CS 2420 Honors Contract Runtime Analysis

Carl Anderson

April 17, 2021

Introduction

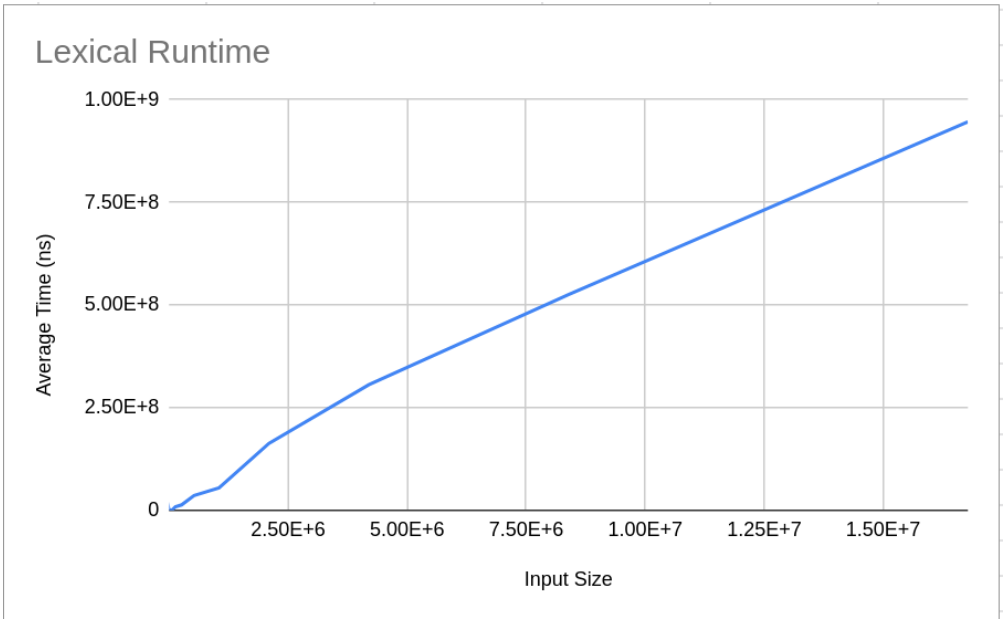
Leetcode is a popular coding practice website. It contains a database of many coding questions aiming to test a programmer's ability to identify a problem and create an algorithm solving it.
This report goes over 12 Leetcode questions, my solutions for them, an Oh-time analysis for those solutions, and a graph of the program's performance as input increases. The performance takes an average time from 10 trials using a random string or a random array of integers as its input.

Lexicographical Numbers

Problem: *Given an integer n, return all the numbers in the range [1, n] sorted in lexicographical order.*

Solution: The *lexicalOrder* method accepts an integer n as its maximum number. It then runs a loop for each digit. It then adds every number beginning with that digit, increasing by powers of 10 until it goes above n.

Oh-time analysis: *lexicalOrder* has a total of 3 nested loops. However, because the inner loops break whenever it has passed the maximum number, no number is reviewed twice. Because of this, the method's runtime is linear.

