**Client**

**Data Structures Used:**

int\* tripleSumArrayPtr → new int [listSize]

This is an int pointer that is assigned memory for a dynamic array of integers for the TripleSum function to calculate Triple Sums that equate to 0.

**Function: main()**

time\_t totalTime

int\* tripleSumArrayPtr

int testListSize

prompt user for list size

read in list size

tripleSumArrayPtr = new int [testListSize]

CreateRandomNumbers(tripleSumArrayPtr, testListSize)

totalTime = TimeClock(tripleSumArrayPtr, testListSize)

output run-time for TripleSum function

delete[] tripleSumArrayPtr

return(0)

**Function:** int TripleSumFunction(int listSize, int\* sumArray)

Narrative: function implementation of the TripleSum algorithm

Pre-condition: list of integers is non-empty

Post-condition: a list of triples that count to zero has been calculated

int count = 0

int i = 0

int j = 0

int k = 0

while(i < listSize)

j = i + 1

while(j < listSize)

k = j + 1

while(k < listSize)

if(sumArray[i] + sumArray[j] + sumArray[k] == 0)

count += 1

endif

k += 1

endwhile

j += 1

endwhile

i += 1

endwhile

return(count)

**Function:** void CreateRandomNumbers(int\* sumArray, int& listSize)

Narrative: creates a random list of numbers of size N

Pre-condition: none

Post-condition: a list of size N is populated with random integers

int randomNumberGenerator

for(int i = 0; i < listSize; i++)

randomNumberGenerator = rand() % 20000 -10000

sumArray[i] = randomNumberGenerator

endfor

**Function:** time\_t TimeClock(int\* sumArray, int listSize)

Narrative: times the run-time of the TripleSum function

Pre-condition: none

Post-condition: run-time of the TripleSum function has been calculated

time\_t startTime, endTime

int totalTripSums

time\_t totalTime

time(&startTime)

totalTripSums = TripleSumFunction(listSize, sumArray)

time(&endTime)

totalTime = difftime(endTime, startTime)

return(totalTime)

**Test Client**

**Data Structures Used:**

int\* tripleSumArrayPtr → new int [listSize]

This is an int pointer that is assigned memory for a dynamic array of integers for the TripleSum function to calculate Triple Sums that equate to 0.

**Function: main()**

time\_t totalTime

int\* tripleSumArrayPtr

int testListSize

prompt user for list size

read in list size

tripleSumArrayPtr = new int [testListSize]

CreateRandomNumbers(tripleSumArrayPtr, testListSize)

totalTime = TimeClock(tripleSumArrayPtr, testListSize)

output run-time for TripleSum function

delete[] tripleSumArrayPtr

return(0)

**Function:** int TripleSumFunction(int listSize, int\* sumArray)

Narrative: function implementation of the TripleSum algorithm

Pre-condition: list of integers is non-empty

Post-condition: a list of triples that count to zero has been calculated

int count = 0

int i = 0

int j = 0

int k = 0

while(i < listSize)

j = i + 1

while(j < listSize)

k = j + 1

while(k < listSize)

if(sumArray[i] + sumArray[j] + sumArray[k] == 0)

count += 1

output numbers

output statement because triple sum is found

else

output numbers

output statement because triple sum not found

endif

k += 1

endwhile

j += 1

endwhile

i += 1

endwhile

return(count)

**Function:** void CreateRandomNumbers(int\* sumArray, int& listSize)

Narrative: creates a random list of numbers of size N

Pre-condition: none

Post-condition: a list of size N is populated with random integers

int randomNumberGenerator

for(int i = 0; i < listSize; i++)

randomNumberGenerator = rand() % 20000 -10000

sumArray[i] = randomNumberGenerator

output statement stating the random number created

endfor

**Function:** time\_t TimeClock(int\* sumArray, int listSize)

Narrative: times the run-time of the TripleSum function

Pre-condition: none

Post-condition: run-time of the TripleSum function has been calculated

time\_t startTime, endTime

int totalTripSums

time\_t totalTime

time(&startTime)

totalTripSums = TripleSumFunction(listSize, sumArray)

time(&endTime)

totalTime = difftime(endTime, startTime)

return(totalTime)

