

EECE 5643: Simulation and Performance Evaluation
Professor Ningfang Mi

Homework 4

- Assignment Due: 03/02/2023 -

Harrison Sun
Monday, Thursday 11:45 am - 1:25 pm
Completed: March 2, 2023

1 Ex. 4.1.11

Calculate \bar{x} and s by hand, using the two-pass algorithm, the one-pass algorithm, and Welford's algorithm in the following two cases.

(a) Data based on $n = 3$ observations: $x_1 = 1, x_2 = 6, x_3 = 2$.

One Pass Mean:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\bar{x}_1 = \frac{1}{1} \sum_{i=1}^1 x_i = 1$$

$$\bar{x}_2 = \frac{1}{2} \sum_{i=1}^2 x_i = 3.5$$

$$\bar{x}_3 = \frac{1}{3} \sum_{i=1}^3 x_i = 3$$

Welford's Algorithm Mean:

$$\bar{x}_i = \bar{x}_{i-1} + \frac{1}{i}(x_i - \bar{x}_{i-1})$$

$$\bar{x}_1 = 0 + \frac{1}{1}(1 - 0) = 1$$

$$\bar{x}_2 = 1 + \frac{1}{2}(6 - 1) = 3.5$$

$$\bar{x}_3 = 3.5 + \frac{1}{3}(2 - 3.5) = 3$$

Two-Pass Standard Deviation:

Pass 1: Calculate Mean (From Above)

$$\bar{x}_1 = 1$$

$$\bar{x}_2 = 3.5$$

$$\bar{x}_3 = 3$$

Pass 2: Compute the Squared Differences About \bar{x}

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

$$s_1 = \sqrt{\frac{1}{1} \sum_{i=1}^1 (x_i - \bar{x}_1)^2} = 0$$

$$s_2 = \sqrt{\frac{1}{2} \sum_{i=1}^2 (x_i - \bar{x}_2)^2} = 2.5$$

$$s_3 = \sqrt{\frac{1}{3} \sum_{i=1}^3 (x_i - \bar{x}_3)^2} = 2.16$$

One-Pass Standard Deviation:

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i^2) - \bar{x}^2$$

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i^2) - \bar{x}^2}$$

$$s_1 = \sqrt{\frac{1}{1} \sum_{i=1}^1 (x_i^2) - \bar{x}_1^2} = 0$$

$$s_2 = \sqrt{\frac{1}{2} \sum_{i=1}^2 (x_i^2) - \bar{x}_2^2} = 2.5$$

$$s_3 = \sqrt{\frac{1}{3} \sum_{i=1}^3 (x_i^2) - \bar{x}_3^2} = 2.16$$

Welford's Algorithm Standard Deviation:

$$\begin{aligned}
 v_i &= v_{i-1} + \left(\frac{i-1}{i}\right)(x_i - \bar{x}_{i-1})^2 \\
 v_1 &= v_0 + \left(\frac{0}{1}\right)(x_1 - \bar{x}_0)^2 = 0 \\
 v_2 &= v_1 + \left(\frac{1}{2}\right)(x_2 - \bar{x}_1)^2 = 12.5 \\
 v_3 &= v_2 + \left(\frac{2}{3}\right)(x_3 - \bar{x}_2)^2 = 14 \\
 s_i^2 &= \sqrt{\frac{v_{i-1} + \left(\frac{i-1}{i}\right)(x_i - \bar{x}_{i-1})^2}{i}} \\
 s_1^2 &= \sqrt{\frac{0}{1}} = 0 \\
 s_2^2 &= \sqrt{\frac{12.5}{2}} = 2.5 \\
 s_3^2 &= \sqrt{\frac{14}{3}} = 2.16
 \end{aligned}$$

(b) The sample path $x(t) = 3$ for $0 < t \leq 2$, and $x(t) = 8$ for $2 < t \leq 5$, over the time interval $0 < t < 5$.

One Pass Mean:

$$\begin{aligned}
 \bar{x} &= \frac{1}{\tau} \int_0^{\tau} x(t) dt \\
 \bar{x}_1 &= \frac{1}{2} \int_0^2 x(t) dt = 3 \\
 \bar{x}_2 &= \frac{1}{5} \int_0^2 x(t) dt + \frac{1}{5} \int_2^5 x(t) dt = 6
 \end{aligned}$$

Welford's Algorithm Mean:

$$\begin{aligned}
 \bar{x}_i &= \frac{1}{t_i}(x_1\delta_1 + x_2\delta_2 + \dots + x_i\delta_i) \\
 \bar{x}_i &= \bar{x}_{i-1} + \frac{\delta_i}{t_i}(x_i - \bar{x}_{i-1}) \\
 \bar{x}_1 &= 0 + \frac{2}{2}(3 - 0) = 3 \\
 \bar{x}_2 &= 3 + \frac{3}{5}(8 - 3) = 6
 \end{aligned}$$

Two Pass Standard Deviation:

Pass 1: Calculate Path Mean (From Above)

$$\bar{x}_1 = 3$$

$$\bar{x}_2 = 6$$

Pass 2: Compute the Squared Differences About \bar{x}

$$s_i = \sqrt{\frac{1}{\tau} \int_0^\tau (x(t) - \bar{x})^2 dt}$$

$$s_1 = \sqrt{\frac{1}{2} \int_0^2 (x(t) - 3)^2 dt} = 0$$

$$s_2 = \sqrt{\frac{1}{5} \int_0^2 (x(t) - 6)^2 dt + \frac{1}{5} \int_2^5 (x(t) - 6)^2 dt} = 2.45$$

One Pass Standard Deviation:

$$s_i = \sqrt{\left(\frac{1}{t_n} \sum_{i=1}^n x_i^2 \delta_i\right) - \bar{x}^2}$$

$$s_1 = \sqrt{\left(\frac{1}{2} \sum_{i=1}^1 3^2 \times 2\right) - 3^2} = 0$$

$$s_2 = \sqrt{\frac{1}{5}(3^2 \times 2 + 8^2 \times 3) - 6^2} = 2.45$$

Welford's Algorithm Standard Deviation:

$$v_i = v_{i-1} + \frac{\delta_i t_{i-1}}{t_i} (x_i - \bar{x}_{i-1})^2$$

$$v_1 = 0 + \frac{2 \times 0}{2} (3 - 0)^2 = 0$$

$$v_2 = 0 + \frac{3 \times 2}{5} (8 - 3)^2 = 30$$

$$s_i = \sqrt{\frac{v_i}{t_i}}$$

$$s_1 = \sqrt{\frac{0}{2}} = 0$$

$$s_2 = \sqrt{\frac{30}{5}} = 2.45$$

2 Ex. 4.2.2

- (a) Generate the 2000-ball histogram in Example 4.2.2.
- (b) Generate the corresponding histogram if 10,000 balls are placed, at random, in 1000 boxes.
- (c) Calculate the histogram mean (x) and the histogram standard deviation (s) for both the 2000 balls and the 10,000 balls.

