Big O performance of the sortOfSort method

by Michael Font

In this report the time complexity of the sortOfSort method is analyzed.

Code

|  |
| --- |
| public class SortOfSort {  public static void sortOfSort(int[] unsortedArray) {  1 int sortedOffsetRight = unsortedArray.length;  2 int sortedOffsetLeft = -1;  3 int temp = 0;  4 while(sortedOffsetLeft +1 < sortedOffsetRight){  5 for(int i = sortedOffsetRight-1; i >= sortedOffsetRight-2; --i){  6 temp = 0;  7 for(int j = i - 1; j > sortedOffsetLeft; --j){  8 if(unsortedArray[j] > unsortedArray[i]){  9 temp = unsortedArray[i];  10 unsortedArray[i] = unsortedArray[j];  11 unsortedArray[j] = temp;  }  }  }  12 sortedOffsetRight -= 2;  13 for (int k = sortedOffsetLeft +1; k <= sortedOffsetLeft+2; ++k){  14 temp = 0;  15 for (int l = k + 1; l < sortedOffsetRight; ++l){  16 if (unsortedArray[l] > unsortedArray[k]){  17 temp = unsortedArray[k];  18 unsortedArray[k] = unsortedArray[l];  19 unsortedArray[l] = temp;  }  }  }  20 sortedOffsetLeft+=2;  }  //Return statement used for testing  //return unsortedArray;  }  } |

Analysis

The first three numbered lines of the code are elements that are only called once when the program is run regardless of the input size. Consequently, these constants will not be considered in the rest of this analysis.

Line number four is only called, first, at the start of looping over the array, and then only after 4 elements of the array at most, or 3 elements of the array at the least, have been processed by the inner For Loops. This leads to, on average, line four being called n/4 times, with n equal to the length of the array.

Line 4 = n/4.

Line number five has 3 instructions. The first instruction is called only at the initialization of the loop. Given that this outer loop only iterates over half of the array and is called once for every 2 elements in that half of the array, it is called n/4 times. Both instructions 2 and 3 are called once for each element from the end of the array to the array midpoint. Therefore, instructions 2 and 3 are called n/2 times, on average.

Line 5 = (n/4 + n/2 + n/2) = n/4 + n.

Line number six is called one per element, over half the array, and so six is equal to n/2.

Line 6 = n/2.

Line number seven is another For Loop, with instruction 1 of this loop being called once for each of instruction 1 of the outer loop. Since the outer loop is only called for approximately half of the elements of the array, instruction 1 of the inner loop is equal to n/4. Inner loop instruction 2 is called 1 time if the array is almost sorted and n-1 times if the array isn’t; on average there will be n/2 calls. The same analysis applies to instruction 3 of the inner loop. Consequently, the inner loop runs for n/4 plus n times.

Line 7 = (n/4 +n/2 + n/2) = n/4 + n.

Line number eight is called for each element of the array that has not already been sorted. It will be called n-1 times at the start of sorting the array to 1 time for the last comparison. The total number of times that line 8 is called, for an array of n elements, is equal to approximately one half of the sum of n-1. If the loop was being called for every element of the array, it would be equal to the sum of n-1 because on the first call line eight would make a comparison between the nth element and every element less than n, so it would be called n-1 times. Then it would compare n-2 with every element less than it, and again for n-3, and so on. So we would have N-1 calls, plus n-2 calls, … down to 1 call. It’s approximately half because each nested loop needs to sort only half of the array and an array can be either even or odd in length.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | 9 |
| Part of array sorted by left hand nested loop | | | | | Part of array sorted by right hand nested loop\* | | | | | |

\* n = 10

Line 8 = ~ 1/2 , where x = n-1

Lines number nine, ten, and eleven are only called if line eight finds a larger element in the unsorted array. Consequently, these lines are called zero times in the best case because the array is already sorted and in the worst case, would be called each time line eight is executed.

Lines twelve to twenty comprise the left- hand loop and are a mirror image of the instructions contained for the right-hand loop which is lines five to eleven.

Because of this, the left-hand loop would be additive to the time complexity of the right-hand loop. So, to get the Big O value of sortOfSort we only need to analyze the right-hand loop. For Big O we care only about the highest exponent for the number of operations. This is because the part of the algorithm with the most steps determines the amount of time for the algorithm to finish as n grows. It’s also why we ignore the constants, which are anything that’s run only a set number of times regardless of how big n is. As stated previously, line number five grows by n/4 + n as does line number seven. The n/4 can be ignored as it’s a lower power than n. Given that for an input of size n, the O value for line 5 and 7 is n, and lines 5 and 7 are multiplicative with respect to each other, then the O value of the nested For loop is n2 andn2 is the highest power of the function; therefore, sortOfSort is O(n2).