Comp 230

Fall 2020

Nico de la Fuente & Dillan Lopez

Professor Jiang

10/13/20

Project 2

Table of Contents

[Assertion 3](#_Toc54799306)

[Introduction 3](#_Toc54799307)

[Algorithm Comparison 3](#_Toc54799308)

[Algorithm 1 3](#_Toc54799309)

[Algorithm 2 3](#_Toc54799310)

[Algorithm 3 4](#_Toc54799311)

[Algorithm 4 4](#_Toc54799312)

[Conclusion 4](#_Toc54799313)

[Figures 5](#_Toc54799314)

[Sample Run 8](#_Toc54799315)

[Source Code 9](#_Toc54799316)

# Assertion

Our programs have no issues. The code was submitted 3 hours late, and this report is being submitted two days after it was due. We apologize for the tardiness but hope that the quality and thoroughness of our work makes up for it.

# Introduction

Project 2 required us to read in a text or ASCII file that contained numbers separated by commas. We added additional functionality so that the user can also create a random sequence of a specified length.

There is also a hidden functionality that we used to generate the data we used for our algorithm analysis. It was useful because it stored the averages in a CSV file, allowing us to create more precise curves. To use it, type at the command line:

java max\_subsequence\_sum -t <sequence\_length> <iterations>

Our project has four “Max Subsequence Sum” algorithms. They are categorized by their run time efficiency into degree-3 polynomial, degree-2 polynomial, linear-logarithmic, and linear, respectively. The program allows the user to choose whether they want to run one of the algorithms individually, all of the algorithms at once, use a different sequence, or quit. If the user chooses to run an algorithm, the program calculates the average runtime for the algorithm(s) that were selected using 1000 iterations. It finally displays all the times to the corresponding algorithm on the screen. The program will run until the user chooses to quit.

# Algorithm Comparison

Algorithm 1 has a time complexity of O(n3). There are 3 nested loops that are used to generate the sum of every possible sequence in the list. After each sum is generated, it is compared to the maximum sum thus far and if it is greater, then it replaces the maximum sum.

Algorithm 2 has a time complexity of O(n2). There are 2 nested loops that are used to calculate the maximum sum of the sequence that starts with every element. This reduces computation time by keeping track of the maximum sum while going through the second nested loop.

Algorithm 3 has a time complexity of O(n log(n)). This algorithm recursively checks each half of the sequence to find the maximum sum and chooses the maximum value of both the halves as well as the maximum sum of a sequence that includes the middle element.

Algorithm 4 has a time complexity of O(n). There is only 1 loop that passes through each element of the sequence calculating a local sum and keeping track of the maximum value of those sums. If an addition of an element causes the local sum to be larger than that max, it replaces the max.

# Conclusion

Figure 0a displays the average run time of each algorithm when performed on sequences with lengths 1 to 500. Algorithm 4 was shown to be the most efficient. Algorithms 2 and 3 are less efficient, but Algorithm 3 in particular likely creates too much overhead due to its recursive nature. At the end of the line is Algorithm 1. Figure 0b clarifies some of the uncertainty on the lower end of Figure 0a. With a much larger sample size, it shows that even at smaller sample sizes the efficiency of the algorithms remains the same. Figures 1-4 show the lines of best fit. The lines of best fit closely matched the time-complexity that was found from the algorithm analysis. Overall, Algorithm 1 was the most efficient, and with such a simple implementation, there is no reason to use any other.

# Figures

Figure 0a. A comparison of the runtime curves for each of the 4 algorithms. Run times were calculated by average of 1000 iterations. The y-axis scale is logarithmic. The curves have several instances where the run time spikes suddenly. This is most likely due to environmental conditions such as CPU load or the random distribution of the sequences. With a larger sample size, we expect that the curves would smooth out, but we only performed 1000 iterations for each sequence length. The data for n < 50 is unclear.

Figure 0b. A comparison of the runtime curves for each of the 4 algorithms for sequence lengths less than 50. Run times were calculated by average of 1 million iterations. The y-axis scale is logarithmic

Figure 1. The runtime curve for algorithm 1. The line of best fit is a degree 3 polynomial.

Figure 2. The runtime curve for algorithm 1. The line of best fit is a degree 3 polynomial.

Figure 3. The runtime curve for algorithm 3. The line of best fit should be linear-logarithmic, however the graphing software did not support this

Figure 4. The runtime curve for algorithm 4. The line of best fit is linear.