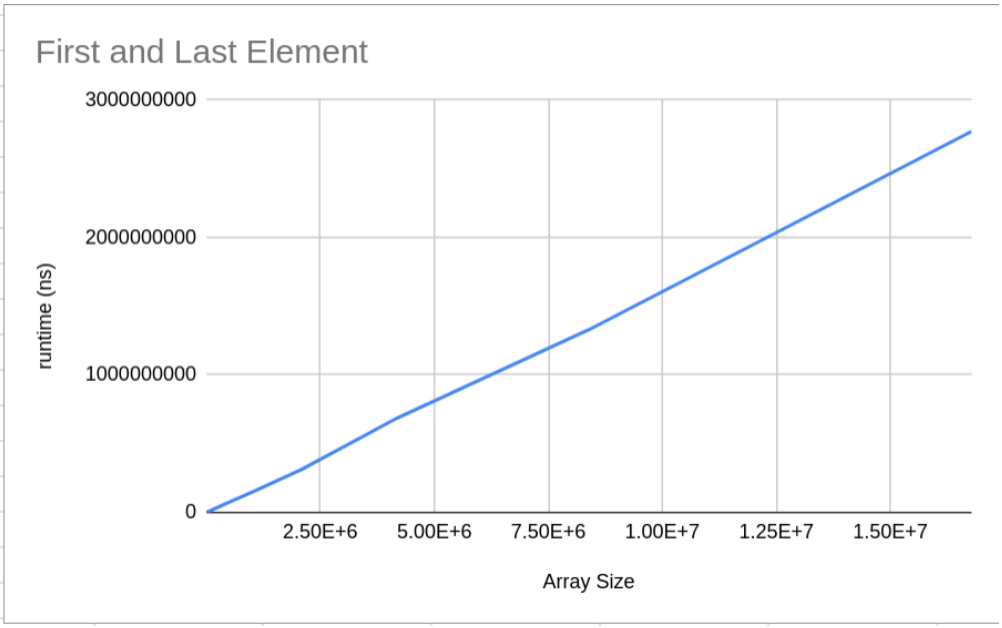


Find First and Last Position in a Sorted Array

Problem: *Given an array of integers nums sorted in ascending order, find the starting and ending position of a given target value*

Solution: *searchRange* method accepts an int array and a target. I altered the method's purpose slightly: rather than assuming an ordered array, it begins by doing a merge sort of the array. It then runs a binary search on the array to find the place immediately prior to the first instance of the target number, and then a binary search to find the index immediately after the last instance of the target number. It returns these two numbers in an array of size two.

Oh-time analysis: The method runs a merge sort and two binary searches. Because the merge sort has a $N \log N$ runtime, the runtime of our method is $N Log N$. Without it, we may suppose a $\log N$ runtime.

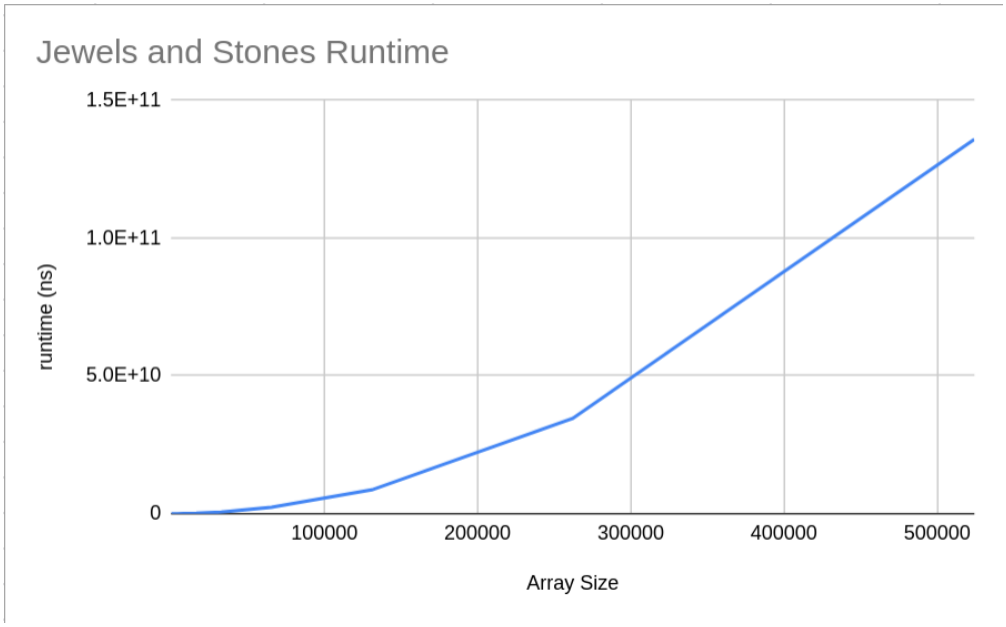


Number of jewels and stones

Problem: *You're given strings jewels representing the types of stones that are jewels, and stones representing the stones you have. Each character in stones is a type of stone you have. You want to know how many of the stones you have are also jewels.*

Solution: the method *numOfJewelsInStones* accepts two strings. It then runs two nested loops: the first goes through each character in the first string, and the second loop compares that character to every character in the second string. If it finds a match, it increases a counter. After it has exhausted every possibility, it returns the count.