**Function Prototype: main()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| M1 | Successful program execution | Testing if the program can successfully run | testlistSize = 100, tripleSumArrayPtr set to new dynamic array of size 100 | Program executes successfully | Yes |
| M2 | Program exits because of an inconsistency | Testing if the program will exit because list is too large compared to allowable instructions | testListSize = 256,000 tripleSumArrayPtr not assigned because list is too big per specs | Program outputs error message because list is too large | Yes |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Function Prototype: int TripleSumFunction(int listSize, int\* sumArray)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| TSF1 | Triple sum is found | A triple sum has been found in the if statement | listSize = 100, tripleSumArray has 100 random integers within, output a statement when a triple sum is found | Statement is output because a triple sum has been found | Yes |
| TSF2 | Triple sum is not found | A triple sum has not been found in the if statement | listSize = 100, tripleSumArray has 100 random integers within, output a statement when a triple sum is not found | Statement is output because a triple sum is not found | Yes |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Function Prototype: void CreateRandomNumbers(int\* sumArray, int& listSize)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| CRN1 | Creating random numbers | Are random numbers being created between -10000 and 10000 | N/A | Array is populated with randomized values between -10000 and 10000, each number created is output to screen in a small list | Yes |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Function Prototype: time\_t TimeClock(int\* sumArray, int listSize)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
| TC1 | Timing the run of the TripleSum function | Is the time being correctly recorded for the TripleSum function | list is of size 2000 | Time taken is between 1 and 5 seconds, most likely closer to 1 second | Yes; however, time was 3 seconds, so mid-range |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Source Code**

Test Driver Code

/\*Bruce Conrad

//TripleSum Program

\*/

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <string>

#include <random>

using namespace std;

int TripleSumFunction(int listSize, int\* sumArray);

//Narrative: function implementation of the TripleSum algorithm

//Pre-condition: list of integers is non-empty

//Post-condition: a list of triples that count to zero has been calculated

void CreateRandomNumbers(int\* sumArray, int& listSize);

//Narrative: creates a random list of numbers of size N

//Pre-condition: none

//Post-condition: a list of size N is populated with random integers

time\_t TimeClock(int\* sumArray, int listSize);

//Narrative: times the run-time of the TripleSum function

//Pre-condition: none

//Post-condition: run-time of the TripleSum function has been calculated

int main()

{

time\_t totalTime;

int\* tripleSumArrayPtr;

int testListSize;

cout << "Enter the test list size: ";

cin >> testListSize;

if(testListSize <= 128000)

{

tripleSumArrayPtr = new int [testListSize];

CreateRandomNumbers(tripleSumArrayPtr, testListSize);

totalTime = TimeClock(tripleSumArrayPtr, testListSize);

cout << "TripleSum Runtime is: " << totalTime << " seconds" << endl;

delete[] tripleSumArrayPtr;

}

else

{

cout << "Invalid list size per Program specs--terminating program run!" << endl;

return(1);

}

return(0);

}

int TripleSumFunction(int listSize, int\* sumArray)

{

int count = 0;

int i = 0;

int j;

int k;

while(i < listSize)

{

j = i + 1;

while(j < listSize)

{

k = j + 1;

while(k < listSize)

{

if(sumArray[i] + sumArray[j] + sumArray[k] == 0)

{

count = count + 1;

cout << "Numbers: " << sumArray[i] << " ";

cout << sumArray[j] << " " << sumArray[k] << endl;

cout << "Triple Sum found!" << endl;

}

else

{

cout << "Numbers: " << sumArray[i] << " ";

cout << sumArray[j] << " " << sumArray[k] << endl;

cout << "Triple Sum not found!" << endl;

}

k = k + 1;

}

j = j + 1;

}

i = i + 1;

}

return(count);

}

void CreateRandomNumbers(int\* sumArray, int& listSize)

