# Major assignment 2: Your submission

This is your assignment template for [BigDataX Major assignment 2](https://courses.edx.org/courses/course-v1:AdelaideX+BigDataX+3T2018/courseware/52ddf76bd8c24e5691e50ddee65f5cc3/240befde4592486a8837e9d9e20f4d9d/1?activate_block_id=block-v1%3AAdelaideX%2BBigDataX%2B3T2018%2Btype%40vertical%2Bblock%40b2db1afe137d4dd38e6fc7f7c0e25937). Save this document on our local machine and include all of your work within the relevant sections. Once you’ve completed all five parts of the assignment, upload the document via the submission area on the “[Submit your assignment](https://courses.edx.org/courses/course-v1:AdelaideX+BigDataX+3T2018/courseware/52ddf76bd8c24e5691e50ddee65f5cc3/240befde4592486a8837e9d9e20f4d9d/6?activate_block_id=block-v1%3AAdelaideX%2BBigDataX%2B3T2018%2Btype%40vertical%2Bblock%40d9c42326f98240f4a3419ff2f1548752)” page at the end of Major assignment 2.

# You will need to include:

# the answer to the question

# your working/calculations associated with each question

# the name of your associated code (.java) and output (.txt) files for each part of the assignment.

# Your answer will assist the University of Adelaide academic staff member assess your code submission. The point(s) value at the end of each question is for your working/calculation, your code and your output. You will have received points for correctly answering the questions in the course (on edX).

**Quick links:**

[Major assignment 2: Part 1](#_Major_assignment_2:)

[Major assignment 2: Part 2](#_Major_assignment_2:_2)

[Major assignment 2: Part 3](#_Major_assignment_3:)

# Major assignment 2: Part 1 Reservoir sampling

1. Your response to steps 1 (a) – (d) [10 points]

Your associated working/calculations:

package major\_assignment;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Random;

import java.util.stream.IntStream;

/\*\*

 \* ReservoirSampling --- This class is an implementation of reservoir sampling.

 \*/

public class ReservoirSampling {

    /\*\*

     \* Determine if the j-th item should be included based on probability k/j

     \* @param k The maximum number of items to include in the sumple.

     \* @param j The index of the item to consider (the j-th element received).

     \* @return Returns true if the item should be included in the sample, false otherwise.

     \*/

    public static boolean shouldInclude(int k, int j) {

        boolean include = false;

        if (j <= k) {

            include = true;

        }

        else {

            double pr = (double)k / j;

            include = Math.random() <= pr;

        }

        return include;

    }

    /\*\*

     \* Select a sample of data points of size k from the input array.

     \* @param inputs The data points to take the sample on.

     \* @param k The maximum of items to include in the sample.

     \* @return Returns an integer array of points sampled form the input stream.

     \*/

    public static int[] sampleData(int[] inputs, int k) {

        Random rand = new Random();

        ArrayList<Integer> sample = new ArrayList<Integer>();

        int n = 0;

        for (int item : inputs) {

            n++;

            if (shouldInclude(k, n)) {

                // determine where the item should be inserted

                if (n <= k) {

                    sample.add(item);

                } else {

                    int insertIndex = rand.nextInt(k);

                    sample.set(insertIndex, item);

                }

            }

            // print the current sample

            System.out.print(String.format("%1$4s", n));

            System.out.println(" -- " + sample);

        }

        // generate the output array

        int[] sampleArray = new int[sample.size()];

        for (int i = 0; i < sample.size(); i++) sampleArray[i] = sample.get(i);

        return sampleArray;

    }

    /\*\*

     \* The main function to test the reservoir sampleing method.

     \* @param args Command line areguments are ignored.

     \*/

    public static void main(String[] args) {

        int[] dataStream = IntStream.rangeClosed(1, 1000).toArray();

        sampleData(dataStream, 10);

    }

}

The name of any associated code (.java) and output (.txt) files in your .zip file:

* ReservoirSampling.java

1. Your response to step 2 [4 points]

Your sample/s for first 10 elements goes here:

|  |
| --- |
| [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] |

Your sample/s for first 50 elements goes here:

|  |
| --- |
| [27, 49, 40, 39, 23, 6, 19, 43, 11, 32] |

Your sample/s for first 100 elements goes here:

|  |
| --- |
| [27, 49, 59, 39, 72, 81, 19, 92, 100, 91] |

Your sample/s for first 500 elements goes here:

|  |
| --- |
| [346, 410, 409, 431, 401, 425, 451, 456, 377, 307] |

Your sample/s for first 1000 elements goes here:

|  |
| --- |
| [346, 574, 409, 431, 634, 740, 979, 456, 377, 998] |

Your associated working/calculations:

        int[] dataStream = IntStream.rangeClosed(1, 1000).toArray();

        sampleData(dataStream, 10);