2000-ball

$x = 2, s = 1.42$

10,000-ball

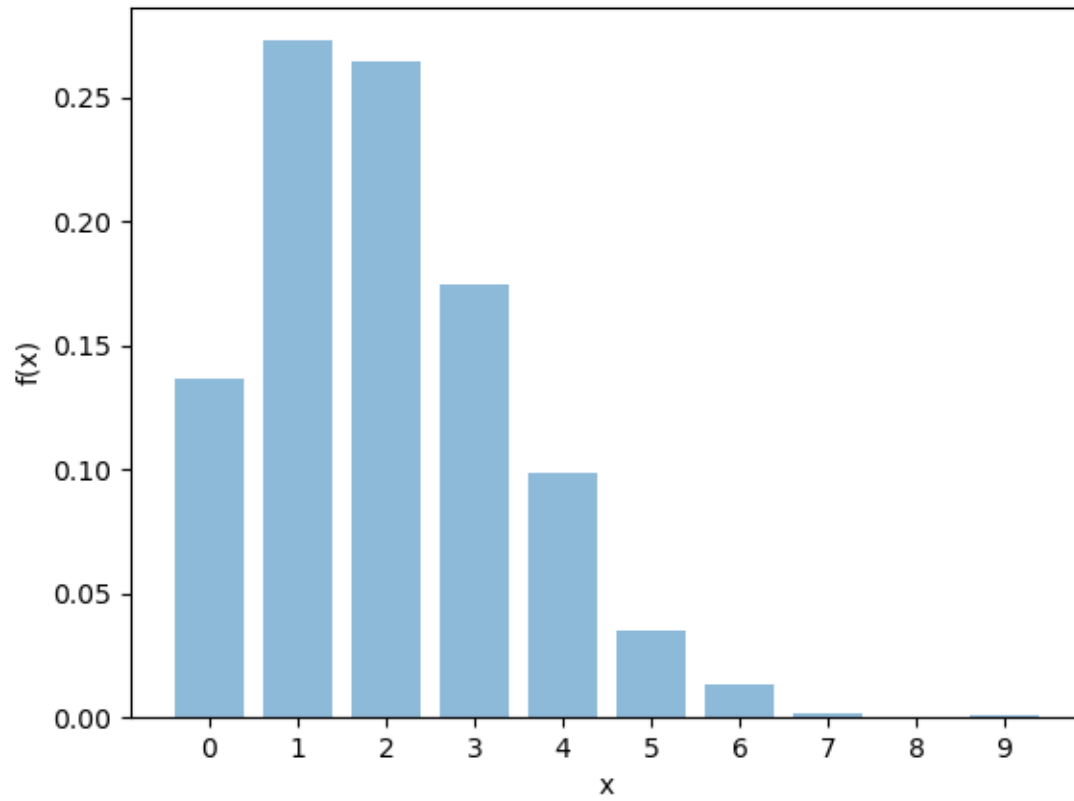
$x = 18, s = 3.1$

Terminal Output:

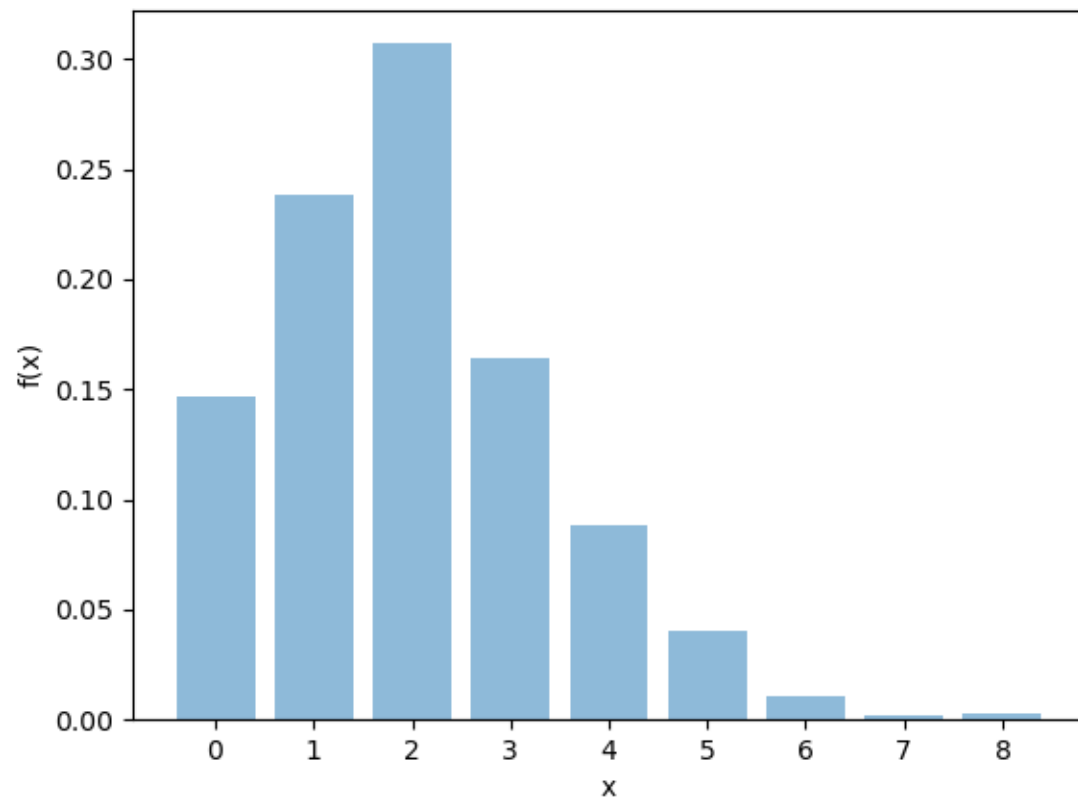
```
hlsun: Simulation-and-Perfor x + v
(base) hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.2 b 1000 n 2000 s 12345 f "4_2_a_12345.csv"
Mean: 2
Standard Deviation: 1.41915
(base) hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.2 b 1000 n 2000 s 246810 f "4_2_a_246810.csv"
Mean: 2
Standard Deviation: 1.41138
(base) hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.2 b 1000 n 2000 s 13579 f "4_2_a_13579.csv"
Mean: 2
Standard Deviation: 1.42688
(base) hlsun:Simulation-and-Performance-Evaluation$ python Histogram.py 4_2_a_12345.csv 4_2_a_12345.png
(base) hlsun:Simulation-and-Performance-Evaluation$ python Histogram.py 4_2_a_246810.csv 4_2_a_246810.png
(base) hlsun:Simulation-and-Performance-Evaluation$ python Histogram.py 4_2_a_13579.csv 4_2_a_13579.png
(base) hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.2 b 1000 n 10000 s 12345 f "4_2_b_12345.csv"
Mean: 10
Standard Deviation: 3.19906
(base) hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.2 b 1000 n 10000 s 246810 f "4_2_b_246810.csv"
Mean: 10
Standard Deviation: 3.11641
(base) hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.2 b 1000 n 10000 s 13579 f "4_2_b_13579.csv"
Mean: 10
Standard Deviation: 3.0835
(base) hlsun:Simulation-and-Performance-Evaluation$ python Histogram.py 4_2_b_12345.csv 4_2_b_12345.png
(base) hlsun:Simulation-and-Performance-Evaluation$ python Histogram.py 4_2_b_246810.csv 4_2_b_246810.png
(base) hlsun:Simulation-and-Performance-Evaluation$ python Histogram.py 4_2_b_13579.csv 4_2_b_13579.png
(base) hlsun:Simulation-and-Performance-Evaluation$
```

Part A:

