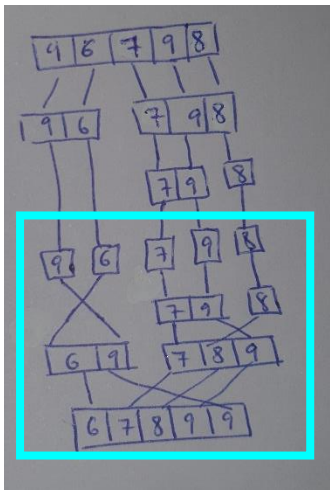
Paula Beltrán Marianini
Lucía Gil Maroto
Group 1291 Pair 3
Practice 3 AEDAs

LAB 3: Disjoint sets and connected components; the Traveling Salesman Problem

Part I: The Selection Problem

1. **Argue that MergeSort sorts a table of 5 elements with at most 8 key comparisons**



To explain this exercise, we are going to use the example from the image. The comparisons are made in the section marked in blue. We have prepared the array with the numbers that are going to give us the worst case of comparisons.

First comparison is between 9 and 6. Then, we compare 7 with nine. Then, we compare 8 with 7 and, as 7<8, we need to compare 8 with 9.

Then, we compare 6 with 7. And finally, we have to compare 9 with 7, with 8 and with 9 to order completely the array.

If we make the sum of the comparisons for the worst case, we get 8 comparisons.

2. **Actually, in qsel_5 we are only looking for the median of 5 elements, not necessarily an ordering. Could we reduce the number of comparisons necessary?**

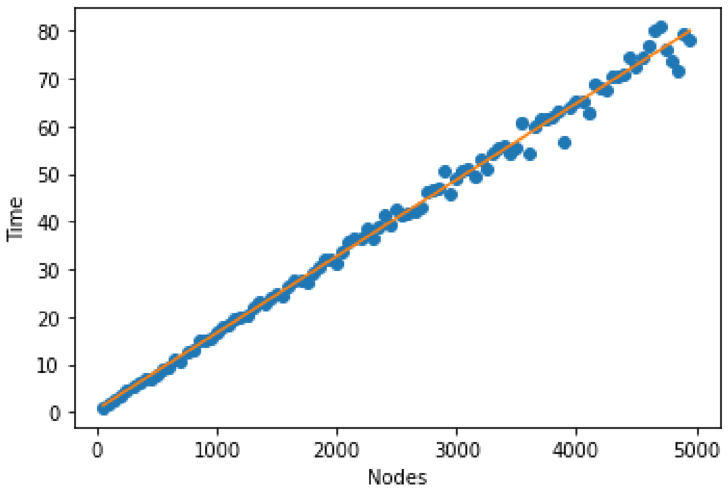
Yes, because setting $W_{QS5}(N) = \max_{\sigma\in\Sigma_N,k} n_{QS5}(\sigma, k)$; then computing the N/5 medians has a cost of 8N/5 kcs.

It ehn recursively applies QuickSelect5 over a table with N/5 elements, with a cost at most $W_{QS5}(N/5)$. Also, $|\sigma_{left}|, |\sigma_{right}| \leq 7N/10$.

Thus, the overall cost of QuickSelect5 can be bounded as:

$$\begin{aligned} n_{QS5}(\sigma, 1, N, k) &= n_{Pivot5}(\sigma, 1, N) + n_{split}(\sigma, 1, N) + \\ &\quad \max_m\{n_{QS5}(\sigma_{left}, k), n_{QS5}(\sigma_{right}, k-m)\} \\ &\leq \frac{8N}{5} + W_{QS5}\left(\frac{N}{5}\right) + N - 1 + \\ &\quad W_{QS5}\left(\frac{7N}{10}\right) \end{aligned}$$

We can see with a plot that the cost is linear (O(n)).

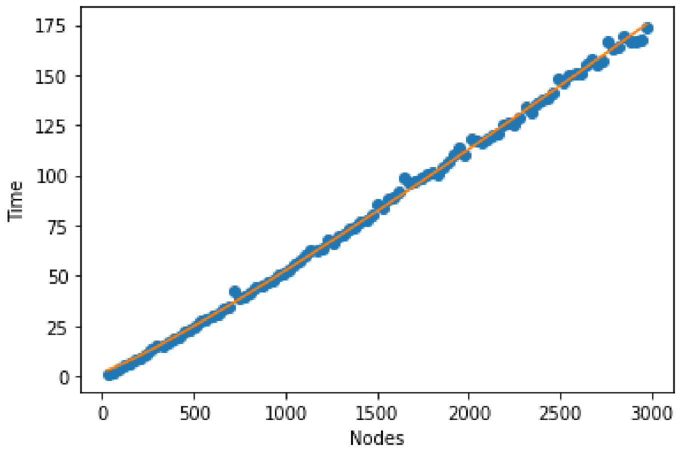


Where the blue dots are the results of our function execution and the orange line, the representation of O(n). We see that our function follows a linear tendency.

3. **What kind of growth behavior vis-a-vis execution time can we expect in the worst case for the function qsort_5? Try justifying your answer experimentally and analytically.**

   The worst case we are going to obtain after implementing qsort_5 will be O(nlogn). The way to obtain this complexity will be:

   - Divide the array into [n/5] groups with each group having 5 elements
   - Find median of each group using insertion sort and then pick the median from this list
   - Then you will need to recursively try and find the median of [n/5] medians calculated from each group.
   - Partition the array around this median.



Where the blue dots are the results of our function execution and the orange line, the representation of O(n*log(n)). We see that our function follows the tendency expected.