The name of any associated code (.java) and output (.txt) files in your .zip file:

* ReservoirSampling.java
* ReservoirSampling.txt

# Major assignment 2: Part 2 Frequent itemsets

        // create the list of item names

        String[] itemNames = new String[150];

        for (int i = 1; i <= 150; i++) itemNames[i-1] = Integer.toString(i);

        // create the apriori object

        Apriori apriori = new Apriori(itemNames);

        // generate the baskets

        for (int basketIndex = 1; basketIndex <= 150; basketIndex++) {

            ArrayList<String> basket = new ArrayList<String>();

            // selet the items to add to the basket -- if basketIndex mod item = 0

            for (int item = 1; item <= 150; item++) {

                if (basketIndex % item == 0) {

                    basket.add(Integer.toString(item));

                }

            }

            // add the basked to the apriori object

            apriori.addBasket(basket.toArray(new String[basket.size()]));

        }

1. Your answer for Question 1 and all associated working/calculations [3 points]

Your answer for Question 1 goes here:

|  |
| --- |
| [1] 150  [2] 75  [3] 50  [4] 37  [5] 30  [6] 25  [7] 21  [8] 18  [9] 16  [10] 15  [11] 13  [12] 12  [13] 11  [14] 10  [15] 10  [16] 9  [17] 8  [18] 8  [19] 7  [20] 7  [21] 7  [22] 6  [23] 6  [24] 6  [25] 6  [26] 5  [27] 5  [28] 5  [29] 5  [30] 5 |

Your associated working/calculations:

    /\*\*

     \* If the support threshold is 5, which items are frequent?

     \* @param apriori The initialized Apriori object.

     \*/

    public static void Question01(Apriori apriori) {

        System.out.println("--- Question 1: Frequent Items ---");

        ArrayList<ItemCount> frequentItems = apriori.getSingletons(5);

        apriori.printItemCountList(frequentItems);

    }

The name of any associated code (.java) and output (.txt) files in your .zip file:

* Apriori.java
* FrequentItemsets.java

2. Your answer for Question 2 and all associated working/calculations [3 points]

Your answer for Question 2 goes here:

|  |
| --- |
| --- Question 2: Baskets Containing (5, 20) ---  [20, 40, 60, 80, 100] |

Your associated working/calculations:

    /\*\*

     \* Get the indexes for a list of items.

     \* @param apriori The initialized Apriori object.

     \* @param items The item names to get the item indexes for.

     \* @return Returns an integer array with the indexes.

     \*/

    public static int[] getItemIndexes(Apriori apriori, String[] items) {

        int[] indexes = new int[items.length];

        for (int i = 0; i < items.length; i++) {

            indexes[i] = apriori.itemIndexes.get(items[i]);

        }

        return indexes;

    }

    /\*\*

     \* Find baskets containing a set of items.

     \* @param apriori The initialized Apriori object.

     \* @param set The indexes, NOT names, of the set items to look for.

     \* @param limit The maximum number of baskets to find.

     \*/

    public static void findBasketsWithSet(Apriori apriori, int[] set, int limit) {

        ArrayList<Integer> matchingBaskets = new ArrayList<Integer>();

        int basketsFound = 0;

        // get a list of baskets matching the items

        for (int basketIndex = 0; basketIndex < apriori.baskets.size(); basketIndex++) {

            int[] basket = apriori.baskets.get(basketIndex);

            int sum = 0;

            for (int itemIndex = 0; itemIndex < set.length; itemIndex++) {

                sum += basket[set[itemIndex]];

            }

            // if the basket contains all the items show it's number

            if (sum == set.length) {

                matchingBaskets.add(basketIndex + 1);

                basketsFound++;

                // quit the loop if the max number of baskets was found

                if (basketsFound == limit) {

                    break;

                }

            }

        }

        // show the matching baskets

        System.out.println(Arrays.toString(matchingBaskets.toArray(new Integer[matchingBaskets.size()])));

    }

    /\*\*

     \* Get 5 baskets with [5, 20] in them.

     \* @param apriori The initialized Apriori object.