{

int randomNumberGenerator;

for(int i = 0; i < listSize; i++)

{

randomNumberGenerator = rand() % 20000 - 10000;

sumArray[i] = randomNumberGenerator;

cout << "Random number generated for slot " << i << " is ";

cout << sumArray[i] << endl;

}

}

time\_t TimeClock(int\* sumArray, int listSize)

{

time\_t startTime, endTime;

int totalTripSums;

time\_t totalTime;

time(&startTime);

totalTripSums = TripleSumFunction(listSize, sumArray);

time(&endTime);

cout << "Total number of Triple Sums: " << totalTripSums << endl;

totalTime = difftime(endTime, startTime);

return(totalTime);

}

Test Driver Bash Script

#!/bin/bash

echo "\*\*\*\*\*\*\*\*\*\*\*Compiling TripleSumTestDriver.cpp\*\*\*\*\*\*\*\*\*"

echo

c++ TripleSumTestDriver.cpp -o TripleSumTestDriver

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*Beginning runs of TripleSumTestDriver\*\*\*\*\*\*\*\*\*\*\*\*"

echo

./TripleSumTestDriver

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*"

echo

read answer

while [ $answer == "Yes" ]

do

./TripleSumTestDriver

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*"

echo

read answer

done

echo "\*\*\*\*\*\*\*\*\*\*Ending BASH Script!\*\*\*\*\*\*\*\*\*\*\*"

Main Code

/\*Bruce Conrad

//TripleSum Program

\*/

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <string>

#include <random>

using namespace std;

int TripleSumFunction(int listSize, int\* sumArray);

//Narrative: function implementation of the TripleSum algorithm

//Pre-condition: list of integers is non-empty

//Post-condition: a list of triples that count to zero has been calculated

void CreateRandomNumbers(int\* sumArray, int& listSize);

//Narrative: creates a random list of numbers of size N

//Pre-condition: none

//Post-condition: a list of size N is populated with random integers

time\_t TimeClock(int\* sumArray, int listSize);

//Narrative: times the run-time of the TripleSum function

//Pre-condition: none

//Post-condition: run-time of the TripleSum function has been calculated

int main()

{

time\_t totalTime;

int\* tripleSumArrayPtr;

int testListSize;

cout << "Enter the test list size: ";

cin >> testListSize;

if(testListSize <= 128000)

{

tripleSumArrayPtr = new int [testListSize];

CreateRandomNumbers(tripleSumArrayPtr, testListSize);

totalTime = TimeClock(tripleSumArrayPtr, testListSize);

cout << "TripleSum Runtime is: " << totalTime << " seconds" << endl;

delete[] tripleSumArrayPtr;

}

else

{

cout << "Invalid list size per Program specs--terminating program run!" << endl;

return(1);

}

return(0);

}

int TripleSumFunction(int listSize, int\* sumArray)

{

int count = 0;

int i = 0;

int j;

int k;

while(i < listSize)

{

j = i + 1;

while(j < listSize)

{

k = j + 1;

while(k < listSize)

{

if(sumArray[i] + sumArray[j] + sumArray[k] == 0)

{

count = count + 1;

}

k = k + 1;

}

j = j + 1;

}

i = i + 1;

}

return(count);

}

void CreateRandomNumbers(int\* sumArray, int& listSize)

{

int randomNumberGenerator;

for(int i = 0; i < listSize; i++)

{

randomNumberGenerator = rand() % 20000 - 10000;

sumArray[i] = randomNumberGenerator;

}

}

time\_t TimeClock(int\* sumArray, int listSize)

{

time\_t startTime, endTime;

int totalTripSums;

time\_t totalTime;

time(&startTime);

totalTripSums = TripleSumFunction(listSize, sumArray);

time(&endTime);

cout << "Total number of Triple Sums: " << totalTripSums << endl;

totalTime = difftime(endTime, startTime);

return(totalTime);