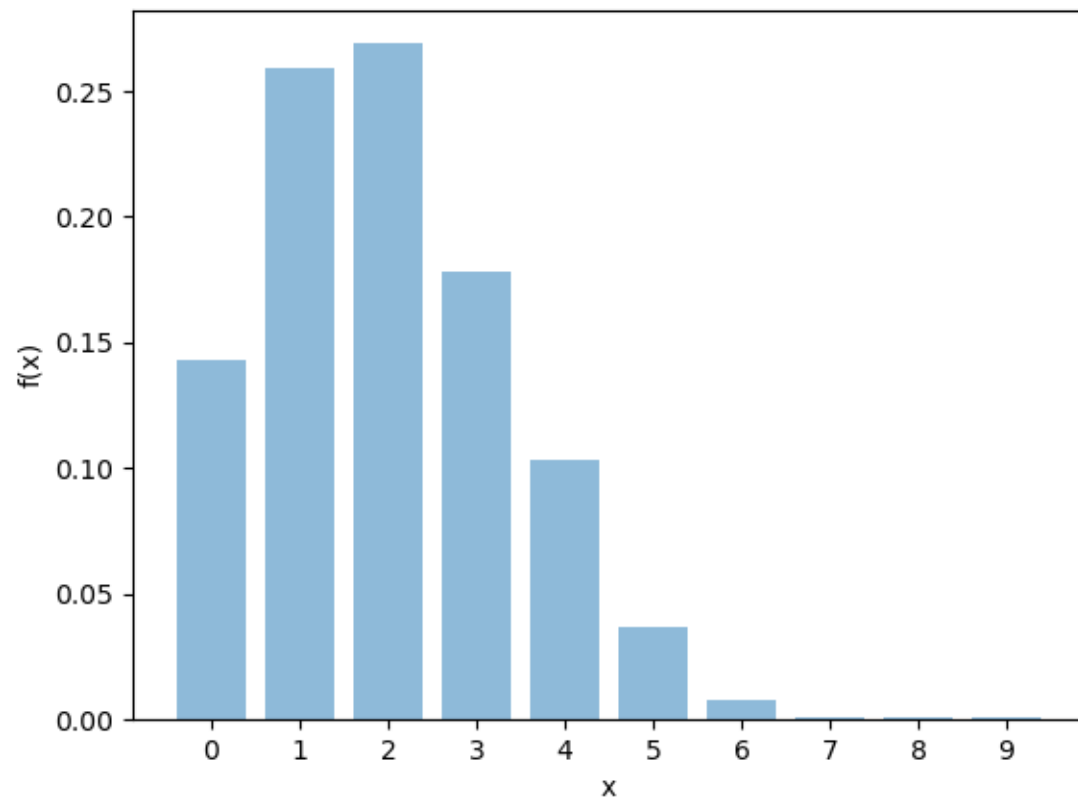
Seed: 12345



Seed: 13579

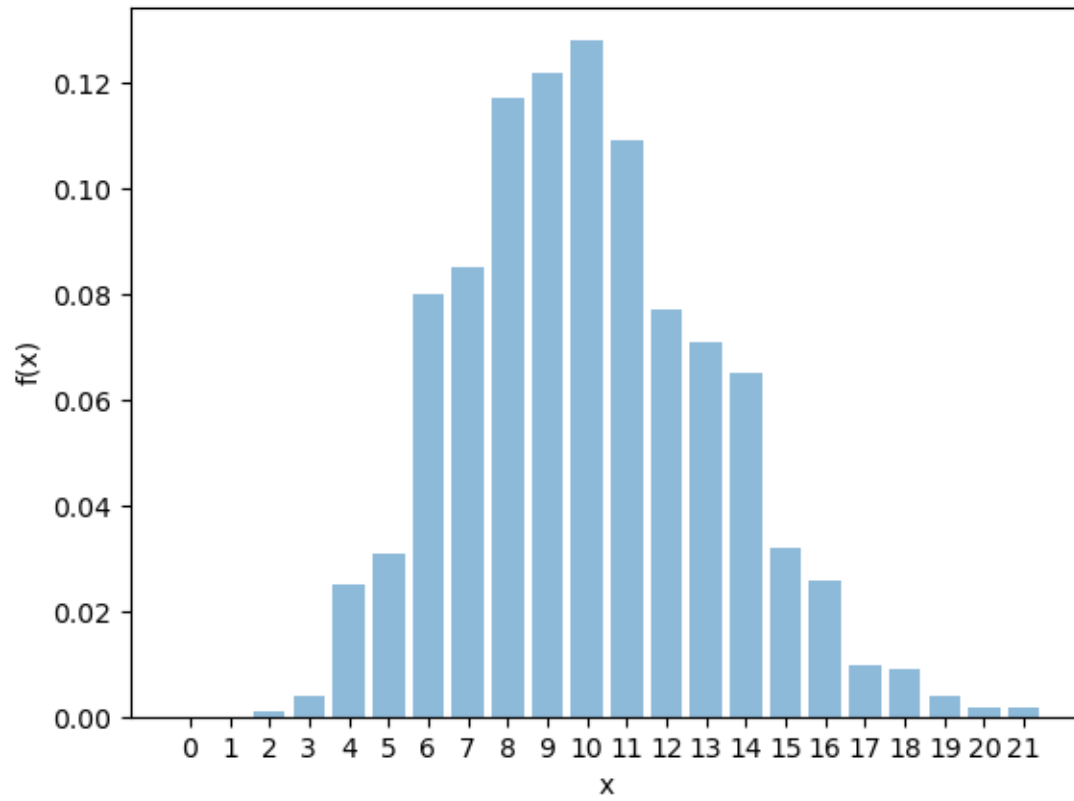


Seed: 246810

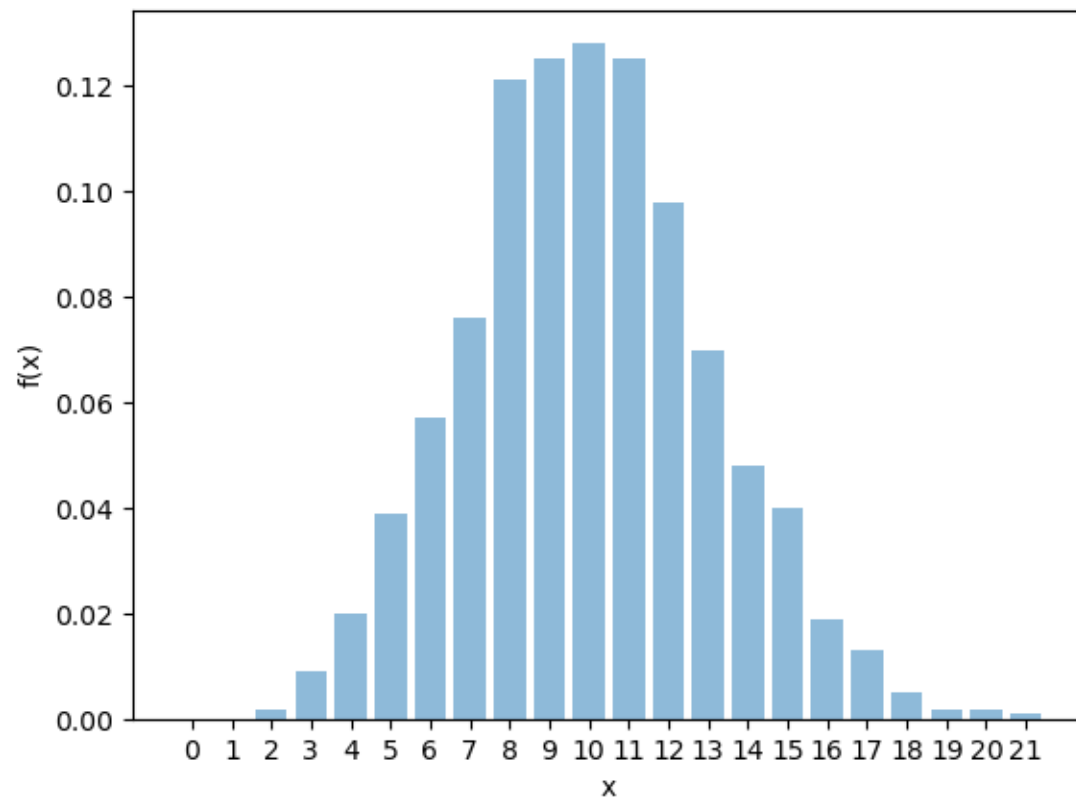


Part B:

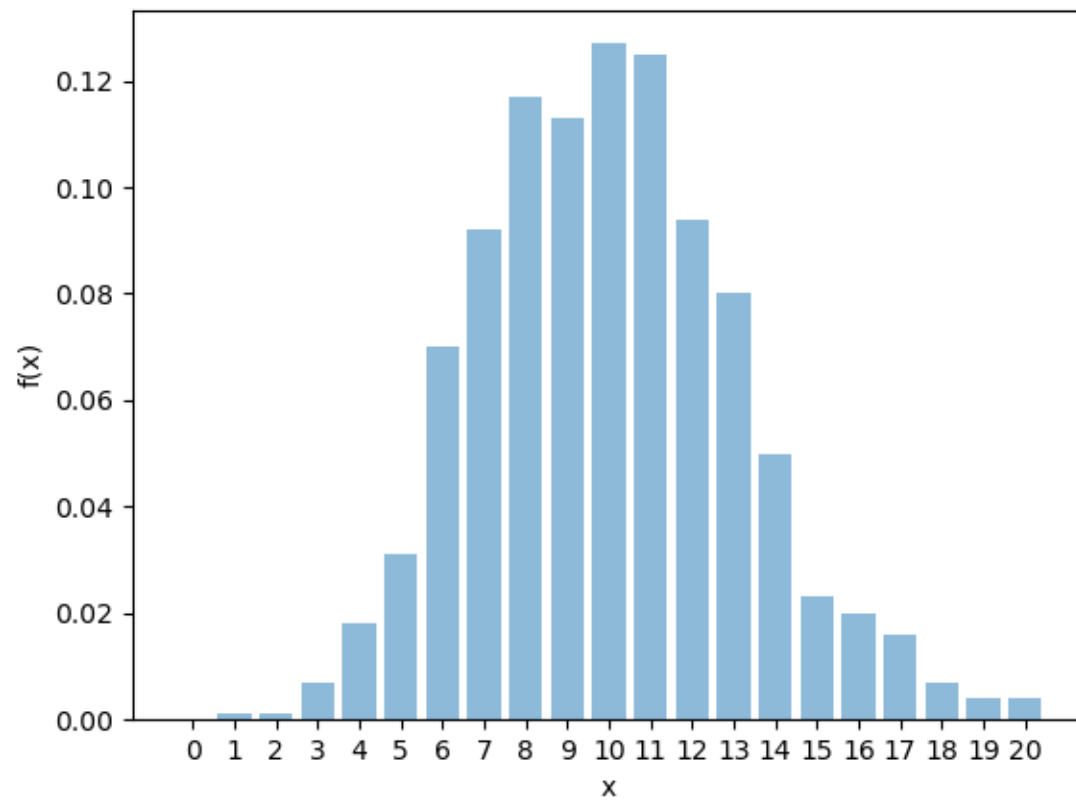
Seed: 12345



Seed: 13579



Seed: 246810



```

1  /**
2   * Harrison Sun (sun.har@northeastern.edu)
3   * EECE 5643 – Simulation and Performance Evaluation
4   * Homework 4.2
5   */
6
7  #include <cstdlib>
8  #include <cstring>
9  #include <fstream>
10 #include <stdio.h>
11 #include <exception>
12 #include <iostream>
13 #include <list>
14 #include <math.h>
15 #include <string>
16 #include <vector>
17 #include "c_lib/rvgs.h"
18 #include "c_lib/rngs.h"
19 #include "checkarg/checkarg.h"
20
21 #define DEFAULT_N_BOX 1000
22 #define DEFAULT_N_BALLS 2000
23 #define DEFAULT_SEED 12345
24 #define DEFAULT_OFILE "output.txt"
25
26 /**
27  * int main() – The main function
28  *
29  * @param int argc – the number of arguments
30  * @param char* argv[] – the arguments
31  *
32  * @return 0 if the program runs successfully
33  */
34
35 int main(int argc, char* argv[])
36 {
37     // Variable Declarations
38     int nBoxes{};
39     int nBalls{};
40     std::string outputFileName{};
41
42     // doubly linked list <box number, number of balls in box>
43     std::list<std::pair<long, long>> boxCount;
44
45     // Set the seed
46     for (int i = 0; i < argc; ++i)
47     {
48         if (*argv[i] == 's' && checkArg(argv[i + 1]))
49         {
50             PutSeed(std::stol(argv[i + 1]));
51             break;
52         }
53         else
54         {
55             PutSeed(DEFAULT_SEED);
56         }
57     }
58
59     // Set the number of boxes
60     for (int i = 0; i < argc; ++i)
61     {
62         if (*argv[i] == 'b' && checkArg(argv[i + 1]))
63         {
64             nBoxes = std::stol(argv[i + 1]);
65             break;

```

```

66     }
67     else
68     {
69         nBoxes = DEFAULT_N_BOX;
70     }
71 }
72
73 // Set the number of balls
74 for (int i = 0; i < argc; ++i)
75 {
76     if (*argv[i] == 'n' && checkArg(argv[i + 1]))
77     {
78         nBalls = std::stoi(argv[i + 1]);
79         break;
80     }
81     else
82     {
83         nBalls = DEFAULT_N_BALLS;
84     }
85 }
86
87 // Set the filestream
88 for (int i = 0; i < argc; ++i)
89 {
90     if (*argv[i] == 'f')
91     {
92         outputFileName = argv[i + 1];
93         break;
94     }
95     else
96     {
97         outputFileName = DEFAULT_OFFILE;
98     }
99 }
100
101 /* Generate balls and determine which box they go in. */
102 for (int i = 0; i < nBalls; ++i)
103 {
104     // Equilikely distribution between box 0 and box nBoxes - 1
105     int box = Equilikely(0, nBoxes - 1);
106
107     // Iterate through the linked list looking for the box number. If it is found, increment
108     // the count.
109     // If the box is not in the list, add it to the list and set the count to 1.
110     // The box is sorted in ascending order.
111     bool found = false;
112     for (auto it = boxCount.begin(); it != boxCount.end(); ++it)
113     {
114         // The box is in the list
115         if (it->first == box)
116         {
117             it->second++;
118             found = true;
119             break;
120         }
121
122         // The box is not in the list and a greater box value is found
123         else if (it->first > box)
124         {
125             boxCount.insert(it, std::make_pair(box, 1));
126             found = true;
127             break;
128         }
129     }

```

```

130 // The box is not in the list and is greater than all boxes currently in the list
131 if (!found)
132 {
133     boxCount.push_back(std::make_pair(box, 1));
134 }
135 }
136
137 // Traverse the linked list nBalls times and determine how many boxes contain n number of
138 // balls.
139 std::vector<long> boxCountVector;
140 for (int i = 0; i < nBalls; ++i)
141 {
142     long count = 0;
143     for (auto it = boxCount.begin(); it != boxCount.end(); ++it)
144     {
145         if (it->second == i)
146         {
147             count++;
148         }
149     }
150     boxCountVector.push_back(count);
151 }
152
153 // Remove trailing zeros
154 for (auto iter = boxCountVector.end() - 1; iter != boxCountVector.begin(); --iter)
155 {
156     if (*iter == 0)
157     {
158         boxCountVector.pop_back();
159     }
160     else
161     {
162         break;
163     }
164 }
165
166 // Set the number of boxes with zero balls to nBoxes subtracted by the sum of the number
167 // of boxes that have balls in them
168 long sum{ 0 };
169 for (auto iter = boxCountVector.begin(); iter != boxCountVector.end(); ++iter)
170 {
171     sum += *iter;
172 }
173 boxCountVector.at(0) = (nBoxes - sum);
174
175 // Store boxCountVector as a csv file for plotting in Python
176 std::ofstream outputFile;
177 outputFile.open(outputFileName);
178 long boxNum{ 0 };
179 for (auto iter = boxCountVector.begin(); iter != boxCountVector.end(); ++iter)
180 {
181     // Output the probability of a box containing n number of balls
182     outputFile << boxNum++ << ", " << (double) *iter / nBoxes << std::endl;
183 }
184 outputFile.close();
185
186 // Calculate the mean and standard deviation
187 double mean{ 0 };
188 double stdDev{ 0 };
189 boxNum = 0;
190
191 for (auto iter = boxCountVector.begin(); iter != boxCountVector.end(); ++iter)
192 {
193     mean += (double) *iter * boxNum / nBoxes;
194     ++boxNum;
195 }

```

```

193     }
194
195     boxNum = 0;
196
197     for (auto iter = boxCountVector.begin(); iter != boxCountVector.end(); ++iter)
198     {
199         stdDev += (double) * iter * pow((boxNum - mean), 2);
200         ++boxNum;
201     }
202
203     stdDev = sqrt(stdDev / nBoxes);
204
205     std::cout << "Mean: " << mean << std::endl;
206     std::cout << "Standard Deviation: " << stdDev << std::endl;
207
208     return 0;
209 }