Oh-time analysis: Because the method checks every character in the inner string for every character in the outer string, this method has an N^2 runtime.

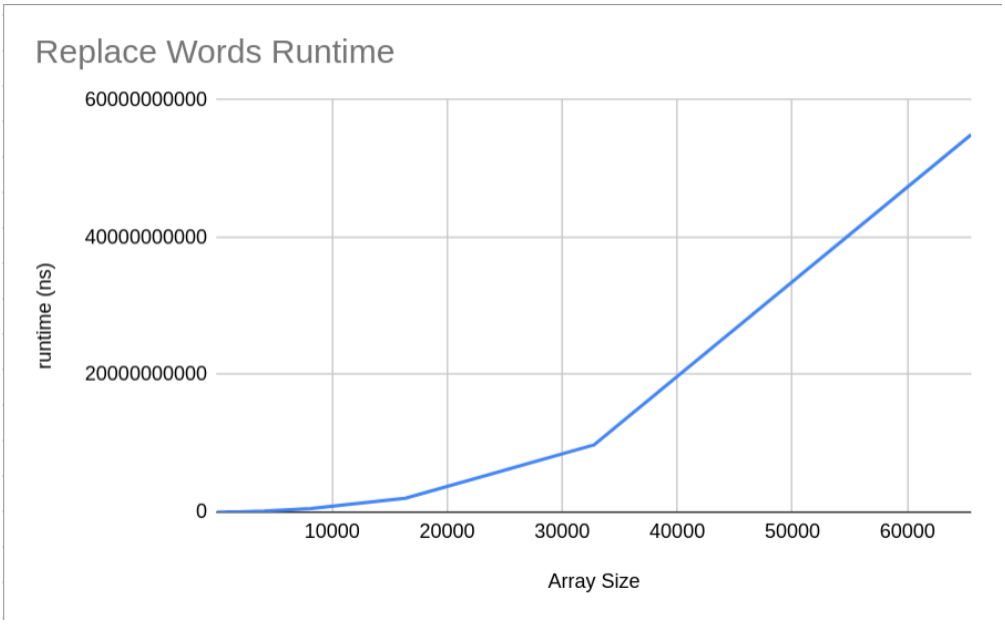


Replace Words

Problem: *Given a dictionary consisting of many roots and a sentence consisting of words separated by spaces, replace all the successors in the sentence with the root forming it. If a successor can be replaced by more than one root, replace it with the root that has the shortest length.*

Solution: This method accepts a list of roots and a sentence String. It parses the string into individual words. For each word, it checks every dictionary root by seeing if the root matches the first characters of every word. If a successful match, it adds the new word to a new sentence. Once every word has been exhausted, it returns the new sentence.

Oh-time analysis: Several elements slow this program down: The lack of StringBuilder means that the new sentence we make runs in N^2 time. Every word is checked against every root, and that check may take a long time depending on the size of the roots. Ultimately, this is N^2, but the average runtime of this method is far higher than other N^2 methods, and times out faster than the others.