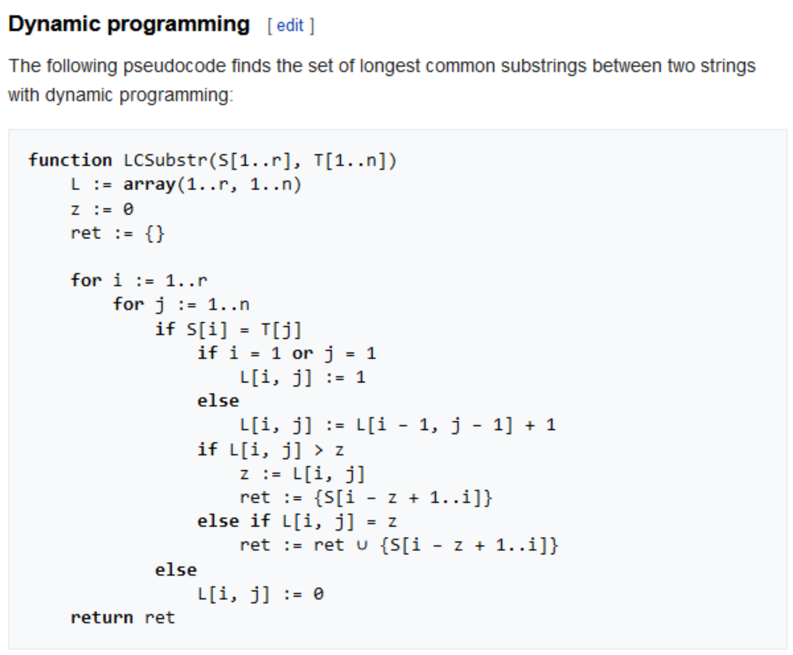
Paula Beltrán Marianini
Lucía Gil Maroto
Group 1291 Pair 3
Practice 3 AEDAs

Part II: Dynamic Programing

1. **The problem of finding the maximum non-consecutive common substring is often confused with that of finding the maximal consecutive one. See, as an example, the entry Longest common substring problem in Wikipedia. Describe a dynamic programming algorithm that finds the longest consecutive common substring between two strings S and T and apply it by hand to find the longest common substring between bajamas and bananas**

   With dynamic programming, the idea is to find the longest common substring in O(m/n) time. We should find the length of the longest common suffix for all substrings of both strings and store these lengths in a table.

   The algorithm we are proposing in this exercise is this one from wikipedia:



Dynamic programming [edit]

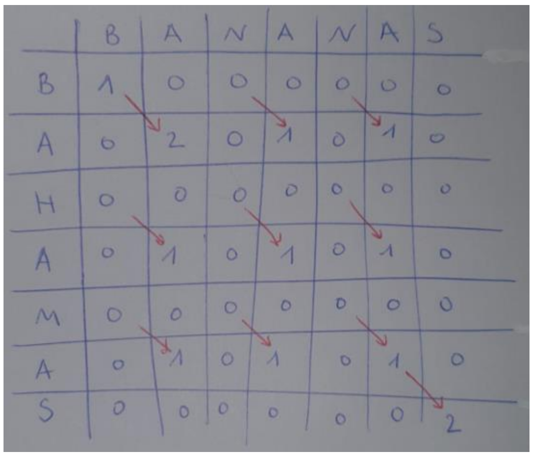
The following pseudocode finds the set of longest common substrings between two strings with dynamic programming:

```
function LCSubstr(S[1..r], T[1..n])
    L := array(1..r, 1..n)
    z := 0
    ret := {}

    for i := 1..r
        for j := 1..n
            if S[i] = T[j]
                if i = 1 or j = 1
                    L[i, j] := 1
                else
                    L[i, j] := L[i - 1, j - 1] + 1
                if L[i, j] > z
                    z := L[i, j]
                    ret := {S[i - z + 1..i]}
                else if L[i, j] = z
                    ret := ret ∪ {S[i - z + 1..i]}
            else
                L[i, j] := 0
    return ret
```

The algorithm proposed saves the length of the substrings and the substring. But it has a limitation, as it saves only one possibility, if there is more than one substring with the same length, the algorithm only returns the last one read.

In our case, if we analysed the algorithm by hand, we get:



	B	A	N	A	N	A	S
B	1	0	0	0	0	0	0
A	0	2	0	1	0	1	0
H	0	0	0	0	0	0	0
A	0	1	0	1	0	1	0
M	0	0	0	0	0	0	0
A	0	1	0	1	0	1	0
S	0	0	0	0	0	0	2

In this case, there are two substrings with the same length, “BA” and “AS” but the algorithm is going to return only “AS” because is the last one read.

2. **We know that finding the smallest number of multiplications necessary to multiply a list of n matrices is O(n^3), but now we want to estimate that number, v(n) more precisely. Estimate such a number using an expression v(n) = f(n) + O(g(n)) with f and g such that |v(n) − f(n)| = O(g(n)).**

   Using the EOS property in theory slides 204 and 205, we get that are made this number of operations.

$$\sum_{j=2}^{N}\left(\sum_{i=1}^{j}(j-i)\right)$$

The solution to that operation is:

$$\frac{N^3}{6}-\frac{N}{6}$$

Therefore, we can say that:

$$v(n) = \frac{N^3}{6}-\frac{N}{6}$$

now, we have only to assign f(n) and O(g(n)). As O(g(n)) can’t be negative because is the complexity of a function. Therefore, we just have to switch the terms in the previous operation.

$$v(n) = -\frac{N}{6} + \frac{N^3}{6}$$

So, we get that f(n) = $-\frac{N}{6}$ and O(g(n)) = $\frac{N^3}{6}$