Techniques for Comparing Hash Functions :
Speed
Load (# of hashes with 2 data sources, 3, 4...)
Scatter plot – X axis is hash value, Y axis is original value.

Performance :
Yes you should see very good performance – if you are not then something might be wrong with your implementation.

Byte Array Question :
If you are using a bit-array-based filter, then you are going to be putting 32 flags in each int value in the list. Don't need to consider byte boundaries, but you do have to make sure you index properly.

4 Graphs verification :

Yes those are the 4 graphs you should plot.

Hash Function coefficients question :
Not sure what you mean by this, please elaborate.

Task 1 and 3 Turnin :
Yes, that is correct, nothing but the report and graphs is required for these tasks.

Plots :

Task 1 plots :
You have freedom here – the goal is to make some point about differences between the functions, including with random data from universe and correlated data (data in a sequence – like 1-> m) from universe. One way to approach it is to have 2 plots for

each Hash function – 4 in total – where 1 pair plots the performance with random data, and the other plots the performance with correlated data.

Task 3 plots :

For each plot :

- fix c to either 10 or 15, fix hash function to either type 1 or type 2.
- sweep k from .4 * c to 1.0 * c (making sure k is an integer – this is how many hashes each function has)
- for each value of c, for each value of k, you will run 10 **trials** :
    - A **trial** is a single execution of adding the m data points and then checking membership of the entire set of 2 m data points.
    - When you finish a trial you should compare what the BF says about

    membership with what you know about the membership – if you use the data files provided, then the first m sequential datapoints should be added, and all 2m should be checked.

    ○ From this you should get a false negative rate (Which better be 0 or you have a problem) and a false positive rate that will be some non-zero value – save this false positive rate.
    ○ Re-execute the trial, **with a new hash function (with new seeds/coefficients)** and repeat above.

- After the 10 trials, take the 10 values for the false positive rate and find median. Then repeat, with ++k
- Plot the median value for each set of trials (10 runs for each k and c). **And also plot the Theoretical false positive rate.**

Things to remember :

1) You must generate new seeds/coefficients for Tasks 1 and 3. You may perform these tasks in any coding environment you wish, including the stub code we have provided you. If you do use our stub code, **be certain that Task 2 runs how we wish it to run when you turn it in (using the given command line in the assignment).**

2) Only do random seed/coefficient generation 1 time per hash function generation (in the constructor).

3) For task 3, you can use the same dataset for each trial and each c/k combination – since the hash functions

are being remade (and coefficients are regenerated), using the same data isn't a problem.

4) For task 2, your code should execute in about **4 seconds** using hash type 2 and **20 seconds** using hash type 1. If it is substantially slower than this (and your computer is not a potato) you might have some improper stuff going on in your add/check functions, which leads to....

5) Keep **add** and **check** functions as skinny as possible. No extra baloney in there or it will slow things down a lot.