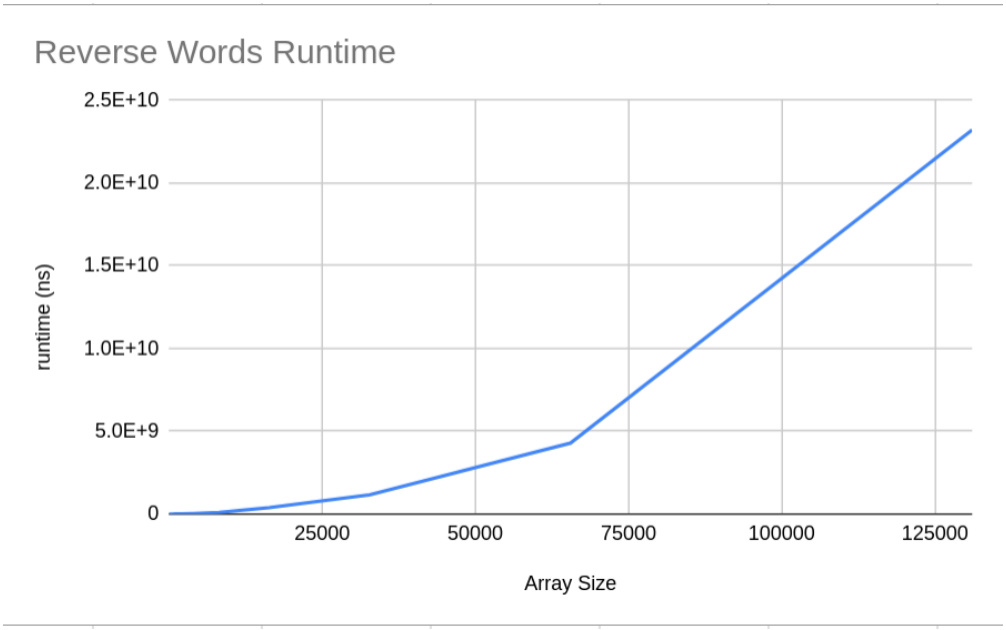


Reverse Words

Problem: *Given an input string s, reverse the order of the words.*

Solution: The method accepts a string and parses it into a List<String>, where each item is a single word. It then adds each item int he list backwards to a new string, and returns that.

Oh-time analysis: Like the earlier question, this was done prior to me learning StringBuilder, so the creation of the string guarantees that this is at least N^2. Because this program only needs to read through the string once to parse it, and only needs to run through its list once, it would otherwise have linear time.