```


3 Ex. 4.2.11

A test is compiled by selecting 12 different questions, at random and without replacement, from a well-published list of 120 questions. After studying this list you are able to classify all 120 questions into two classes, I and II. Class I Questions are those about which you feel confident; the remaining questions define class II. Assume that your grade probability conditioned on the class of the problem, is

	A	B	C	D	E
Class I	0.6	0.3	0.1	0.0	0.0
Class II	0.0	0.1	0.4	0.4	0.1

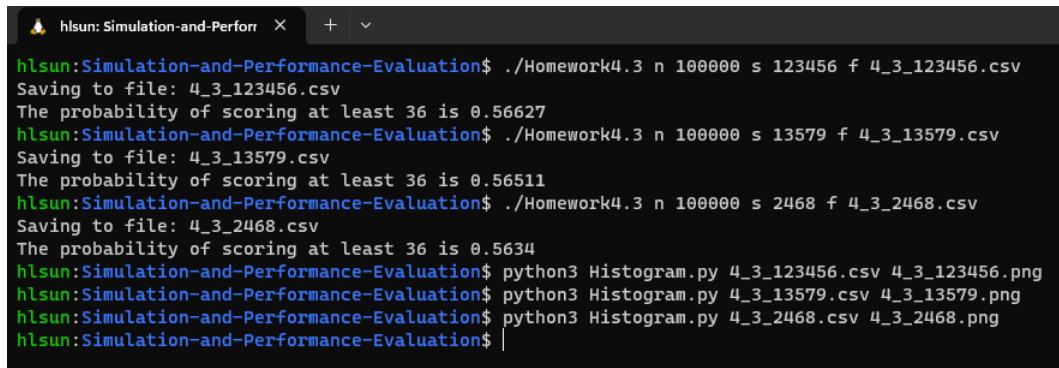
Each test question is graded on an A = 4, B = 3, C = 2, D = 1, F = 0 scale and a score of 36 or better is required to pass the test.

(a) If there are 90 Class I questions in the list, use Monte Carlo Simulation and 100,000 replications to generate a discrete-data histogram of scores.

(b) From this histogram, what is the probability that you will pass the test?

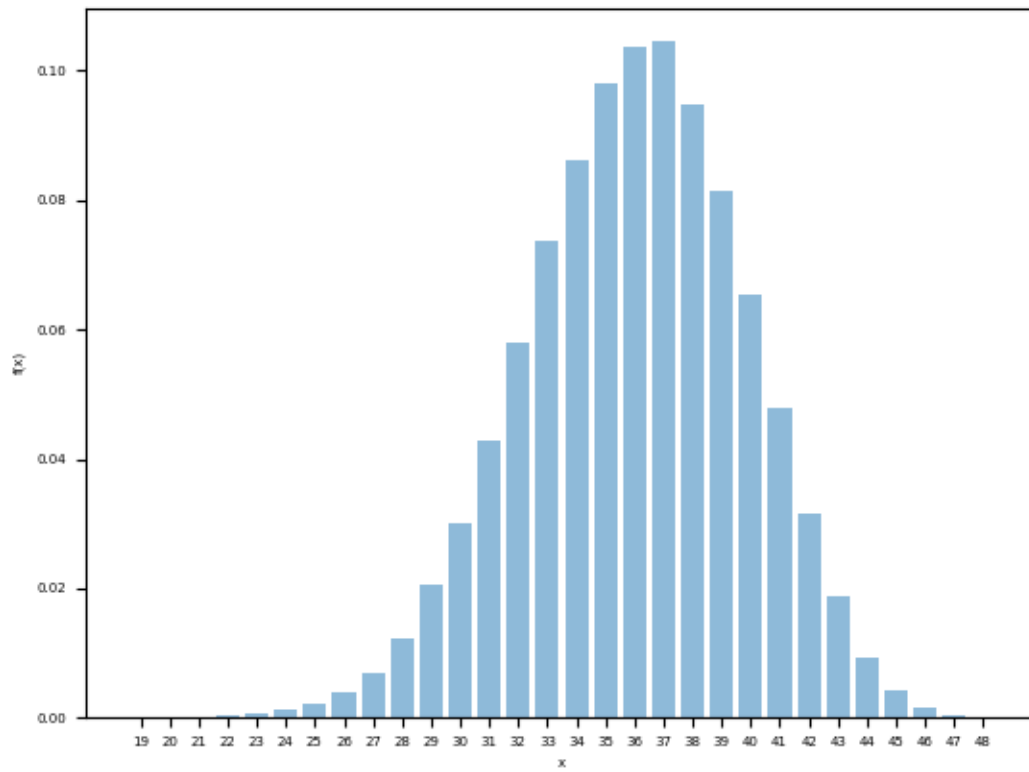
The probability of passing the test is approximately 56%.

Terminal Output:

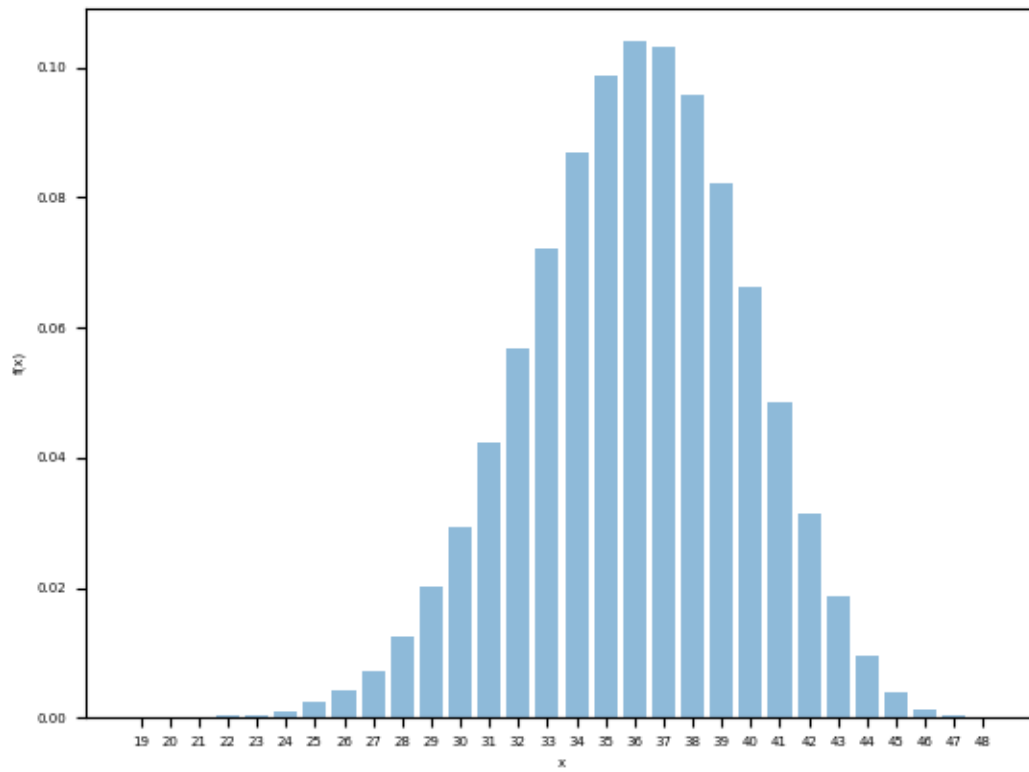
A terminal window titled 'hlsun: Simulation-and-Perfor' with three tabs. The first tab is active and shows the following commands and output:

```
hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.3 n 100000 s 123456 f 4_3_123456.csv
Saving to file: 4_3_123456.csv
The probability of scoring at least 36 is 0.56627
hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.3 n 100000 s 13579 f 4_3_13579.csv
Saving to file: 4_3_13579.csv
The probability of scoring at least 36 is 0.56511
hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.3 n 100000 s 2468 f 4_3_2468.csv
Saving to file: 4_3_2468.csv
The probability of scoring at least 36 is 0.5634
hlsun:Simulation-and-Performance-Evaluation$ python3 Histogram.py 4_3_123456.csv 4_3_123456.png
hlsun:Simulation-and-Performance-Evaluation$ python3 Histogram.py 4_3_13579.csv 4_3_13579.png
hlsun:Simulation-and-Performance-Evaluation$ python3 Histogram.py 4_3_2468.csv 4_3_2468.png
hlsun:Simulation-and-Performance-Evaluation$
```