}

Main Bash Script

#!/bin/bash

echo "\*\*\*\*\*\*\*\*\*\*\*Compiling TripleSum.cpp\*\*\*\*\*\*\*\*\*"

echo

c++ TripleSum.cpp -o TripleSum

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*Beginning runs of TripleSum\*\*\*\*\*\*\*\*\*\*\*\*"

echo

./TripleSum

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*"

echo

read answer

while [ $answer == "Yes" ]

do

./TripleSum

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*"

echo

read answer

done

echo "\*\*\*\*\*\*\*\*\*\*Ending BASH Script!\*\*\*\*\*\*\*\*\*\*\*"

**Scripts**

Test Driver Script

(will take excerpts of test driver since full script is entirely too long to fit into the document)

Script started on 2019-10-16 22:31:19-0400

]777;notify;Command completed;exit]0;s789220c@cslab173:~/CSCI385/Prog3]7;file://cslab173/home/CS/s789220c/CSCI385/Prog3[22:31] s789220c@cslab173:~/CSCI385/Prog3 $ ./TripleSumTestBashScript.sh

\*\*\*\*\*\*\*\*\*\*\*Compiling TripleSumTestDriver.cpp\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Beginning runs of TripleSumTestDriver\*\*\*\*\*\*\*\*\*\*\*\*