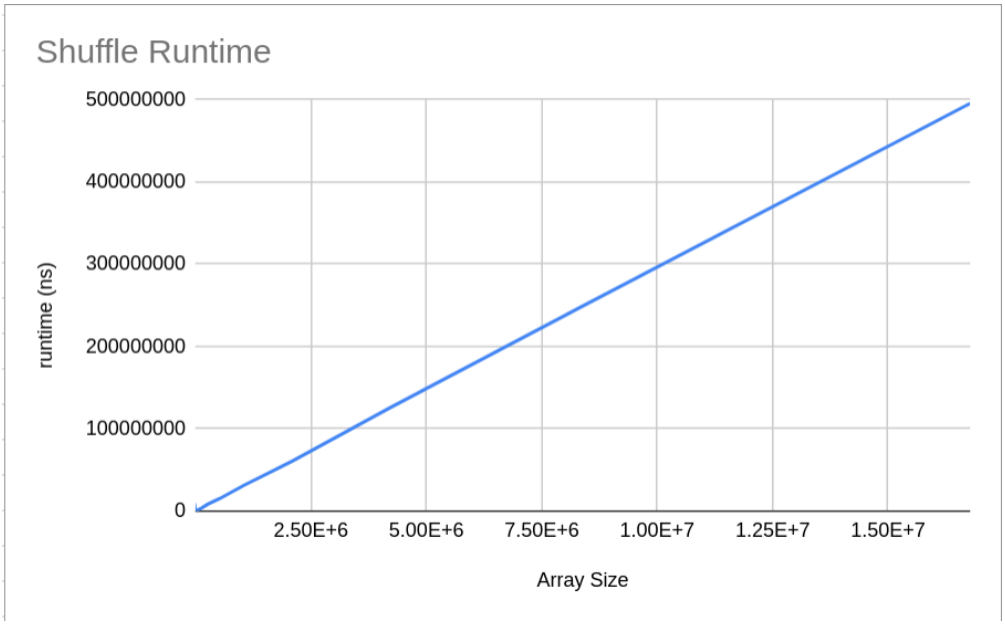


Shuffle

Problem: *Given the array nums consisting of* $2n$ *elements in the form* $[x1, x2, ..., xn, y1, y2, ..., yn]$. *Return the array in the form [x1,y1,x2,y2,...,xn,yn].*

Solution: The method creates a new array and fills it with the first half of *nums*, and then creates another array and fills it with the second half of *nums*. It then runs a loop which transforms *nums* by filling it with the two arrays, one at a time. Note that the method requires that n be specified, but this was not necessary.

Oh-time analysis: The method needs to read *nums* several times, but never within a nested for-loop. Because of this, *nums* runs in linear time.

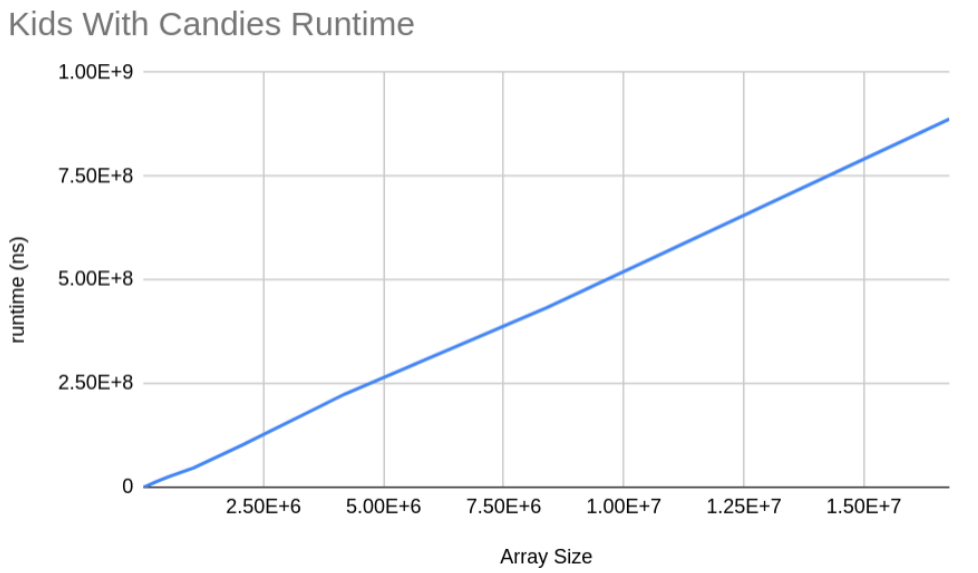


Kids with candies

Problem: *Given the array candies and the integer extraCandies, where candies[i] represents the number of candies that the ith kid has.*
For each kid check if there is a way to distribute extraCandies among the kids such that he or she can have the greatest number of candies among them. Notice that multiple kids can have the greatest number of candies.

Solution: The method begins by running a loop to both find the largest number in the array, while also increasing each array by the extraCandies size. Once finished, it runs a second loop to check if any of the numbers, now that they have been increased, are greater or equal to the max. It adds this as a boolean to an arraylist and returns the list.

Oh-time analysis: The method only runs two loops independent of each other. Because of this is runs a linear time.