# Sample Run

Welcome to the MSS algorithm evaluator

Sequence source menu

-- -- -- -- -- -- -- -- -- -- -- -- -- --

| 1) Read in the sequence from a file |

| 2) Create a random sequence |

-- -- -- -- -- -- -- -- -- -- -- -- -- --

Enter the number of your choice: nums.in

Invalid input. Input must be an integer between 1 and 2

Enter the number of your choice: 10000

Invalid input. Input must be an integer between 1 and 2

Enter the number of your choice: nums.in

Invalid input. Input must be an integer between 1 and 2

Enter the number of your choice: 1

Please enter the input file name: nums.in

Main Menu

-- -- -- -- -- -- -- -- -- -- --

| 0) Quit |

| 1) Test algorithm 1 O(n^3) |

| 2) Test algorithm 2 O(n^2) |

| 3) Test algorithm 3 O(nln(n)) |

| 4) Test algorithm 4 O(n) |

| 5) Test all 4 algorithms |

| 6) Use a different sequence |

-- -- -- -- -- -- -- -- -- -- --

Enter the number of your choice: 2

Testing algorithm 2 with 100 iterations

Average run time (seconds) for algorithm 2: 1.035e-06

Main Menu

-- -- -- -- -- -- -- -- -- -- --

| 0) Quit |

| 1) Test algorithm 1 O(n^3) |

| 2) Test algorithm 2 O(n^2) |

| 3) Test algorithm 3 O(nln(n)) |

| 4) Test algorithm 4 O(n) |

| 5) Test all 4 algorithms |

| 6) Use a different sequence |

-- -- -- -- -- -- -- -- -- -- --

Enter the number of your choice: q

Invalid input. Input must be an integer between 0 and 6

Enter the number of your choice: 0

# Source Code

import java.util.Random;

import java.util.Scanner;

import java.util.StringTokenizer;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.FileWriter;

import java.io.PrintWriter;

import java.io.IOException;

import java.io.FileNotFoundException;

class max\_subsequence\_sum {

public static void main(String[] args) {

// Default usage

if (args.length == 0) {

System.out.println("\n\nWelcome to the MSS algorithm evaluator\n");

// Propogate the sequence

int[] numList = getSequence();

while (true) {

int option = getOption2();

// Quit

if (option == 0) { break; }

// Test a specific algorithm

if (option <= 4) {

System.out.printf("Testing algorithm %d with 100 iterations\n", option);

testAlgorithm(numList, numList.length, 100, option);

}

// Test all 4 algorithms

else if (option == 5) {

System.out.println("Testing each algorithm with 100 iterations");

testAlgorithms(numList, numList.length, 100);

}

// Get a new sequence

else if (option == 6) {

numList = getSequence();

}

}

}

// Testing protocol

else if (args.length == 3 && args[0].equals("-t")) {

runTestingProtocol(args);

}

// Invalid usage

else printUsage();

}

/\*\*

\* Test the run time of a single algorithm

\*

\* @param arr The sequence

\* @param n The length of the sequence

\* @param iter The number of iterations to average

\* @param algoNum The number of the algorithm

\*/

public static void testAlgorithm(int arr[], int n, int iter, int algoNum) {

double totalRunTime = 0;

for (int i = 0; i < iter; i++) {

totalRunTime += getRuntime(algoNum, arr, n);

}

double averageRunTime = totalRunTime / (iter \* 1E9);

System.out.printf("Average run time (seconds) for algorithm %d: %6.3e\n", algoNum, averageRunTime);

}

/\*\*

\* Test the run times of each maximum subsequence sum algorithm

\*

\* @param arr The array to be tested on

\* @param n The length of the array

\* @param iter The number of iterations to perform

\*/

public static void testAlgorithms(int arr[], int n, int iter) {

double totalRunTimes[] = {0, 0, 0, 0};

// Test the run time for each algorithm

for (int i = 0; i < iter; i++) {

for (int algo = 1; algo <= 4; algo++) {

long runTime = getRuntime(algo, arr, n);

totalRunTimes[algo-1] += runTime;

}

}

// Calculate and print the average run time for each algorithm

System.out.printf("Average run times (seconds) for n = %d\n", n);

for (int algo = 0; algo < 4; algo++) {

double averageRunTime = totalRunTimes[algo] / (iter \* 1E9);

System.out.printf("Algorithm %d: %6.3e\n", algo+1, averageRunTime);

}

}

/\*\*

\* Get the run time for a specific algorithm

\*

\* @param algoNum The algorithm number

\* @param arr The array to run the algorithm on

\* @param n The length of the array

\* @return The run time

\*/

public static long getRuntime(int algoNum, int arr[], int n) {

// Get the current system time in nano seconds

long startTime = System.nanoTime();

switch (algoNum) {

case 1:

algorithm1(arr);

break;

case 2:

algorithm2(arr);

break;

case 3:

algorithm3(arr, 0, n-1);

break;

case 4:

algorithm4(arr);

break;

}

// Calculate and return the run time for the algorithm

return System.nanoTime() - startTime;