Enter the test list size: 100

Random number generated for slot 0 is -617

Random number generated for slot 1 is 886

Random number generated for slot 2 is 2777

Random number generated for slot 3 is 6915

Random number generated for slot 4 is -2207

Random number generated for slot 5 is 8335

Random number generated for slot 6 is -4614

Random number generated for slot 7 is -9508

Random number generated for slot 8 is 6649

Random number generated for slot 9 is -8579

Random number generated for slot 10 is -7638

Random number generated for slot 11 is 27

Random number generated for slot 12 is -1310

Random number generated for slot 13 is -9941

Random number generated for slot 14 is 7763

Random number generated for slot 15 is 3926

Random number generated for slot 16 is -9460

Random number generated for slot 17 is -6574

Random number generated for slot 18 is -828

Random number generated for slot 19 is 5736

Random number generated for slot 20 is -4789

Random number generated for slot 21 is 5368

Random number generated for slot 22 is -7433

Random number generated for slot 23 is 6429

Random number generated for slot 24 is -4218

Random number generated for slot 25 is -8470

Random number generated for slot 26 is -7138

Random number generated for slot 27 is -4877

Random number generated for slot 28 is 4067

Random number generated for slot 29 is -6865

Random number generated for slot 30 is 3929

Random number generated for slot 31 is 9802

Random number generated for slot 32 is 4022

Random number generated for slot 33 is -6942

Random number generated for slot 34 is 3069

Random number generated for slot 35 is 8167

Random number generated for slot 36 is -8607

Random number generated for slot 37 is 8456

Random number generated for slot 38 is 5011

Random number generated for slot 39 is 8042

Random number generated for slot 40 is 6229

Random number generated for slot 41 is 7373

Random number generated for slot 42 is -5579

Random number generated for slot 43 is -5081

Random number generated for slot 44 is 3784

Random number generated for slot 45 is 8537

Random number generated for slot 46 is 5198

Random number generated for slot 47 is 4324

Random number generated for slot 48 is 8315

Random number generated for slot 49 is -5630

Random number generated for slot 50 is -3587

Random number generated for slot 51 is -6474

Random number generated for slot 52 is 6091

Random number generated for slot 53 is -1020

Random number generated for slot 54 is 9956

Random number generated for slot 55 is -8127

Random number generated for slot 56 is -3138

Random number generated for slot 57 is 9170

Random number generated for slot 58 is -3004

Random number generated for slot 59 is 7281

Random number generated for slot 60 is -7695

Random number generated for slot 61 is -9075

Random number generated for slot 62 is 7084

Random number generated for slot 63 is 6327

Random number generated for slot 64 is -9664

Random number generated for slot 65 is -3495

Random number generated for slot 66 is 846

Random number generated for slot 67 is -8271

Random number generated for slot 68 is -8687

Random number generated for slot 69 is -4143

Random number generated for slot 70 is 6124

Random number generated for slot 71 is 3895

Random number generated for slot 72 is 9582

Random number generated for slot 73 is -9455

Random number generated for slot 74 is 8814

Random number generated for slot 75 is 3367

Random number generated for slot 76 is 5434

Random number generated for slot 77 is 364

Random number generated for slot 78 is -5957

Random number generated for slot 79 is 3750

Random number generated for slot 80 is 1087

Random number generated for slot 81 is -3192

Random number generated for slot 82 is 7276

Random number generated for slot 83 is -2822

Random number generated for slot 84 is 5788

Random number generated for slot 85 is 3584

Random number generated for slot 86 is -4597

Random number generated for slot 87 is -7349

Random number generated for slot 88 is 2754

Random number generated for slot 89 is 2399

Random number generated for slot 90 is 9932

Random number generated for slot 91 is 5060

Random number generated for slot 92 is -324

Random number generated for slot 93 is 3368

Random number generated for slot 94 is -2261

Random number generated for slot 95 is 12

Random number generated for slot 96 is 6226

Random number generated for slot 97 is 8586

Random number generated for slot 98 is -1906

Random number generated for slot 99 is 7539

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Numbers: 6915 27 -6865

Triple Sum not found!

Numbers: 6915 27 3929

Triple Sum not found!

Numbers: 6915 27 9802

Triple Sum not found!

Numbers: 6915 27 4022

Triple Sum not found!

Numbers: 6915 27 -6942

Triple Sum found!

Numbers: 6915 27 3069

Triple Sum not found!

Numbers: 6915 27 8167

Triple Sum not found!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Total number of Triple Sums: 7

TripleSum Runtime is: 0 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*

Yes

Enter the test list size: 256000

Invalid list size per Program specs--terminating program run!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*

No

\*\*\*\*\*\*\*\*\*\*Ending BASH Script!\*\*\*\*\*\*\*\*\*\*\*

]777;notify;Command completed;./TripleSumTestBashScript.sh]0;s789220c@cslab173:~/CSCI385/Prog3]7;file://cslab173/home/CS/s789220c/CSCI385/Prog3[22:31] s789220c@cslab173:~/CSCI385/Prog3 $ exit

exit

Script done on 2019-10-16 22:31:43-0400

Main Script for Trial Runs

Script started on 2019-10-16 15:58:10-0400

]777;notify;Command completed;exit]0;s789220c@cslab171:~/CSCI385/Prog3]7;file://cslab171/home/CS/s789220c/CSCI385/Prog3[15:58] s789220c@cslab171:~/CSCI385/Prog3 $ ./TripleSumBashScript.sh

\*\*\*\*\*\*\*\*\*\*\*Compiling TripleSum.cpp\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Beginning runs of TripleSum\*\*\*\*\*\*\*\*\*\*\*\*

Enter the test list size: 500

Total number of Triple Sums: 766

TripleSum Runtime is: 0 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*

Yes

Enter the test list size: 1000

Total number of Triple Sums: 6307

TripleSum Runtime is: 1 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*

Yes

Enter the test list size: 2000

Total number of Triple Sums: 50350

TripleSum Runtime is: 3 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*

Yes

Enter the test list size: 4000

Total number of Triple Sums: 401188

TripleSum Runtime is: 20 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*

Yes

Enter the test list size: 8000

Total number of Triple Sums: 3199611

TripleSum Runtime is: 165 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*

Yes

Enter the test list size: 16000

Total number of Triple Sums: 25675917

TripleSum Runtime is: 1323 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Would you like to run the program again?\*\*\*\*\*\*\*\*\*\*