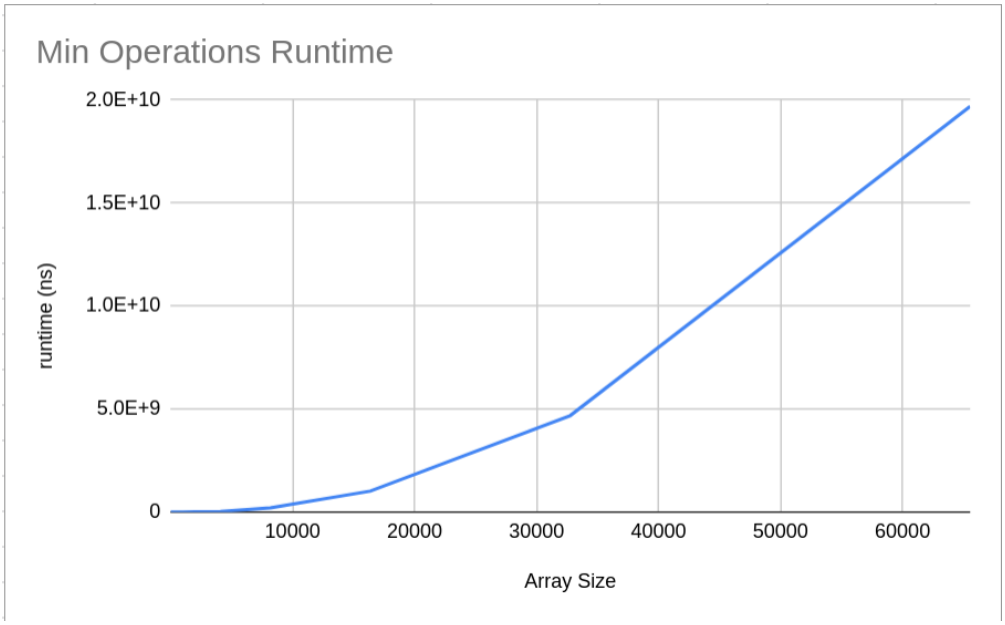


Min Operations

Problem: *You have n boxes. You are given a binary string boxes of length n, where boxes[i] is '0' if the ith box is empty, and '1' if it contains one ball.*
In one operation, you can move one ball from a box to an adjacent box. Box i is adjacent to box j if abs(i - j) == 1. Note that after doing so, there may be more than one ball in some boxes.
Return an array answer of size n, where answer[i] is the minimum number of operations needed to move all the balls to the ith box.
Each answer[i] is calculated considering the initial state of the boxes.

Solution: The metho runs a loop, selecting each box. When a box is selected, it then runs a second loop such that for every other box, if it has a one, calculate its distance from the selected box. The method adds all these distances together, saves it in an array, and after it has exhausted every box, returns the array.

Oh-time analysis: This method contains a nested for-loop, and has to exhaust the entire array for every index in the array. Hence, its runtime is N^2.