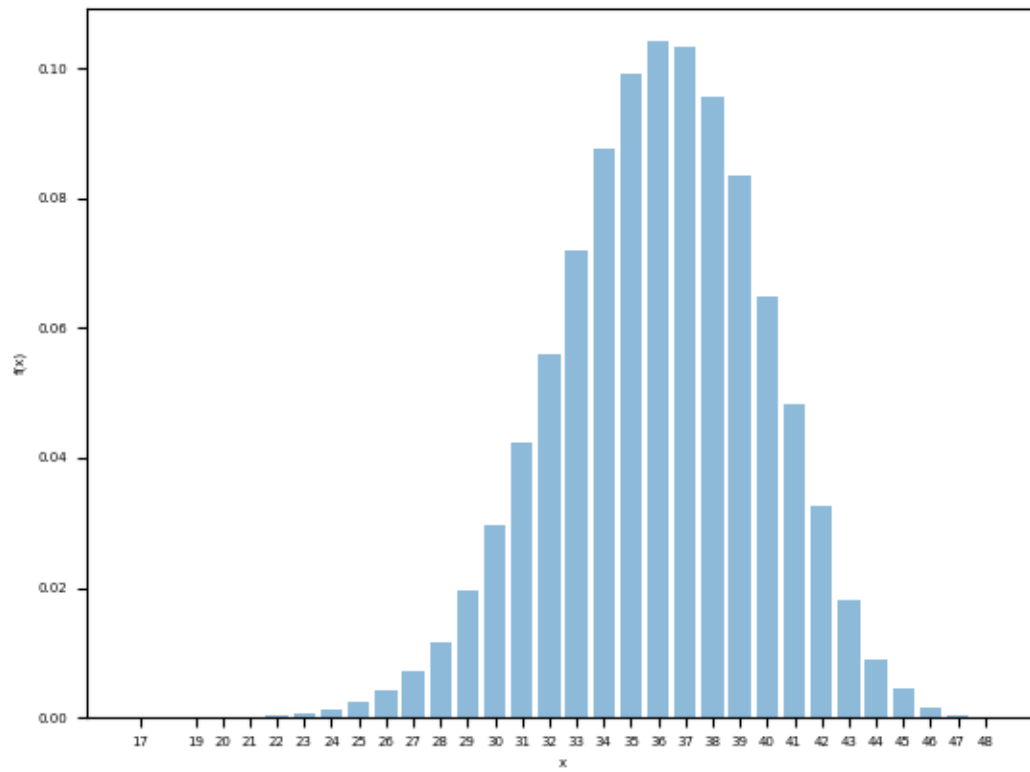
Seed: 2468



Seed: 13579



Seed: 123456



```

1  /**
2   * Harrison Sun (sun.har@northeastern.edu)
3   * EECE 5643 – Simulation and Performance Evaluation
4   * Homework 4.3
5   */
6
7  #define DEFAULT_SEED      12345
8  #define DEFAULT_N_RUNS    100000
9  #define DEFAULT_PASSING   36
10 #define DEFAULT_OFILE     "output.txt"
11
12 // Define the grade weights
13 #define WEIGHT_A          4
14 #define WEIGHT_B          3
15 #define WEIGHT_C          2
16 #define WEIGHT_D          1
17 #define WEIGHT_F          0
18
19 #include <cstdlib>
20 #include <cstring>
21 #include <fstream>
22 #include <stdio.h>
23 #include <exception>
24 #include <iostream>
25 #include <list>
26 #include <math.h>
27 #include <string>
28 #include <vector>
29 #include "c_lib/rvgs.h"
30 #include "c_lib/rngs.h"
31 #include "checkarg/checkarg.h"
32
33 /**
34  * int weight
35  *
36  * @param int n – The grade decision
37  * @param int A – The weight of A
38  * @param int B – The weight of B
39  * @param int C – The weight of C
40  * @param int D – The weight of D
41  * @param int F – The weight of F
42  *
43  * @return int grade – The weighted grade
44  */
45
46 int weight(int n, int A, int B, int C, int D, int F)
47 {
48     int grade = 0;
49     switch (n) {
50     case 0:
51         grade = A;
52         break;
53     case 1:
54         grade = B;
55         break;
56     case 2:
57         grade = C;
58         break;
59     case 3:
60         grade = D;
61         break;
62     case 4:
63         grade = F;
64         break;
65     }

```

```

66     return grade;
67 }
68
69 /**
70  * int main() - The main function
71  *
72  * @param int argc - the number of arguments
73  * @param char* argv[] - the arguments
74  *
75  * @return 0 if the program runs successfully
76  */
77
78 int main(int argc, char* argv[])
79 {
80     // There are 120 possible questions in the exam
81     const int num_q{ 120 };
82     // 90 Questions are classified as Class 1
83     const int num_c1{ 90 };
84     // 30 Questions are classified as Class 2
85     const int num_c2{ 30 };
86     // Doubly linked list to store the number of times each score is achieved <int score, long
      number of times scored>
87     std::list<std::pair<int, long>> scoreCount;
88
89     // Set the seed
90     for (int i = 0; i < argc; ++i)
91     {
92         if (*argv[i] == 's' && checkArg(argv[i + 1]))
93         {
94             PutSeed(std::stoi(argv[i + 1]));
95             break;
96         }
97         else
98         {
99             PutSeed(DEFAULT_SEED);
100         }
101     }
102
103     // Set the number of runs
104     int nRuns{};
105     for (int i = 0; i < argc; ++i)
106     {
107         if (*argv[i] == 'n' && checkArg(argv[i + 1]))
108         {
109             nRuns = std::stoi(argv[i + 1]);
110             break;
111         }
112         else
113         {
114             nRuns = DEFAULT_N_RUNS;
115         }
116     }
117
118     // Set the passing score
119     int passingScore{};
120     for (int i = 0; i < argc; ++i)
121     {
122         if (*argv[i] == 'p' && checkArg(argv[i + 1]))
123         {
124             passingScore = std::stoi(argv[i + 1]);
125             break;
126         }
127         else
128         {
129             passingScore = DEFAULT_PASSING;

```

```

130     }
131 }
132
133 // Set the grade weights
134 int weightA{};
135 int weightB{};
136 int weightC{};
137 int weightD{};
138 int weightF{};
139
140 for (int i = 0; i < argc; ++i)
141 {
142     if (*argv[i] == 'A' && checkArg(argv[i + 1]))
143     {
144         weightA = std::stoi(argv[i + 1]);
145         break;
146     }
147     else
148     {
149         weightA = WEIGHT_A;
150     }
151 }
152
153 for (int i = 0; i < argc; ++i)
154 {
155     if (*argv[i] == 'B' && checkArg(argv[i + 1]))
156     {
157         weightB = std::stoi(argv[i + 1]);
158         break;
159     }
160     else
161     {
162         weightB = WEIGHT_B;
163     }
164 }
165
166 for (int i = 0; i < argc; ++i)
167 {
168     if (*argv[i] == 'C' && checkArg(argv[i + 1]))
169     {
170         weightC = std::stoi(argv[i + 1]);
171         break;
172     }
173     else
174     {
175         weightC = WEIGHT_C;
176     }
177 }
178
179 for (int i = 0; i < argc; ++i)
180 {
181     if (*argv[i] == 'D' && checkArg(argv[i + 1]))
182     {
183         weightD = std::stoi(argv[i + 1]);
184         break;
185     }
186     else
187     {
188         weightD = WEIGHT_D;
189     }
190 }
191
192 for (int i = 0; i < argc; ++i)
193 {
194     if (*argv[i] == 'F' && checkArg(argv[i + 1]))

