Laboratory practice No. 2: Big O Notation

Andrés Grimaldos
Universidad Eafit
Medellín, Colombia
agrimaldoe@eafit.edu.co

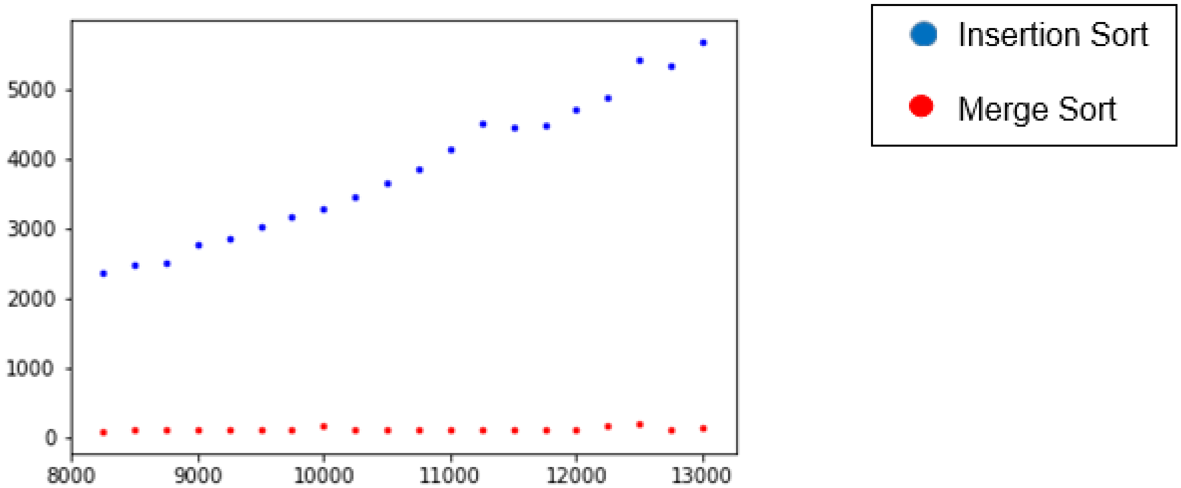
Alejandro Salazar
Universidad Eafit
Medellín, Colombia
asalazara1@eafit.edu.co

3) Practice for final project defense presentation

3.1

	Insertion Sort	**Merge Sort**
Size	**Time in milliseconds**	**Time in milliseconds**
8250	2356.14453125 ms	67.529296875 ms
8500	2487.414794921875 ms	99.700439453125 ms
8750	2512.483642578125 ms	101.377197265625ms
9000	2767.577392578125 ms	103.5048828125 ms
9250	2858.9970703125 ms	103.993408203125 ms
9500	3022.591552734375 ms	105.610107421875 ms
9750	3164.274169921875 ms	106.29443359375 ms
10000	3263.91015625 ms	167.4111328125 ms
10250	3445.053955078125 ms	108.23681640625 ms
10500	3640.7666015625 ms	109.174072265625 ms
10750	3849.56298828125 ms	102.306396484375 ms
11000	4124.0400390625 ms	109.013916015625 ms
11250	4498.550048828125 ms	98.99560546875 ms
11500	4460.859130859375 ms	111.492431640625 ms
11750	4488.846435546875 ms	100.04052734375 ms
12000	4712.1748046875 ms	114.762939453125 ms
12250	4877.6484375 ms	170.05517578125 ms
12500	5416.955322265625 ms	178.380615234375 ms
12750	5339.886474609375 ms	113.726806640625 ms
13000	5681.53076171875 ms	117.151611328125 ms

3.2 Size input vs execution time (milliseconds)



3.3 Merge sort is more efficient than insertion sort, because when the length of the array is very big merge sort tends to a complexity of nLog(n), but insertion sort rise in a proportion of n^2 which increases faster than nLog(n).
3.4 Considering the complexity of each one, it is better to use merge sort instead insertion sort, because when the array goes to a very huge number of objects n^2 rises very fast.
3.5 Although the complexity in the worst of the cases of the merge sort is smaller than insertion sort's, when the array is almost organized, insertion sort could be faster, because their code has less comparations than merge sort's, so as long as we are not in the worst of the cases(almost disorganized), insertion sort could be faster.
3.6 The exercise maxSpan receives an array as parameter and with the leftmost and rightmost ocurrences, it looks for how many elements are between those numbers, that is called "span", returning the maximum span, trying all the posibilities in the array. To get that, you can use a double loop finding the two ocurrences (leftmost and rightmost), after that, you substract their indexes and add 1 to find the number of elements between them.
3.7 Array 2: (notation in github)

1. countEvens:
   $T(n) = C_1 + C_2n + C_3n + C_4$
   $T(n)$ es $O(C_1 + C_2n + C_3n + C_4)$
   $O((C_2 + C_3)n)$ // sum rule
   $O(n)$ // product rule