}

/\*\*

\* An algorithm to calculate the maximum subsequence sum in O(n^3) time

\*

\* @param arr The array

\* @return The maximum subsequence sum

\*/

public static int algorithm1(int arr[]) {

int maxSum = 0;

int n = arr.length;

// Pass over each element in the array

for(int i = 0; i < n; i++){

// Pass over each element from i to the end

for(int j = i; j < n; j++){

// Calculate the sum of the elements from i to j

int sum = 0;

for(int k = i; k < j+1; k++){

sum += arr[k];

}

// If sum is greater than maxSum, update maxSum

if (sum > maxSum)

maxSum = sum;

}

}

return maxSum;

}

/\*\*

\* An algorithm to calculate the maximum subsequence sum in O(n^2) time

\*

\* @param arr The array

\* @return The maximum subsequence sum

\*/

public static int algorithm2(int arr[]){

int maxSum = 0, n = arr.length;

// Pass over each element in the array

for(int i = 0; i < n; i++) {

// Calculate the sum of the elements from i to the end

int sum = 0;

for(int j = i; j <n; j++){

sum += arr[j];

// If sum is greater than maxSum, update maxSum

if(sum > maxSum){

maxSum = sum;

}

}

}

return maxSum;

}

/\*\*

\* A helper function for algorithm 3

\*

\* @param arr The array

\* @return The maximum subsequence sum

\*/

static int algorithm3(int arr[]) {

return algorithm3(arr, 0, arr.length - 1);

}

/\*\*

\* An algorithm to calculate the maximum subsequence sum in O(nlog(n)) time

\*

\* @param arr The array

\* @param start The starting position

\* @param stop The ending position

\* @return The maximum subsequence sum

\*/

static int algorithm3(int arr[], int start, int stop) {

// Base Case: Only one element

if (start == stop)

return arr[start];

// Find middle point

int m = (start + stop)/2;

// Return the max of the left and right halves and the max bounded sum

return Math.max(Math.max(algorithm3(arr, start, m),

algorithm3(arr, m+1, stop)),

maxBoundedSum(arr, start, m, stop));

}

/\*\*

\* Calculate the maximum subsequence sum that includes position m

\*

\* @param arr The array

\* @param start The starting position

\* @param m The middle

\* @param stop The stopping position

\* @return The max bounded sum

\*/

static int maxBoundedSum(int arr[], int start, int m, int stop) {

// Check the elements on left of mid.

int sum = 0;

int left\_sum = 0;

for (int i = m; i >= start; i--)

{

sum = sum + arr[i];

if (sum > left\_sum)

left\_sum = sum;

}

// Check the elements on right of mid

sum = 0;

int right\_sum = 0;

for (int i = m + 1; i <= stop; i++)

{

sum = sum + arr[i];

if (sum > right\_sum)

right\_sum = sum;

}

// Return sum of elements on left and right of mid

return Math.max(left\_sum + right\_sum, Math.max(left\_sum, right\_sum));

}

/\*\*

\* An algorithm to calculate the maximum subsequence sum in O(n) time

\*

\* @param arr The array

\* @return The maximum subsequence sum

\*/

public static int algorithm4(int arr[]) {

int maxSum = 0, sum = 0, n = arr.length;

// Pass over each element in the årray

for (int j = 0; j < n; j++) {

// Add the element to sum

sum += arr[j];

// If sum is greater than maxSum, update maxSum

if (sum > maxSum)

maxSum = sum;

// Else reset sum to 0

else if (sum < 0)

sum = 0;

}

return maxSum;

}

public static int[] getSequence() {

int numList[] = null;

int option = getOption1();

if (option == 0) {

// Read in the list of numbers

while (numList == null) {

numList = readNums(); // returns null if there was an error

}

}

else {

numList = randomizeNums(option, -100, 100);

}

return numList;

}

/\*\*

\* Read in a list of numbers from a file that is specified by the user

\*

\* @return The array of numbers or null if there was an error

\*/

public static int[] readNums() {

try {

// Read in the file name

Scanner scan = new Scanner(System.in);

System.out.print("Please enter the input file name: ");

String filename = scan.nextLine();

// Open the input file and split it into tokens

BufferedReader reader = new BufferedReader(new FileReader(filename));

StringTokenizer tokenizer = new StringTokenizer(reader.readLine(), ",");

// Convert each token into an integer and add it to the array

int len = tokenizer.countTokens();

int numList[] = new int[len];

for (int i = 0; i < len; i++) {

numList[i] = Integer.valueOf(tokenizer.nextToken());

}

return numList;

}

catch (FileNotFoundException ex) {

System.out.println(ex.getMessage());

}

catch (IOException ex) {

System.out.println(ex.getMessage());

}

catch (NumberFormatException ex) {

System.out.println(ex.getMessage());

}

return null;

