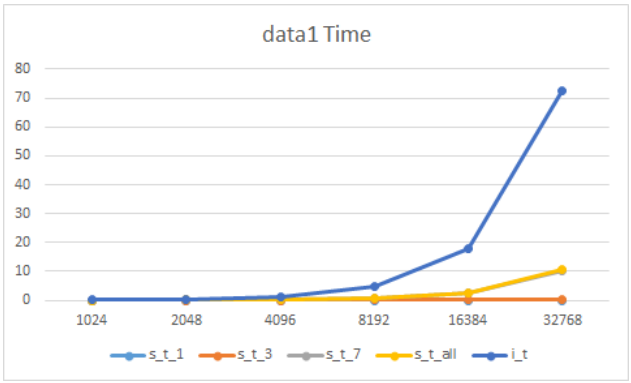
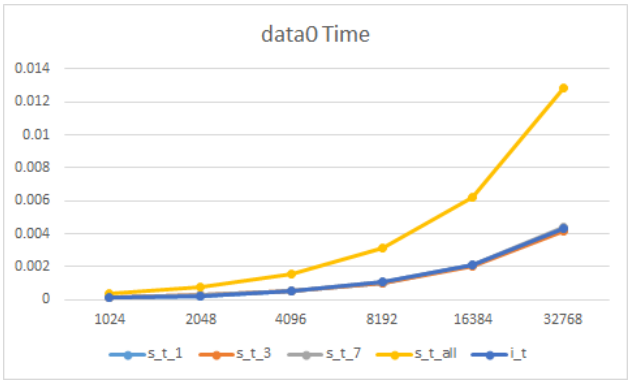
HOMEWOEK 2

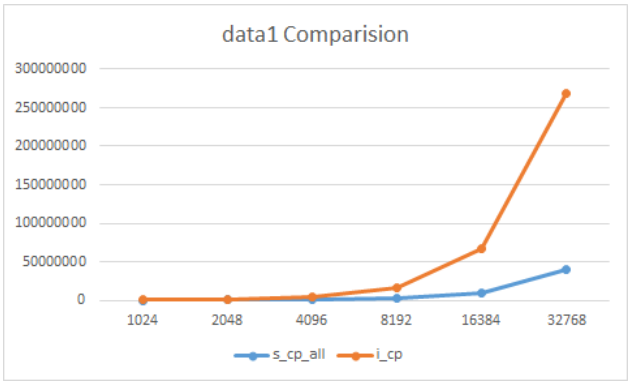
Runlin Hou
ECE, School Of Graduate Studies
Rutgers University
hourunlinxa@gmail.com

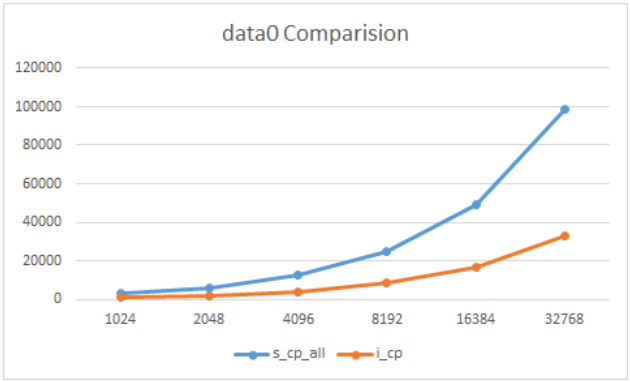
Question 1

As we known, shellsort would become insertion sort when h reduce to one. Meanwhile, the insertion sort will be more efficient when the sequence is almost sorted. The shellsort is more efficient because it take a little time to sort some shorter sequences divided from the original sequence, so that the sequence becomes an almost sorted sequence. By doing so, the insertion sort phase can save much more time.









As we can see in the graph, we deal with a sorted sequence. Time consumption and comparision amount of shellsort are both three times of the insertion sort, since insertion sort just do a traversal but shell do three times.