```

```

195     {
196         weightF = std::stoi(argv[i + 1]);
197         break;
198     }
199     else
200     {
201         weightF = WEIGHT_F;
202     }
203 }
204
205 // Set the filestream
206 std::string outputFileName{};
207 for (int i = 0; i < argc; ++i)
208 {
209     if (*argv[i] == 'f')
210     {
211         outputFileName = argv[i + 1];
212         break;
213     }
214     else
215     {
216         outputFileName = DEFAULT_OFFILE;
217     }
218 }
219
220 // Class conditional probability
221 const std::vector<double> class1Prob{ 0.6, 0.3, 0.1, 0.0, 0.0 };
222 const std::vector<double> class2Prob{ 0.0, 0.1, 0.4, 0.4, 0.1 };
223
224
225 // Run the simulation nRuns times
226 for (int n = 0; n < nRuns; ++n)
227 {
228     // Reset the number of questions to the default
229     int nq { num_q };
230     int nc1{ num_c1 };
231     int nc2{ num_c2 };
232
233     // Select 12 questions from the 120 questions without replacement
234     // Determine the class and value of each question
235     for (int i = 0; i < 12; ++i)
236     {
237         // Select a question
238         int question = Equilikely(1, nq--); /* Note to self: nq-- is used here to decrement nq
239 AFTER taking the Equilikely. --nq would not work the same way */
240         // Determine the class of the question
241         int questionClass = (question <= nc1) ? 1 : 2;
242         // Determine the value of the question based on the class conditional probability and
243 decrement the question class
244         double random01 = Uniform(0, 1);
245         // Store the cumulative value of the questions
246         static int cumulativeValue{ 0 };
247
248         if (questionClass == 1)
249         {
250             for (int j = 0; j < (int) class1Prob.size(); ++j)
251             {
252                 if (random01 < class1Prob[j])
253                 {
254                     cumulativeValue += weight(j, weightA, weightB, weightC, weightD, weightF);
255                     break;
256                 }
257                 else
258                 {
259                     // This adjusts the randomly generated Uniform(0, 1) to account for rejecting

```



```

258     the first comparison. The sum of the class conditional pdf is 1.0.
259         random01 -= class1Prob[j];
260     }
261     —nc1;
262 }
263
264 else // (questionClass == 2)
265 {
266     for (int j = 0; j < (int) class2Prob.size(); ++j)
267     {
268         if (random01 < class2Prob[j])
269         {
270             cumulativeValue += weight(j, weightA, weightB, weightC, weightD, weightF);
271             break;
272         }
273         else
274         {
275             // This adjusts the randomly generated Uniform(0, 1) to account for rejecting
the first comparison. The sum of the class conditional pdf is 1.0.
276             random01 -= class2Prob[j];
277         }
278     }
279     —nc2;
280 }
281 // For the final question, store the score in scoreCount
282 if (i == 11)
283 {
284     // Iterate through the linked list to find the correct scoreCount
285     // If the score is not in the list, add it to the list and set the count to 1.
286     // The score is sorted in ascending order.
287     bool found = false;
288     for (auto it = scoreCount.begin(); it != scoreCount.end(); ++it)
289     {
290         // The cumulativeValue is in the list
291         if (it->first == cumulativeValue)
292         {
293             it->second++;
294             found = true;
295             break;
296         }
297
298         // The box is not in the list and a greater cumulativeValue value is found
299         else if (it->first > cumulativeValue)
300         {
301             scoreCount.insert(it, std::make_pair(cumulativeValue, 1));
302             found = true;
303             break;
304         }
305     }
306     // The cumulativeValue is not in the list and is greater than all cumulativeValues
currently in the list
307     if (!found)
308     {
309         scoreCount.push_back(std::make_pair(cumulativeValue, 1));
310     }
311     // Reset the cumulativeValue to 0 for the next run
312     cumulativeValue = 0;
313 } // End if (i == 11)
314 } // End Question Selection
315 }
316 std::cout << "Saving to file: " << outputFileName << std::endl; /*
*****
317 // Store the contents of the linked list in csv format for plotting in Python
318 std::ofstream outputFile;

```

```

319 outputFile.open(outputFileName);
320 for (auto it = scoreCount.begin(); it != scoreCount.end(); ++it)
321 {
322     outputFile << it->first << "," << (double) it->second / (double) nRuns << std::endl;
323 }
324 outputFile.close();
325
326 // Determine whether pass or fail
327 long passCount{ 0 };
328 for (auto it = scoreCount.begin(); it != scoreCount.end(); ++it)
329 {
330     if (it->first >= passingScore)
331     {
332         passCount += it->second;
333     }
334 }
335
336 std::cout << "The probability of scoring at least " << passingScore << " is " << (double)
337     passCount / static_cast<double>(nRuns) << std::endl;
338 return 0;
339 }

```

4 Ex. 4.3.5

- (a) Construct a continuous-data histogram of the service times (in `ac.dat` - Exercise 1.2.6).
(b) Compare the histogram mean and standard deviation with the corresponding sample mean and standard deviation, and justify your choice of histogram parameters a , b , and either k or δ .

Terminal Output

```
(base) hlsun:Simulation-and-Performance-Evaluation$ make
g++ Homework4.4.cpp -o Homework4.4
(base) hlsun:Simulation-and-Performance-Evaluation$ ./Homework4.4 i ac.dat o Q4_4/4_4.csv
The sample mean: 3.03189 and sample standard deviation: 1.82446
The histogram mean: 2.384 and histogram standard deviation: 1.67415
(base) hlsun:Simulation-and-Performance-Evaluation$ python3 Histogram.py Q4_4/4_4.csv Q4_4/4_4.png
(base) hlsun:Simulation-and-Performance-Evaluation$
```

The sample mean: 3.03

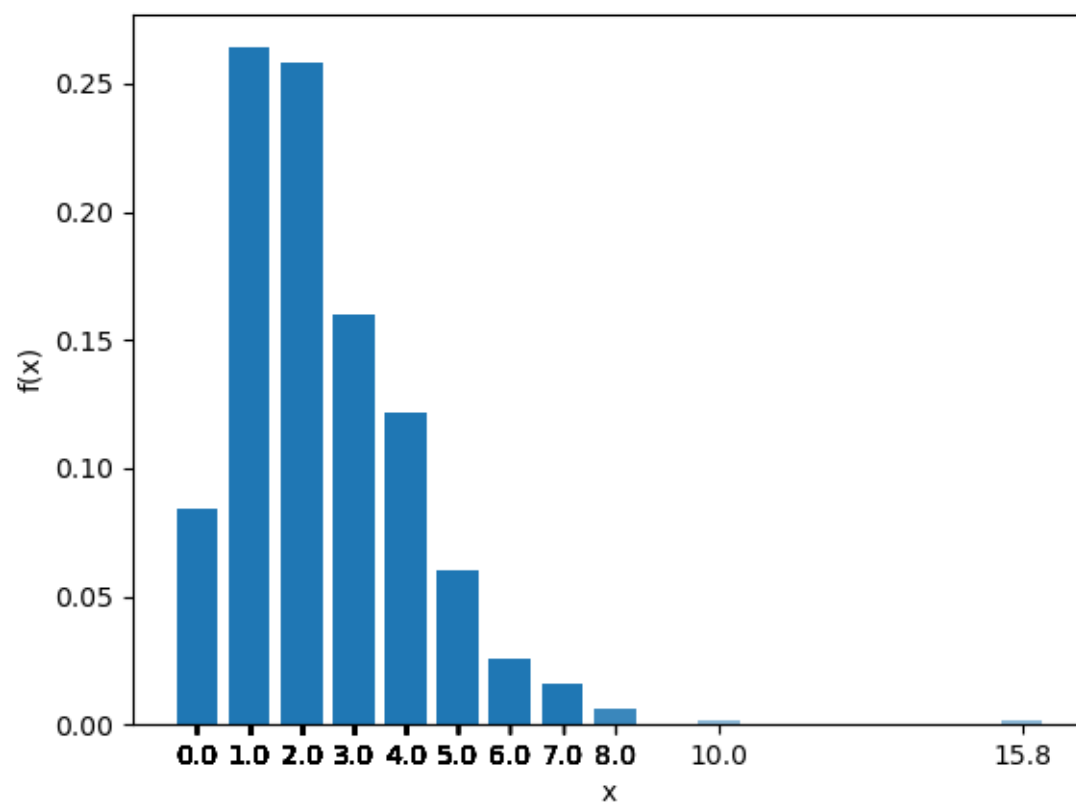
The sample standard deviation: 1.8

The histogram mean: 2.4

The histogram standard deviation: 1.7

We can see that the sample and histogram mean and standard deviation are different. Notably, the histogram statistics are lower than the sample statistics. This is because the histogram utilizes binning to classify the data. Since the bins are based off of the difference between the shortest and longest service times, a high valued maxima results in significantly skewed statistics, even if the outlier is uncommon. Given the finite number of bins and a large range, many values were put in bins below their actual values. This decreased the mean. The standard deviation is less impacted by this, as it is a measurement of scatter, but it is still impacted by the binning.