     \*/

    public static void Question02(Apriori apriori) {

        System.out.println("--- Question 2: Baskets Containing (5, 20) ---");

        // get the item indexes to use for searching the matrix

        int[] items = getItemIndexes(apriori, new String[] {"5", "20"});

        findBasketsWithSet(apriori, items, 5);

    }

The name of any associated code (.java) and output (.txt) files in your .zip file:

* Apriori.java
* FrequentItemsets.java

3. Your answer for Question 3 and all associated working/calculations [8 points]

Your answer for Question 3 goes here:

|  |
| --- |
| --- Question 3 ---  Triple 1 : [1, 2, 10] 15  [10, 20, 30, 40, 50]  Triple 2 : [1, 3, 10] 5  [30, 60, 90, 120, 150]  Triple 3 : [1, 4, 10] 7  [20, 40, 60, 80, 100] |

Your associated working/calculations:

    /\*\*

     \* For a support threshold of 5, give three different frequent triples containing item 10. For each frequent triple, list 5 basket numbers where the different items appear together.

     \* @param apriori The initialized Apriori object.

     \*/

    public static void Question03(Apriori apriori) {

        System.out.println("--- Question 3 ---");

        // get the item index of basket item "10"

        int itemIndex = apriori.itemIndexes.get("10");

        // get the freqent triples containg item 10

        ArrayList<ItemCount> matchingTriples = new ArrayList<ItemCount>();

        ArrayList<ItemCount> frequentTriples = apriori.getFrequentItems(5, 3);

        for (ItemCount triple : frequentTriples) {

            if (np.contains(triple.items, itemIndex)) {

                matchingTriples.add(triple);

            }

        }

        // for the first 3 triples, find 5 baskets containing the triple

        for (int tripleIndex = 0; tripleIndex < 3; tripleIndex++) {

            ItemCount triple = matchingTriples.get(tripleIndex);

            System.out.println("Triple " + Integer.toString(tripleIndex+1) + " : " + triple);

            // get the baskets containing the triple

            System.out.print("    ");

            findBasketsWithSet(apriori, triple.items, 5);

            System.out.println();

        }

    }

The name of any associated code (.java) and output (.txt) files in your .zip file:

* Apriori.java
* FrequentItemsets.java

# Major assignment 3: Part 3 Data streams

    /\*\*

     \* Compute the answers for the hash function of a question.

     \* @param dataStream The data stream of items to process.

     \*/

    public static void ComputeAnswers(int[] dataStream, IntFunction hashFunction) {

        // create the estimator

        FlajoletMartin estimator = new FlajoletMartin(hashFunction, 5);

        // process the data stream items

        for (int item : dataStream) {

            estimator.processStreamItem(item);

        }

        // display the results

        System.out.println("Maximum tail length: " + estimator.maxTailLength);

        System.out.println("Estimate of the number of distinct elements: " + estimator.getUniqueEstimate());

    }

1. Your answer for Question 1a and 1b, including all associated working/calculations [4 points]

Your answer for Question 1a and 1b goes here:

|  |
| --- |
| --- Question 1 ---  Maximum tail length: 0  Estimate of the number of distinct elements: 1.0 |

Your associated working/calculations:

        System.out.println("--- Question 1 ---");

        ComputeAnswers(dataStream,

            (x) -> (2 \* x + 1) % 32);

The name of any associated code (.java) and output (.txt) files in your .zip file:

* FlajoletMartin.java

2. Your answer for Question 2a and 2b, including all associated working/calculations [4 points]

Your answer for Question 2a and 2b goes here:

|  |
| --- |
| --- Question 2 ---  Maximum tail length: 4  Estimate of the number of distinct elements: 16.0 |

Your associated working/calculations:

        System.out.println("--- Question 2 ---");

        ComputeAnswers(dataStream,

            (x) -> (3 \* x + 7) % 32);

The name of any associated code (.java) and output (.txt) files in your .zip file:

* FlajoletMartin.java

3. Your answer for Question 3a and 3b, including all associated working/calculations [4 points]

Your answer for Question 3a and 3b goes here:

|  |
| --- |
| --- Question 3 ---  Maximum tail length: 4  Estimate of the number of distinct elements: 16.0 |

Your associated working/calculations:

        System.out.println("--- Question 3 ---");

        ComputeAnswers(dataStream,

            (x) -> (4 \* x) % 32);

The name of any associated code (.java) and output (.txt) files in your .zip file:

* FlajoletMartin.java