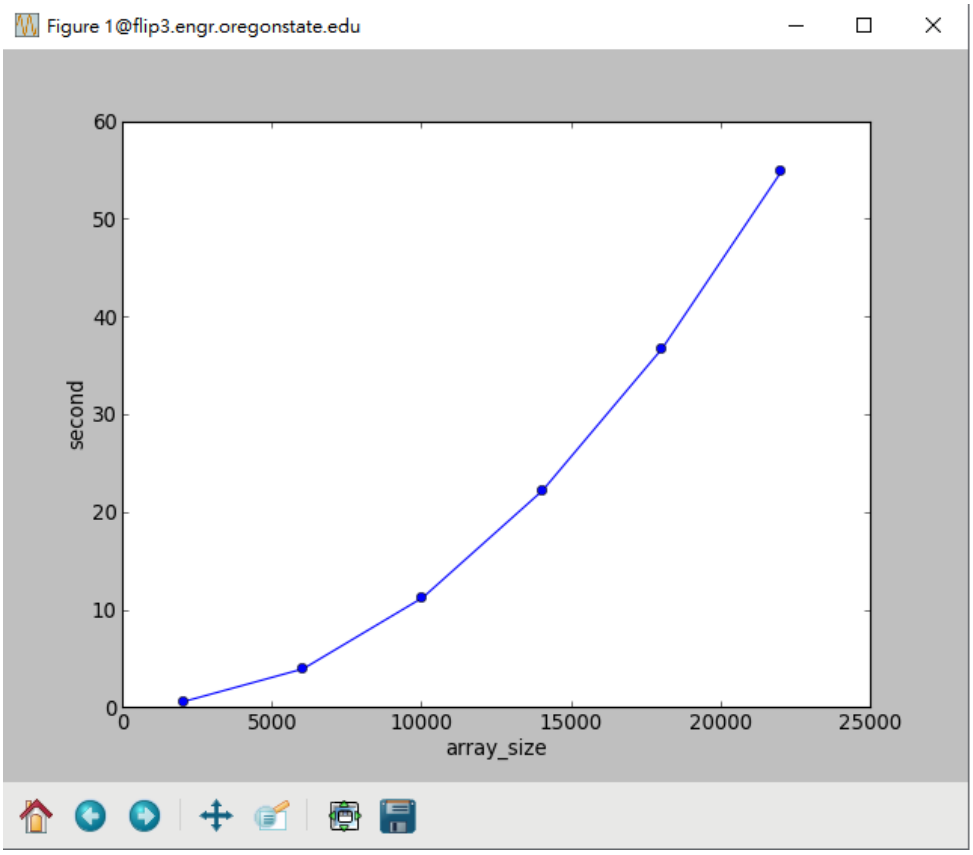
How do I generate the random arrays?

A: I import random at the beginning. Then, I write a function which could return an array if we input the range of integers we want to generate and the sizes of the array. Next, I use an empty array to store it. In other words, it contains multiple arrays which size are 2000, 6000 … 22000.

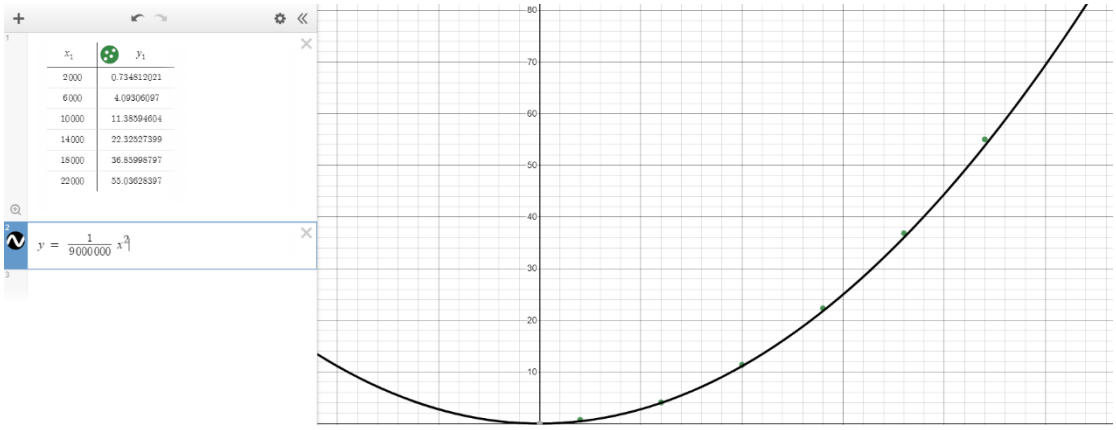
```
def random_list(min,max,length):

        outlist=[]
        for i in range(0,length):
                c = random.randint(min,max)
                outlist.append(c)
        return outlist

num_list = []

for i in range (2000,26000,4000):
        a = random_list(0,10000,i)
        num_list.append(a)
```

b) Plot the data and fit the curve:

Insert sort algorithm



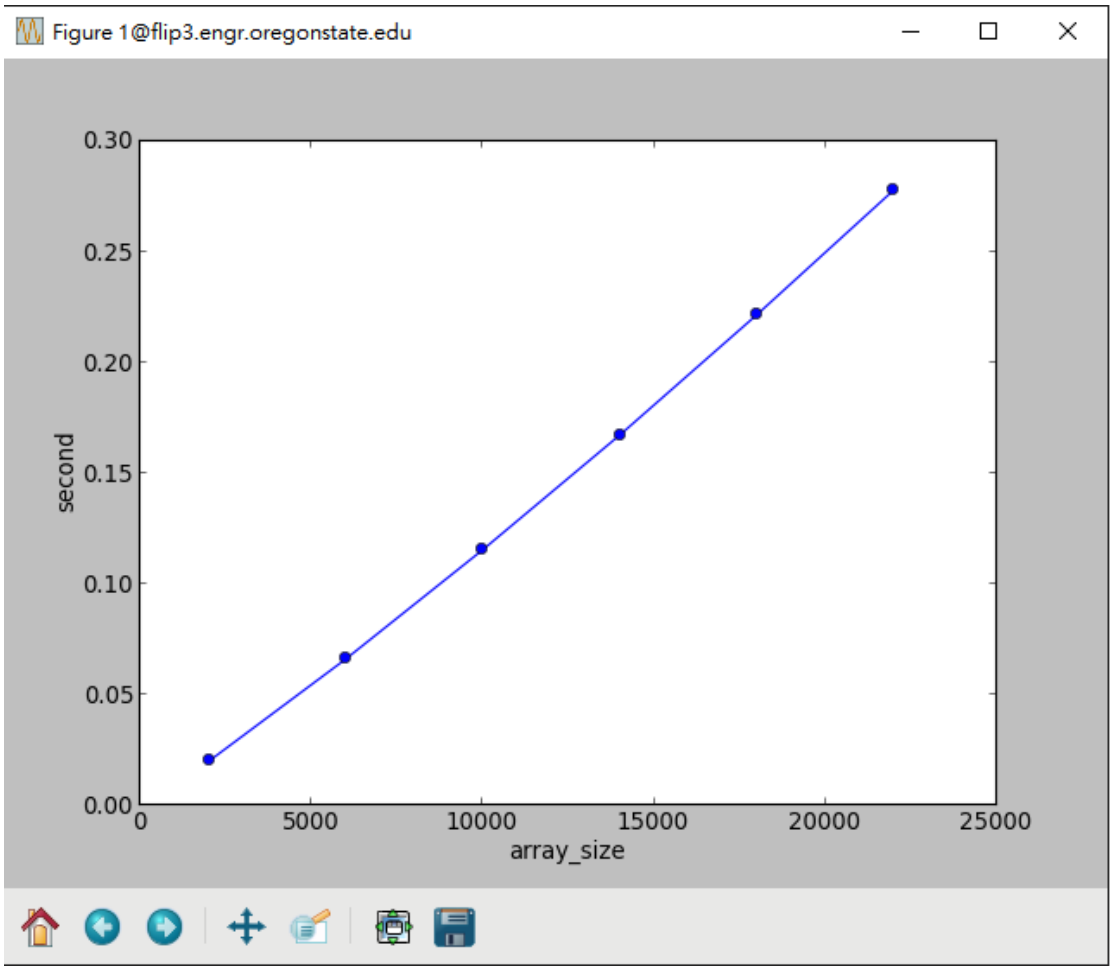
Insert Sort



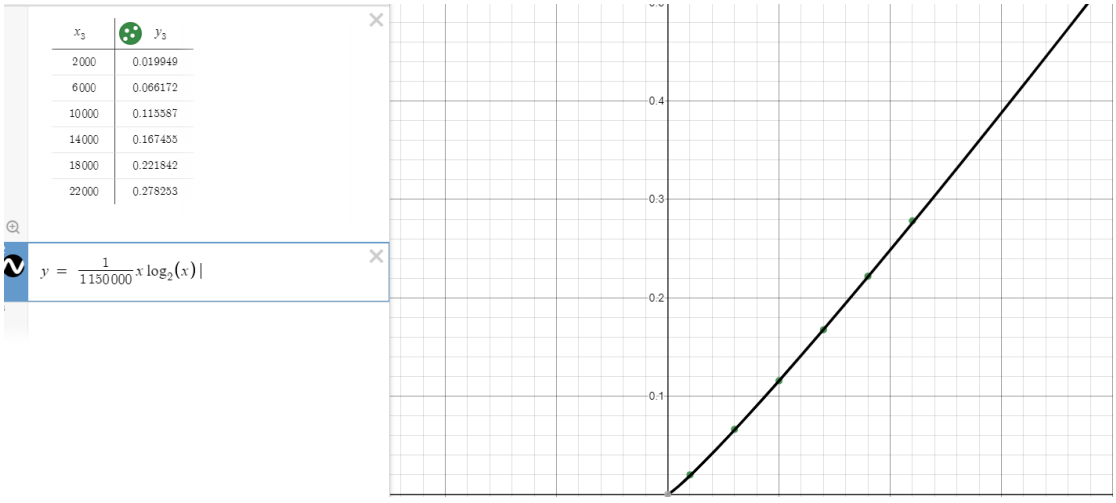
y= $\frac{1}{9000000}\ x^2$ can fit the data

Insert sort: O(n^2)