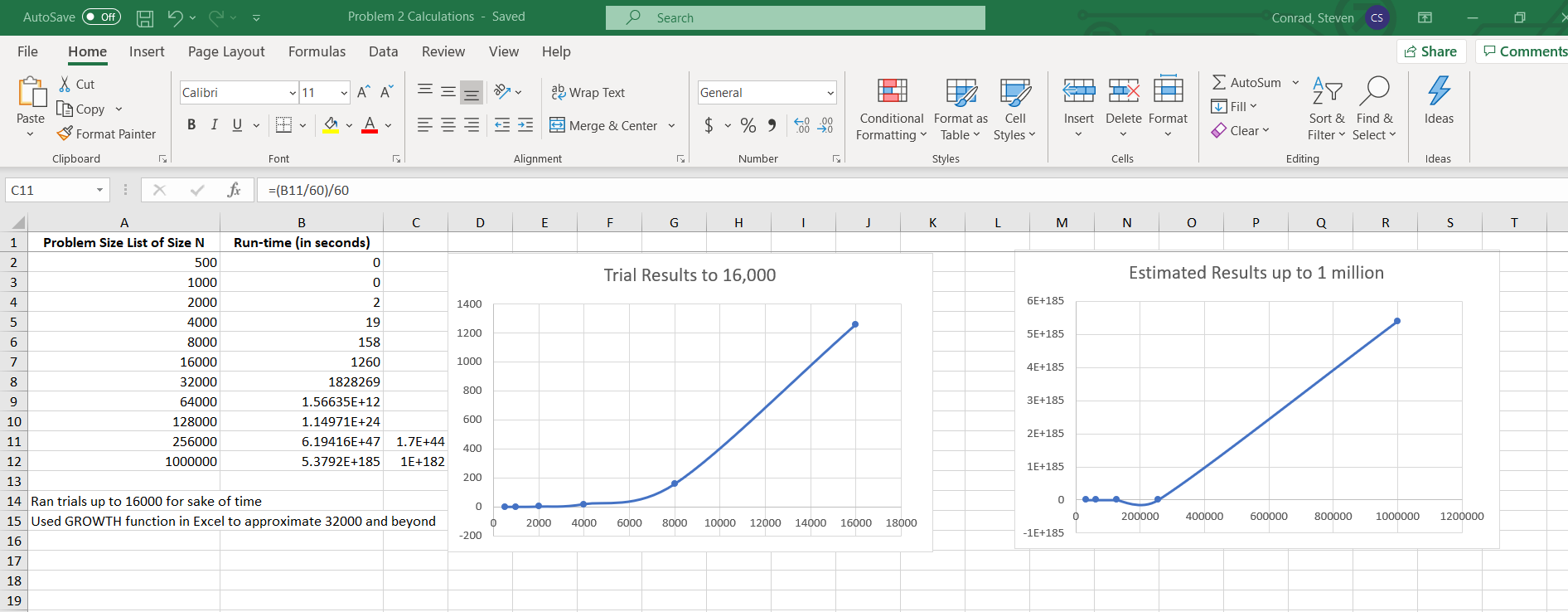
No

\*\*\*\*\*\*\*\*\*\*Ending BASH Script!\*\*\*\*\*\*\*\*\*\*\*

]777;notify;Command completed;./TripleSumBashScript.sh]0;s789220c@cslab171:~/CSCI385/Prog3]7;file://cslab171/home/CS/s789220c/CSCI385/Prog3[16:24] s789220c@cslab171:~/CSCI385/Prog3 $ exit

exit

Script done on 2019-10-16 16:24:20-0400



2) Based on the calculations I did in Excel, a list of size N = 256,000 would take approximately 1.7E+44 years to calculate the Triple Sums of this list. For N = 1,000,000, it is approximately 1E+182 years to calculate the Triple Sums of the list.

3) To randomly generate the negative random integers, I used the following formula in C++: rand() % 20,000 - 10,000. This way, it calculated a random integer between 0 and 20,000 and subtracted 10,000 from it, therefore creating random negative integers assuming the initial integer was below 10,000. It was necessary to perform the RNG calculations in this manner because rand() only creates a random number between 0 and 1, so by using the % to divide by 20,000 and get the remainder and then subtracting 10,000 from the number, it allowed the random numbers to be between -10,000 and 10,000 without going under or over respectively.