2. lucky13:
   $T(l) = C_1l + C_2l + C_3 + C_4$
   $T(l)$ es $O(C_1l + C_2l + C_3 + C_4)$
   $O((C_1+C_2)l)$ // sum rule
   $O(l)$ // product rule

3. Sum28:
   $T(m) = C_1 + C_2m + C_3m + C_4$
   $T(m)$ es $O(C_1 + C_2m + C_3m + C_4)$
   $O((C_2 + C_3)m)$ // sum rule
   $O(m)$ // product rule

4. more14:
   $\mathbf{T(r) = C_1 + C_2 + C_3r + C_4r + C_5r + C_6}$
   $\mathbf{T(r)\ es\ O(C_1 + C_2 + C_3r + C_4r + C_5r + C_6)}$
   $\mathbf{O((C_3 + C_4 + C_5)r)}$ // sum rule
   $\mathbf{O(r)}$ // product rule

5. fizzArray:
   $\mathbf{T(t) = C_1 + C_2t + C_3t + C_4}$
   $\mathbf{T(t)\ es\ O(C_1 + C_2t + C_3t + C_4)}$
   $\mathbf{O((C_2 + C_3)t)}$ // sum rule
   $\mathbf{O(t)}$ // product rule

Array 3: (notation in github):

1. lineraIn:
   $\mathbf{T(n) = C_1 + C_2q + C_3qn + C_4qn + C_5qn + C_6qn + C_7}$
   $\mathbf{T(n)\ es\ O(C_1 + C_2n + C_3n^2 + C_4n^2 + C_5n^2 + C_6n^2 + C_7)}$ in the worst case where q = n, because q<=n.
   $\mathbf{O((C_3 + C_4 + C_5 + C_6)n^2)}$ // sum rule
   $\mathbf{O(n^2)}$ // product rule

2. seriesUp:
   $\mathbf{T(l) = C_1 + C_2 + C_3l + C_4lf + C_5lf + C_6lf + C_7}$
   $\mathbf{T(l)\ es\ O(C_1 + C_2 + C_3l + C_4l^2 + C_5l^2 + C_6l^2 + C_7)}$ in the worst case where f = l, because f <= l
   $\mathbf{O((C_4 + C_5 + C_6)l^2)}$ // sum rule
   $\mathbf{O(l^2)}$ // product rule

3. canBalance:
   $\mathbf{T(m) = C_1 + C_2 + C_3m + C_4m + C_5m(m\text{-}1) + C_6m(m\text{-}1) + C_7m + C_8 + C_9m + C_{10}}$
   $\mathbf{T(m)\ es\ O(C_1 + C_2 + C_3m + C_4m + C_5m(m\text{-}1) + C_6m(m\text{-}1) + C_7m + C_8 + C_9m + C_{10})}$
   $\mathbf{O((C_5 + C_6)m^2)}$ // sum rule
   $\mathbf{O(m^2)}$ // product rule

4. fix34:
   $\mathbf{T(r) = C_1 + C_2r + C_3r + C_4r + C_5r + (C_6 + C_7 + C_8)r(r\text{-}2) + C_9}$
   $\mathbf{T(r)\ es\ O(C_1 + C_2r + C_3r + C_4r + C_5r + (C_6 + C_7 + C_8)r(r\text{-}2) + C_9)}$
   $\mathbf{O((C_6 + C_7 + C_8)r^2)}$ // sum rule
   $\mathbf{O(r^2)}$ // product rule

5. maxSpan:
   $\mathbf{T(t) = C_1 + C_2 + C_3 + C_4 + C_5t + (C_6 + C_7 + C_8 + C_9 + C_{10})t^2 + C_{11}}$
   $\mathbf{T(t)\ es\ O(C_1 + C_2 + C_3 + C_4 + C_5t + (C_6 + C_7 + C_8 + C_9 + C_{10})t^2 + C_{11})}$
   $\mathbf{O((C_6 + C_7 + C_8 + C_9 + C_{10})t^2)}$ // sum rule
   $\mathbf{O(t^2)}$ // product rule

3.8

Array 2:

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

'n', 'l', 'm', 'r' is the array length in the exercises 1, 2, 3 and 4 respectively

't' is the amount of numbers starting in 0 to add(in order) in the array, therefore, the length of the returned array too

Array 3:

'n' is the outer array length and 'q' is the inner array length

'l' is the number of times that the pattern 1 to 'f' would be in the array(with f increasing) and the maximum value of 'f', so the length of the last pattern; and 'f' is the amount of numbers of the pattern 1 to 'f'

'm', 'r', 't' is the array length in exercises 3, 4 and 5 respectively

4) Practice for midterms

4.1 *c*
4.2 *d*
4.3 *b*
4.4 b
4.5 d
4.6 a
4.7
4.7.1 $T(n-1)+C_1$
4.7.2 n
4.8 a
4.9 d
4.10 c
4.11 c
4.12 b
4.13 c
4.14 c

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473