Merge Sort



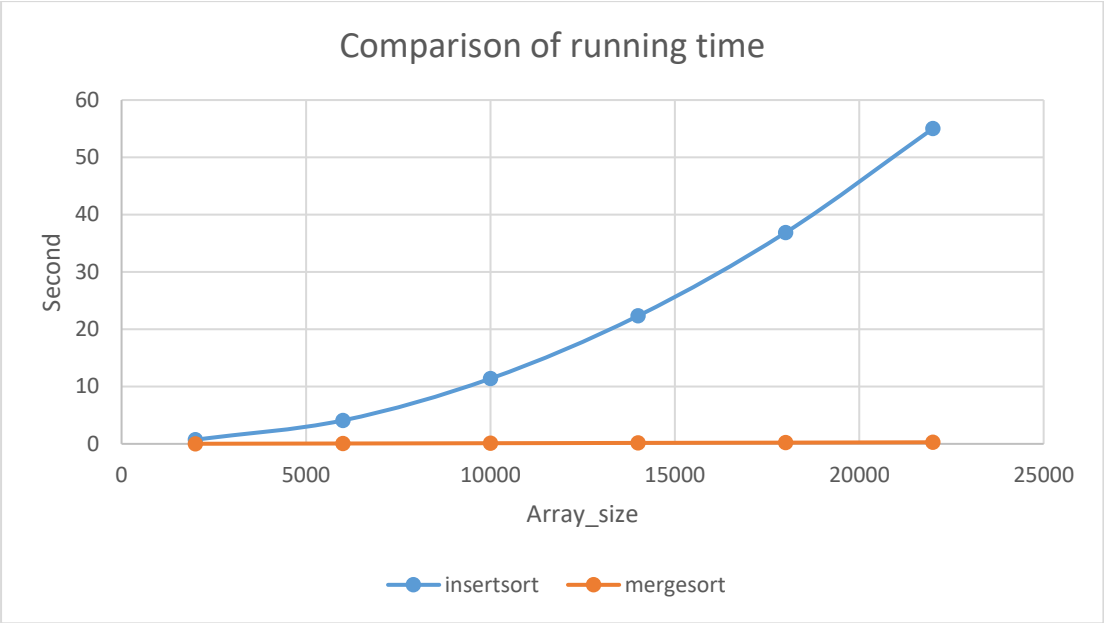
Merge Sort

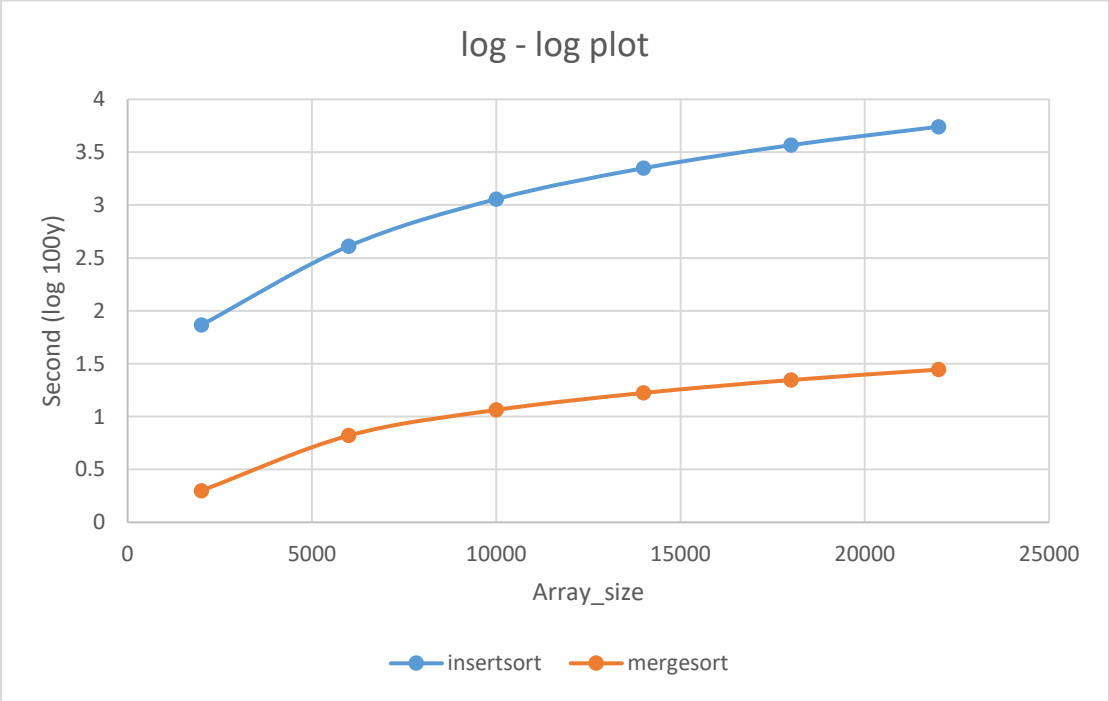


y = $\frac{1}{1150000}$ x $log_2 x$ can fit the data

Merge Sort: O(nlog_2n)

c) **Combine-**





Comparison between insert sort and merge sort

y here is actually log(100y)because the speed of merge sort is too fast. Its running time is less than 1. Using log(100y)could have a better graph or it would have a negative value in the graph.

As long as the size of the array is large enough, merge sort is faster than insert sort because the time complexity of merge sort is O (n$log_2 n$) and the time complexity of insert sort is

O(n^2)

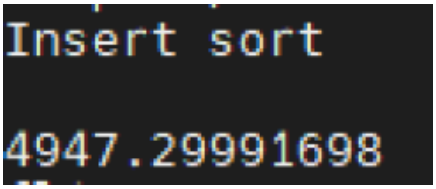
d) Prediction

About insert sort algorithm, I guess the equation is y=$\frac{1}{9000000}$ x^2. So, if the size of array is 200000, y would be $\frac{1}{9000000}$ * 200000 * 200000 =4444 (s)

About merge sort algorithm, I guess the equation is y = $\frac{1}{1150000}$ x$log_2\ x$. So, if the size of array is 200000, y would be $\frac{1}{1150000}$ * 200000* $log_2$200000= 2.966(s)

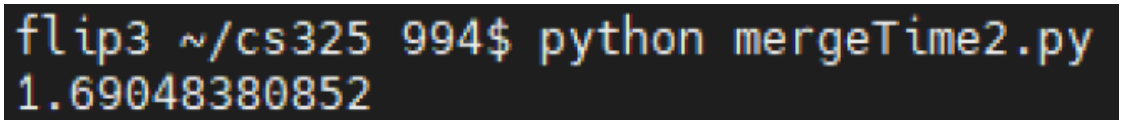
e) Comparison

Doing the insert sort algorithm by my code, the running time of it is 4947.299 seconds. And the theoretical running time is 4444 seconds. They are pretty similar, so my guessing equation is almost correct.



Experimental running time of insert sort.

Doing the merge sort algorithm by my code, the running time of it is 1.6904 seconds. And the theoretical running time is 2.966 seconds. Actually, the theoretical running time is twice of my experimental running time.



Experimental running time of merge sort.

Then, I do it again with a larger n. N is equal to 5000,1000,15000 … 50000 this time. My guessing equation becomes y=$\frac{1}{2200000}$ x $log_2\ x$. If n = 20000, y would be 1.5509 seconds.

That would be very similar as my new theoretical running time. Although the time complexity is still O(n$log_2 n$), in order to have a more precise guessing equation on our algorithms, we need to have a larger n to test.

$$y = \frac{1}{2200000} x \log_2(x)$$

x_1	y_1
9000	0.029249907
10000	0.062211037
15000	0.096895933
20000	0.132843018
25000	0.20029211
30000	0.213359833
35000	0.241859913
40000	0.280117989
45000	0.325939894
50000	0.362513065