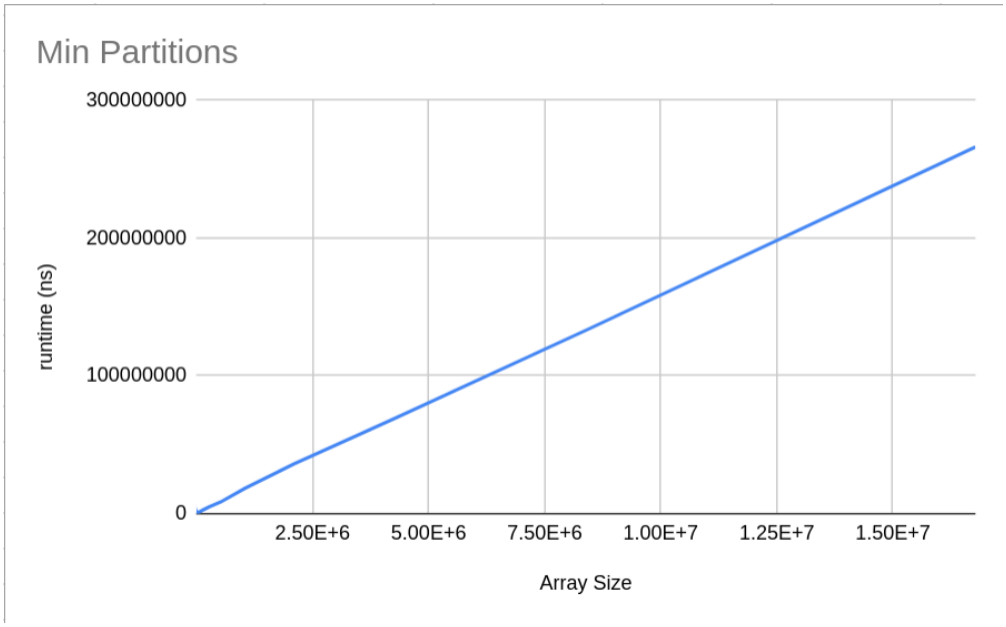


Min Partitions

Problem: *Given a string n that represents a positive decimal integer, return the minimum number of positive deci-binary numbers needed so that they sum up to n.*

Solution: It can be shown that any number will require a number of deci-binary numbers equal to the highest individual digit. This is because a deci-binary number can only increase each place by one, but can do this for every place at a time.
With that, we can simplify our method: Our method runs a for loop through the String searching for the maximum character. It then returns the integer represented by that character. We may decrease the average runtime by breaking the loop early when the program finds a 9, because that is the maximum number of required deci-binary numbers.

Oh-time analysis: Because this runs only a single for loop, it has linear runtime, with a smaller average case when it finds a 9.

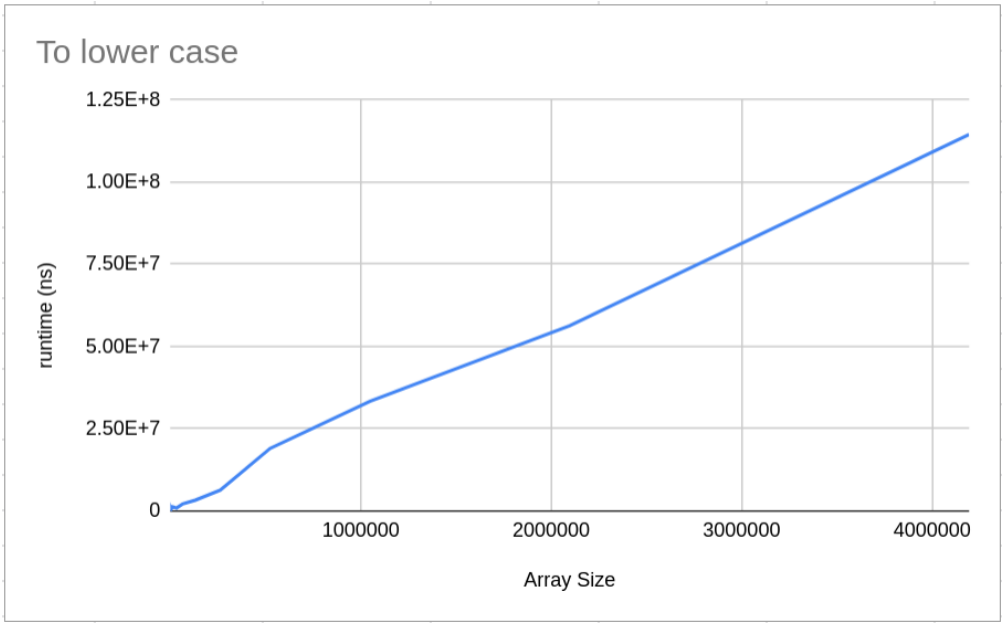


To Lower Case

Problem: *Implement function ToLowerCase() that has a string parameter str, and returns the same string in lowercase.*

Solution: The method begins by constructing a StringBuilder. it then runs a loop through each character in the string, and appends the lower case version to the stringbuilder. It then returns the Stringbuilder's string.

Oh-time analysis: By using String builder, the method improves on the previous ones and can maintain a linear time.