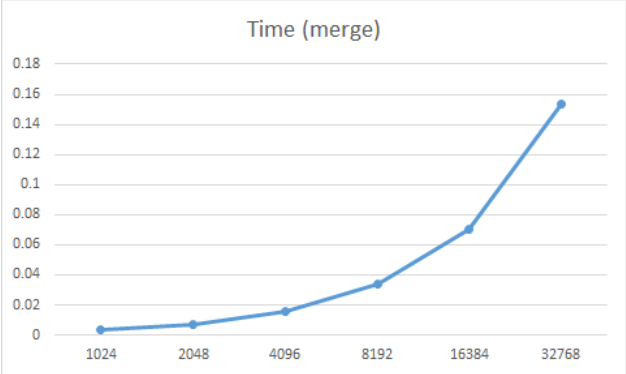
But as we can see, when we are dealing with a chaotic sequence, insertion sort takes much more time and comparision. Because every data is far from where it should be, it would takes more comparision to find its position.

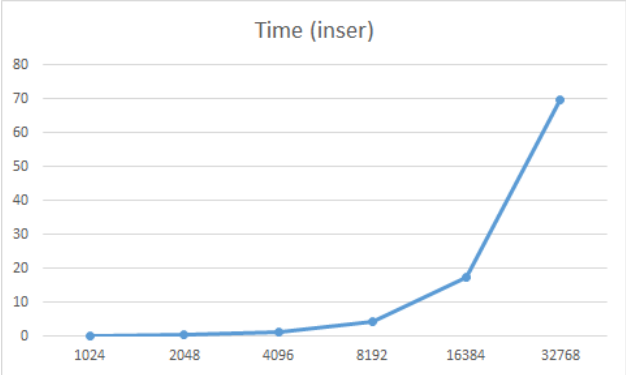
Question 2

The result and time consumption are as follow:

TABLE I
KENDALL TAU DISTANCE

1024	264541
2048	1027236
4096	4183804
8192	16928767
16384	66641183
32768	267933908





My basic logic of deciding KTD is to set one sequence as reference sequence, which means we will take whatever the fisrt value in the reference sequence is the smallest and whatever the last value in the reference sequence is the largest value. Then we will resort the other sequence based on the index of the reference sequence. Then we find the inverse pairs in the 2nd sequence, we find the KTD.

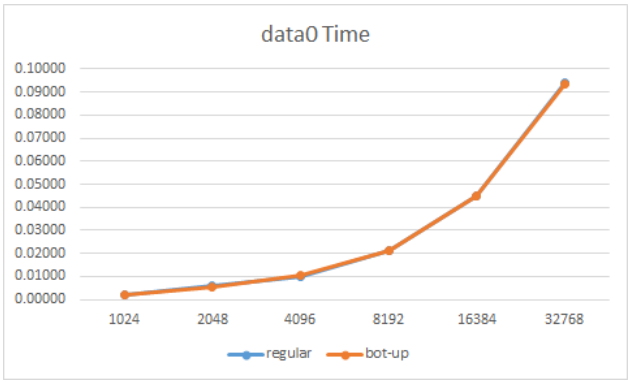
The way to decide the amount of inserve pairs would is to find the exchange time of a stable sort algorithm. Cause every exchange means the latter value is smaller than the former value, which is a pair of inverse values.

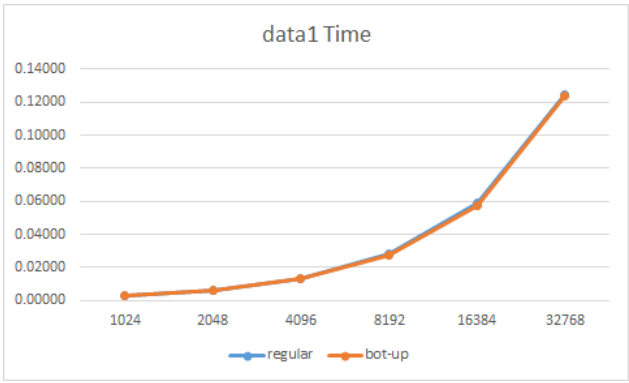
In my implementatoin, I use both merge sort and insertion sort to find KTD. As expected, the time consumption refers to $(O(NlogN))$ and $O(N^2)$.

Question 3

Comparision and array access are the two operations should be consider in merge sort. For the comparisions, every time we finish a merge to the whole array takes N comparisions. And we will take $logN$ whole merge, since every time will take half of the array to merge together.

