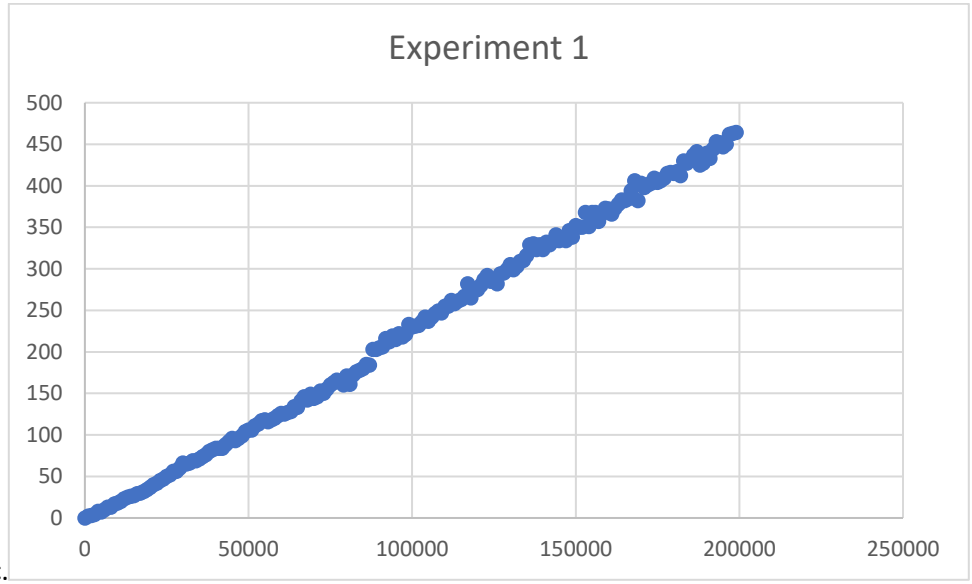
1. Parent(i)=(i-1)/4, child(i, j)=i * 4 + j

2. The algorithm we used on checking the four child is to keep a minimum value and check for if the next child is smaller than our min value and exchange the min we storage. Because we have 4 children, we used a for loop to calculate the min child to avoid redundancy.

3.(1) a. It is testing sorting a fixed number of items when the list size is constantly growing.

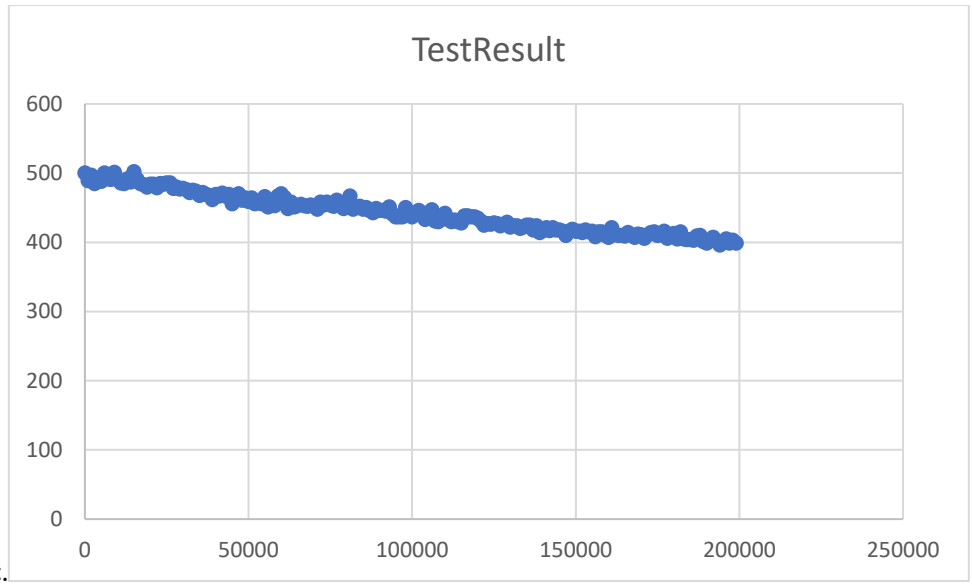
b. The outcome should be O(n), because the total list size is linearly growing, the time we need to traverse the list to sort it will linearly grow.

c. 

d. The outcome is reasonable.

(2) a. It is testing sorting a fixed sized list with k growing.

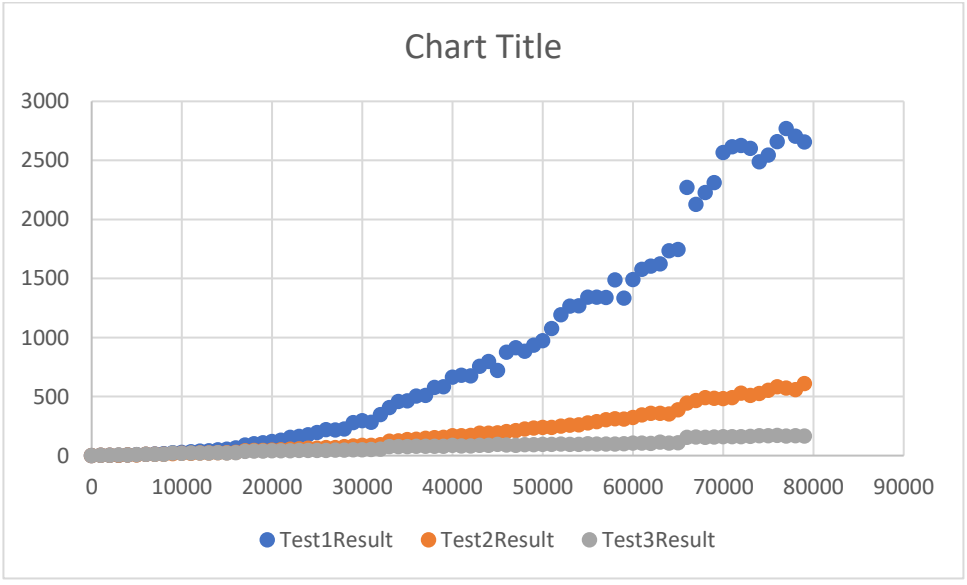
b. The outcome should O(1), because in our algorithm, we only used k to delete n-k items from the double linked list, which is in O(1) and it is far too small to take in to account since the operation we did with the heap takes much more time than that. As a result, changing k won't affect results too much.

c. 

d. The outcome is reasonable.

(3) a. It is testing different implementations of hash code's effecting the performance of our chained hash dictionary. The first implementation is to add the first four chars together; the second implementation is to add all the chars together; the third is basically the hash code method implemented by java's list ADT itself.

b. The third one should be the fastest one since it has the least possibility to have a giant cluster in one bucket. The first one should be the slowest because when there are lots of items(fake strings), there will be more items that have the same hash code for the first four chars, because the algorithm for randomizing the fake strings is to randomly choose a char from a to z, so there will be more items having hash code that will be near some middle point value.

c. 

d. The outcome accords to our prediction, the first test is the slowest, the third test is the fastest.