4) Given that the Merge Sort time complexity is O(nlogn) and that the Binary Search time complexity is O(logn), the PairSumFast algorithm is significantly faster than the TripleSum brute force approach. Let’s take a list of N distinct integers, say 1-10 in any order, the count starts at 0 at this point for pairs. First, the sort occurs, placing 1-10 in the proper order. Then, while i < N, if the binary search result is greater than the index, it increments the count. But, the only way for it to increment the count is if it finds a pairing in the array between the negative of a[i] and the array itself. For example, if the Binary Search looks at -1, and it finds 1 in the array, that is greater than the index (which starts at 0 for the sake of the array being completely searched). The index is then incremented in order to search the entire array. Now, given the run times of MergeSort and Binary Search, the following items are the asymptotics (upper-bound, lower-bound, tight-bound):

Worst-case run-time: O(nlogn) (because of Merge Sort)

By this logic,

O(N)🡪 nlogn

Ω(N)🡪 logn

Θ(N) 🡪 nlogn

based on the way the PairSumFast algorithm runs. The while loop will execute n times, while inside the loop the if statement has a function that executes at most logn times, therefore the asymptotics logically match to the upper, lower, and tight bounds as previously stated.

5)

**TripleSumFast** algorithm

cnt = 0

i = 0

while(i < N)

j = i + 1

while(j < N)

if(BinSearch(a[i] + a[j], a)

cnt = cnt + 1

endif

endwhile

endwhile

return cnt

This algorithm cuts down the worst-case run-time to O(N2). For a list of size N = 500, with randomly generated integers that are not distinct, it starts by indexing into the array for the first two integers. When it adds the two together for the BinSearch, this will run at most logn times, however, should it find the value of a[i] + a[j], it guarantees that there exists a Triple Sum in the list. Because the while loops run N2 times worst-case, but there is no third nested while loop, the run-time is significantly reduced as a result.

New worst-case run-time: O(N2)

By this logic,

O(N)🡪 n2

Ω(N)🡪 logn

Θ(N) 🡪 n2

because the worst-case run-time is O(N2). There is the BinSearch, which runs logn times, logn the lower bound, but the upper and tight bound are n2 based on the fact that the total run-time is O(N2) at worst.

6) By the nature of the question “Does the input contain a triple that sums to 0” this implies that the algorithm no longer needs to run through the input in full, but can instead stop once it finds a triple sum. The algorithm changes entirely because of the way the question is worded, and can therefore be rewritten as follows:

**New TripleSum Algorithm**

cnt = 0

i = 0

tripleSumFound = false

while(i < N && !tripleSumFound)

j = i + 1

while(j < N)

k = j + 1

while(k < N)

if(a[i] + a[j] + a[k] = 0)

tripleSumFound = true

endif

endwhile

endwhile

endwhile

return tripleSumFound

The algorithm now no longer needs to run through the entirety of the list, though it does not reduce worst-case run-time from O(N3). It significantly reduces the time taken to run the TripleSum program because it is looking for a triple sum in a general sense versus looking for every instance of a triple sum throughout the list. Because it can terminate at the first instance of a triple sum, the time in seconds for the program to run will be reduced, therefore reducing the necessary run-time of the program (although it is not guaranteed this will occur as there may not exist a Triple Sum in the list, though this case is unlikely).

7)

Given: N(N-1)(N-2)/6 is the number of different triples chosen from N items

Base Case:

Assume the given is true for N=3, then we have 3(3-1)(3-2)/6 triples from N items

This simplifies to 3(2)(1)/6 = 3!/6 = 6/6 = 1, which is true

Inductive step:

Now, assume this given is true for all N >= 3

For the given, it can be rewritten for N+1 terms

I want to show that the equation is as follows for N+1 terms: (N+1)(N)(N-1)/6

So we have the following:

(N+1)((N+1)-1)((N+1)-2)((N+1)-3)!/((N+1)-3)!\*6

=(N+1)(N)(N-1)(N-2)!/(N-2)!\*6

=(N+1)(N)(N-1)/6

Therefore, by induction, the original given is true for all lists of size N