And for array access operation, we can decide from the amount of merge happens. Every merge is going to take a comparision, which means two read of arrays. And after deciding the smaller value, we will write it to the auxiliary array, which means one read from the original array and one write to the auxiliary array. Then we will write the value from auxiliary array back to original array, which mean one read from auxiliary array and one write to the original array. So we total have 6 array access in each merge, 2 writes and 4 reads. For whole process we will have $6NlogN$ array access.





From the time consumption we can see that the two algorithm as expected have a pretty similar time consumption.

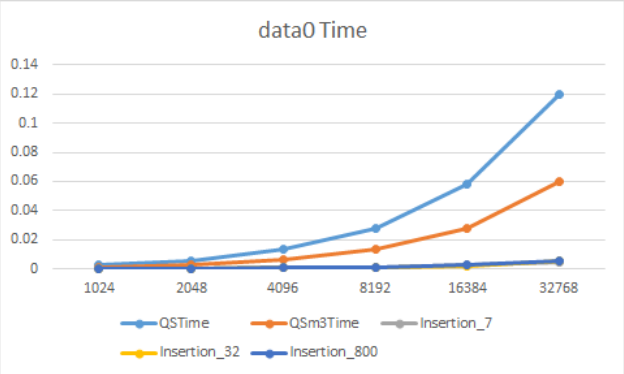
Question 4

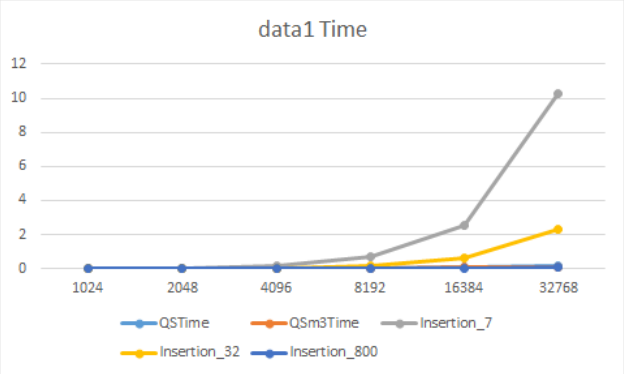
For this problem, I assume we know that the sequence we are going to deal with has 1024 repeats of 1, 2048 repeats of 11, 4096 repeats of 111 and 1024 repeats of 1111. So I set four pointers in a auxiliary array points to the positions where the 4 entities are supposed to start. So whin one traversal, I can set the entity into where it should be by moving the pointers. Then we write back this auxiliary array to sort it. The process rakes N comparisions and $2N$ writes. Which I think should be the smallest time consumption.

Question 5

As we can see on the follow graph, insertion sort can always take advantage in sort of data0, since data0 is already

sorted. But we it came to data1, which is unsorted sequence, the performence is much more worse than quicksort. If we increase the cut-off to 35 the performence of insertion sort can achive the same level with quicksort on the sqquence whose length is 1024, but the time increase ripadly as the length goes on. Meanwhile, if we want the insertion sort has a similar performence on the sequence whose length is 32768, we need to increase the cut-off to 800.





Question 6

We mark the sequence from left to right as 1 to 8 except the input and the result. 1

1) Mergesort (bottom-up). Every 4 item are sorted.
2) Quicksort (standard, no shuffle). Every entities before navy is smaller than it and every entities after navy is greater. And palm is not been exchanged.
3) Knuth shuffle. All the entities after silk are in place and plum, pink are not in place, which means this is not insertion sort.
4) Mergesort (top-down). The first 12 entities are sorted and the 13 to 18, 19 to 24 are sorted.
5) Insertion sort. Every thing beform silk is sorted and every thing after silk is the same as origin.
6) Heapsort. If we fill this sequence into a tree from 0 to 23, we can see that all child nodes are greater than their parents.
7) Selection sort. Every entity on the right of silk is larger and every entity on the left are sorted.
8) Quicksort (3-way, no shuffle). Every entities before navy is smaller than it and every entities after navy is greater. And palm is been exchanged, because it is the median of plum, lime and palm.