a and b were chosen to be the minimum and maximum values, as this allows the histogram to fit all of the data while also not allocating bins for data outside the range. The number of bins was chosen such that $k \in [\lceil \ln(n) \rceil, \lfloor \sqrt{n} \rfloor]$. Given that $n = 500$, $k \in [6, 22]$. 15 was chosen because more bins than the minimum were needed knowing that outlier data is significantly larger than the mean. This can be seen in the histogram below, where there are several empty bins before reaching the maximum bin. However, setting k too high, such as $k = 22$ would result in a noisy histogram. Therefore, it was decided that $k = 15$.



```

1  /**
2   * Harrison Sun (sun.har@northeastern.edu)
3   * EECE 5643 – Simulation and Performance Evaluation
4   * Homework 4.4
5   */
6
7  #define DEFAULT_IFILE    "ac.dat"
8  #define DEFAULT_OFILE    "output.txt"
9
10 #include <algorithm>
11 #include <cstdlib>
12 #include <cstring>
13 #include <fstream>
14 #include <stdio.h>
15 #include <exception>
16 #include <iostream>
17 #include <list>
18 #include <math.h>
19 #include <string>
20 #include <vector>
21 #include "c_lib/rvgs.h"
22 #include "c_lib/rngs.h"
23 #include "checkarg/checkarg.h"
24
25 /**
26  * int main() – The main function
27  *
28  * @param int argc – the number of arguments
29  * @param char* argv[] – the arguments
30  *
31  * @return 0 if the program runs successfully
32  *
33  * This is the main function. It reads in a data file in tab separated format. The first
34  * column contains the arrival times and the second column
35  * contains departure times.
36  *
37  */
38 int main(int argc, char* argv[])
39 {
40     // Create two vectors to store the data
41     std::vector<double> arrival_times{};
42     std::vector<double> departure_times{};
43     std::vector<double> service_times{};
44
45     // Histogram Parameters
46     int nHistogram { 500 }; // The number of jobs in the data file
47     // The number of bins in the histogram. Ideally,  $k \sim [\ln(n), \sqrt{n}] \Rightarrow$  Choosing from  $k \sim$ 
48     // [6, 22]
49
50     /*****
51
52     The number of bins is chosen to be 15. This is significantly larger than the minimum  $k =$ 
53     6. The reason for this choice is that the service times are empirical
54     data and may have significant outliers that would not fit well with other data when
55     given too few bins. However, 22 bins would be too many, as the data has
56     high variance and would thus have a lot of empty bins. 15 bins is a good compromise
57     between the two.
58
59     *****/
60
61     int nBins{15};
62
63     // File names
64     std::string inputFileName{};

```

```

61  std::string outputFileName{};
62
63  // Set the input file stream
64  for (int i = 0; i < argc; ++i)
65  {
66      if (*argv[i] == 'i')
67      {
68          inputFileName = argv[i + 1];
69          break;
70      }
71      else
72      {
73          inputFileName = DEFAULT_IFILE;
74      }
75  }
76
77  // Set the output file stream
78  for (int i = 0; i < argc; ++i)
79  {
80      if (*argv[i] == 'o')
81      {
82          outputFileName = argv[i + 1];
83          break;
84      }
85      else
86      {
87          outputFileName = DEFAULT_OFILE;
88      }
89  }
90
91  double arrival_time{};
92  double departure_time{};
93
94  std::ifstream infile;
95  infile.open(inputFileName.c_str());
96
97  // Read in the data from the tsv to the vectors
98  while (infile >> arrival_time >> departure_time)
99  {
100     arrival_times.push_back(arrival_time);
101     departure_times.push_back(departure_time);
102 }
103 infile.close();
104
105 // Calculate the service times and store them in the service_times vector
106 for (int i = 0; i < arrival_times.size(); ++i)
107 {
108     // Check if the service node is free at arrival
109     if (arrival_times[i] > departure_times[i - 1])
110     {
111         service_times.push_back(departure_times[i] - arrival_times[i]);
112     }
113     // If the job has to wait in a queue, the service starts after the previous job is
    finished
114     else
115     {
116         service_times.push_back(departure_times[i] - departure_times[i - 1]);
117     }
118 }
119
120 // Calculate the total service time
121 double total_service_time{};
122 for (std::vector<double>::iterator i = service_times.begin(); i != service_times.end(); ++
    i)
123 {

```

```

124     total_service_time += *i;
125 }
126
127 // Calculate the sample mean and standard deviation
128 double sample_mean{ total_service_time / service_times.size() };
129 double sample_std{};
130
131 // Standard Deviation
132 for (std::vector<double>::iterator i = service_times.begin(); i != service_times.end(); ++
133     i)
134 {
135     sample_std += pow(*i - sample_mean, 2);
136 }
137 sample_std = sqrt(sample_std / (service_times.size() - 1));
138
139 // Bin the data into nBins bins
140 double max_service_time = *std::max_element(service_times.begin(), service_times.end());
141 double min_service_time = *std::min_element(service_times.begin(), service_times.end());
142
143 // Calculate the bin width
144 double bin_width = (max_service_time - min_service_time) / nBins;
145
146 // Put the data in the bins
147 for (std::vector<double>::iterator i = service_times.begin(); i != service_times.end(); ++
148     i)
149 {
150     // Find the bin that the data point belongs to
151     for (int j = 0; j < nBins; ++j)
152     {
153         if (*i >= (double) j * bin_width && *i < (double) (j + 1) * bin_width)
154         {
155             *i = j;
156             break;
157         }
158     }
159 }
160
161 // Output the binned histogram data to a file
162 std::ofstream outfile;
163 outfile.open(outputFileName.c_str());
164
165 // Output the data as a histogram
166 for (std::vector<double>::iterator i = service_times.begin(); i != service_times.end() +
167     1; ++i)
168 {
169     outfile << *i << ", " << (double) std::count(service_times.begin(), service_times.end(),
170         *i) / service_times.size() << std::endl;
171 }
172 outfile.close();
173
174 // Calculate the histogram mean and standard deviation
175 double histogram_mean{};
176 double histogram_std{};
177
178 for (int i = 0; i < nBins; ++i)
179 {
180     histogram_mean += i * std::count(service_times.begin(), service_times.end(), i);
181 }
182 histogram_mean = histogram_mean / service_times.size();
183
184 for (int i = 0; i < nBins; ++i)
185 {
186     histogram_std += pow(i - histogram_mean, 2) * std::count(service_times.begin(),
187         service_times.end(), i);
188 }

```

```

184 histogram_std = sqrt(histogram_std / (service_times.size() - 1));
185
186 // Output the means and standard deviations
187 std::cout << "The sample mean: " << sample_mean << " and sample standard deviation: " <<
    sample_std << std::endl;
188 std::cout << "The histogram mean: " << histogram_mean << " and histogram standard
    deviation: " << histogram_std << std::endl;
189 return 0;
190 }

```