Vanessa Chen and vc86

Run WordGramBenchmark for wordgrams of size 2-10 and record
the number of WordGram values/objects that occur more than
once as reported by the runs. For example, with WSIZE = 2,
which generates 2-grams, the output of benchmark and benchmarkShift
each indicates that the total # wordgrams generated is 177,634
and that the # unique wordgrams is 117,181

This means there are 177,634 - 117,181 = 60,453 WordGram values that
occur more than once. Find these same numbers/values for other orders
of k and complete the table below for different k-grams/different
values of WSIZE

WSIZE	# duplicates
2	60,453
3	10,756
4	177618 - 175631 = 1,987
5	177610 - 176943 = 667
6	177602 - 177240 = 362
7	177594 - 177368 = 226
8	177586 - 177435 = 151
9	177578 - 177473 = 105
10	177570 - 177497 = 73

=====

Explain in your own words the conceptual differences between
the benchmark and benchmarkShift methods.
Answer these questions:

(1) Why the results of these methods should be the same in
terms of changes made to the HashSet parameter passed to each method.
They should be the same, because the two methods are working to get
the same result, but doing them in a different method.
I have made comments on both methods in WordGramBenchmark.java
illustrating what the steps are doing.
Because of the way the methods are written, they both are adding each
constructed wordgram to the hashset parameter.
Benchmark first creates the String[] of all the words in the file
(after converting from a list), and loops through each
word in the String[] and creates a wordgram at a starting point of k,
which increments. This gives the same result as creating one initial
wordgram and using the .shiftAdd() method, which starts at an index of
1 (comparative to the array its being called on) and appends a new
string to the end (each time this method is called). Either way, we
are adding each wordgram to the set.

(2) What are the conceptual differences between the two

benchmarking methods
As described in the previous question, Benchmark essentially creates a new arrayList and adds each element of the file into this list.
Then, it converts this list into an array and loops through each element of the array (minus WSIZE then +1) and creates a new wordgram starting at that index.
However, BenchmarkShift creates a new String[] of size WSIZE, and sets each element equal to an element in the source file (until size WSIZE).
Then, it creates a new WordGram to start with, and it loops through each element in the source file, performing the .shiftAdd() method on the
current wordgram (which creates a new arrayList, starting at index one of the arrayList of the singular wordgram until the size of the WordGram,
and adding String s as the last element.
They both add these individual wordgrams to the set of wordgrams.

Thus, conceptually, Benchmark first creates the arrayList of every word, converts it to an array, then loops through each word and separates the word array into a WordGram beginning with word[k] as k increments, while BenchmarkShift creates an array of size equal to one
wordgram, creates an initial wordgram, and loops through each element from the source file and continues to create new wordgrams that are a shift 1 to the right of the current wordgram. Basically, Benchmark initializes the entire array of every word first, then creates new WordGrams from that start array, while BenchmarkShift starts with one wordgram and continues to get new wordgrams by shifting it and adding each word from source file to the new shifted wordgram.

However, BenchmarkShift executes faster than benchmark.

(3) Is the total amount of memory allocated for arrays
the same or different in the two methods? Account for
arrays created in the methods and arrays created by
WordGram objects. Try to be quantitative in answering.

For Benchmark, it creates an empty arrayList that ends up as size = number of elements in the file (lets call this n).
So, a block of contiguous memory is allocated to this arrayList with n elements. Next, we loop through each element of this array
(n - WSIZE + 1) times, each time creating a new WordGram. Each time we create a new WordGram, we initialize the String[] myWords to the size of the WordGram (WSIZE). System.arraycopy(source, start, myWords, 0, size); copies a source array from a specific beginning position to the destination array from the specified position. This means we are assigning values to the array myWords which we already initialized (/ allocated memory for). So, the total allocated memory is (n - WSIZE + 1) * (WSIZE) = [n*WSIZE] - WSIZE^2 + WSIZE; however, we must add n

to this equation because we initialized an arraylist of size n at the beginning, so the final equation is n + [n*WSIZE] - WSIZE^2 + WSIZE

For benchmarkShift, we create an array of size WSIZE, which allocates a space of WSIZE. Then, we create a WordGram of size WSIZE, which also allocates a space of WSIZE. Then, for each next word in the file (we have n-WSIZE left) and calling .shiftAdd(), which creates a new array of size WSIZE (and creates a wordgram from this array). So, we have WSIZE + ((n-WSIZE) * WSIZE) = [n*WSIZE] - WSIZE^2 + WSIZE

Thus, because Benchmark initializes an empty arraylist of size n at the beginning, then loops through each element of the array to create wordgram arrays, while benchmark shift initializes a WSIZE sized array and then shifts through the entire array, Benchmark allocates more memory compared to benchmarkShift.