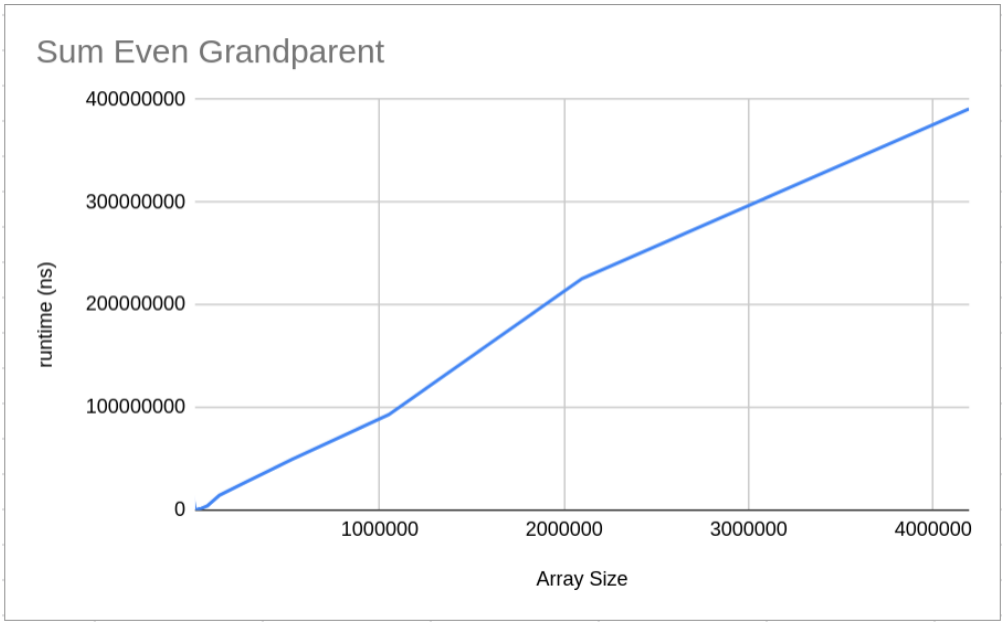


Sum Nodes of Even Grandparent

Problem: *Given a binary tree, return the sum of values of nodes with even-valued grandparent. (A grandparent of a node is the parent of its parent, if it exists.)*

Solution: The method begins at the root, and checks to see if the node is even. If it is, it then checks whether its grandchildren exist. If any of them do, the method adds the existing grandchildren's values to a sum. It the calls itself recursively on both the left and right nodes, and adds their result to the sum, and finally returns the sum.

Oh-time analysis: The entire tree is visited at least once in order to find all grandchildren, and for each visit to an even node, four other nodes must be rechecked. Because every node is visited, but each node only requires a constant number of re-checks, the method runs in linear time. However, it has a higher average linear time than previous methods.



Subrectangle Queries

Problem: *Implement the class SubrectangleQueries which receives a rows x cols rectangle as a matrix of integers in the constructor and supports two methods:*
updateSubrectangle(int row1, int col1, int row2, int col2, int newValue): Updates all values with newValue in the subrectangle whose upper left coordinate is (row1,col1) and bottom right coordinate is (row2,col2).
getValue(int row, int col): Returns the current value of the coordinate (row,col) from the rectangle.

Solution: The subrectangle class keeps track of a int multi-array. The updateSubrectangle, which is the one whose runtime is tested here, two nested loops are used. The outer one runs through the starting and ending columns, while the inner one runs through the starting and ending rows. For every index inside this area, the value is updated to the parameter accepted.
The testRectangleUpdate method tests this by creating a new subrectangle query of a given length and height, and updates every index within the rectangle.

Oh-time analysis: Because the multi-array is squared with every increase, and we are changing every index in the array, this program runs in N^2 time. Beyond that, the method has a demanding memory output. After the length and width were 32768, the virtual machine return a memory overload error.