}

/\*\*

\* Create a randomized list of integers

\*

\* @param n The length of the list

\* @param min The minimum value for the integers

\* @param max The maximum value for the integers

\* @return The list

\*/

public static int[] randomizeNums(int n, int min, int max) {

int arr[] = new int[n];

Random rand = new Random();

for (int i = 0; i < n; i++) {

arr[i] = rand.nextInt() % (max - min + 1) + min;

}

return arr;

}

/\*\*

\* Get the user's choice for where to get the sequence from

\* @return 0 for file input, or the length of the random sequence

\*/

public static int getOption1() {

System.out.println("\n Sequence source menu ");

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

System.out.println("| 1) Read in the sequence from a file |");

System.out.println("| 2) Create a random sequence |");

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

// Read in the option and ensure that it is valid

int option = getInput("Enter the number of your choice: ", 1, 2);

if (option == 1) return 0;

System.out.println("The random integer values will range from -100 to 100");

return getInput("Enter the desired length of the sequence: ", 0, Integer.MAX\_VALUE);

}

/\*\*

\* Get the user's choice for which algorithms to run

\* @return The user's choice

\*/

public static int getOption2() {

int OPT\_MIN = 0, OPT\_MAX = 6;

System.out.println("\n Main Menu ");

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

System.out.println("| 0) Quit |");

System.out.println("| 1) Test algorithm 1 O(n^3) |");

System.out.println("| 2) Test algorithm 2 O(n^2) |");

System.out.println("| 3) Test algorithm 3 O(nln(n)) |");

System.out.println("| 4) Test algorithm 4 O(n) |");

System.out.println("| 5) Test all 4 algorithms |");

System.out.println("| 6) Use a different sequence |");

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

// Read in the option and ensure that it is valid

return getInput("Enter the number of your choice: ", OPT\_MIN, OPT\_MAX);

}

/\*\*

\* Get an integer input from the user, reprompt if outside of range

\*

\* @param msg The prompt

\* @param min The minimum input value

\* @param max The maximum input value

\* @return The input

\*/

public static int getInput(String msg, int min, int max) {

try {

Scanner scan = new Scanner(System.in);

// Get the input and eprompt the user if it is out of the range

int input = min - 1;

while (input < min || input > max) {

// Print the message

System.out.print(msg);

try {

input = Integer.valueOf(scan.nextLine());

if (input < min || input > max) { throw new NumberFormatException(); }

System.out.println();

}

catch (NumberFormatException ex) {

System.out.printf("Invalid input. Input must be an integer between %d and %d\n\n", min, max);

input = min - 1;

}

}

return input;

}

catch (NumberFormatException ex) {

return min - 1;

}

}

/\*\*

\* Asks the user if they want to run the program again

\*

\* @return true if they do, false if they do not

\*/

public static boolean runAgain() {

// Ask the user if they want to run again

Scanner scan = new Scanner(System.in);

System.out.print("Do you want to run the program again (y/n): ");

String runProgram = scan.nextLine();

// Validate the user input

while ( !runProgram.equalsIgnoreCase("y") && !runProgram.equalsIgnoreCase("n") )

{

System.out.println("Invalid input.");

System.out.print("Do you want to run the program again (y/n): ");

runProgram = scan.nextLine();

}

return runProgram.equalsIgnoreCase("y");

}

/\*\*

\* Run the testing protocol

\* @param args The arguments

\*/

public static void runTestingProtocol(String args[]) {

try {

int n = Integer.valueOf(args[1]);

int iter = Integer.valueOf(args[2]);

PrintWriter outFile = new PrintWriter(new FileWriter("run\_times.txt"));

for (int len = 1; len <= n; len++) {

int testList[] = randomizeNums(len, -100, 100);

// Test the run time for each algorithm and write it to the output file{

long totalRunTime[] = {0, 0, 0, 0};

for (int i = 0; i < iter; i++) {

for (int algo = 1; algo <= 4; algo++) {

totalRunTime[algo - 1] += getRuntime(algo, testList, len);

}

}

outFile.printf("%d", len);

for (int algo = 0; algo < 4; algo++) {

outFile.print(",");

double avgRunTime = (double) totalRunTime[algo] / iter;

outFile.printf("%.4f", avgRunTime);

}

outFile.print("\n");

}

System.out.println("\nThe full data set can be found in run\_times.txt");

System.out.println("The format is: n,i,algo1,algo2,algo3,algo4\n");

outFile.close();

}

catch (NumberFormatException ex) {

printUsage();

}

catch (IOException ex) {

System.out.println(ex.getMessage());

System.exit(1);

}

}

/\*\*

\* Print the correct usage of the program

\*/

public static void printUsage() {

System.out.println("Usage: java max\_subsequence\_sum [-t <n> <iterations>]");

System.exit(1);

}

}