Verónica Gaspes
School of Information Technology
May 25, 2023

Python: a gateway to machine learning DT8051
Written Exam

June 2 2023 from 09:00 to 14:00

- No aids or tools are allowed.
- **Grading criteria:** You can get at most 26 points.

  To pass you need at least 50% of the points.

  For the highest grade you need more than 90% of the points.
- **Responsible:** Verónica Gaspes (0722391416)

- **Read carefully!** Some exercises might include explanations, hints and/or some code. What you have to do in each exercise is marked with the points that you can get for solving it (as **(X pts)**).
- **Write clearly!** You must write complete sentences in your answers and when you provide an example you should explain in what way it is an example of what was asked for.
- **Motivate your answers!**

Good Luck!

1. Consider the following Python function that calculates the number of pairs where an element that comes before in the array is larger than an element that comes after in the array.

```python
def count_inversions(a):
    n = len(a)

    count_inv = 0
    for i in range(n):
        for j in range(i+1, n):
            if a[i] > a[j]: count_inv += 1
    return count_inv
```

The function has one argument: an array a with integer numbers. The function returns the number of times the following happens: $i < j$ and $a[i] > a[j]$

(a) **(1 pts)** What does the function calculate when it is called with the argument `[3,1,2,5,4,0]`?

(b) **(1 pts)** Provide an argument for which the result is `0`.

(c) **(1 pts)** What aspect of the input would you consider the size of the input?

(d) **(1 pts)** Are there inputs that can be considered worst cases? Explain how you came up with your answer.

(e) **(1 pts)** Explain how you calculate the Big-O for the execution time of this function. What do you get as a result?

2. The algorithm in Exercise 1 is a so called *Brute Force* algorithm: it tests all possible pairs of elements where the first element comes before in the array than the second element. In this exercise you will develop a *Divide & Conquer* solution to the same problem.

   (a) **(1 pts)** What aspect of the input will you use to determine the sub-problems?

   (b) **(1 pts)** Explain what subproblems you consider and what you have to do to obtain the subproblems.

   (c) **(2 pts)** Explain what you have to do to combine the solutions to the subproblems, possibly doing some extra work, to solve the problem.

   (d) **(2 pts)** Write down the algorithm (or Python function) that implements your ideas from parts 2(a), 2(b) and 2(c).

   (e) **(1 pts)** The Master Method helps calculate the Big-O for recursive algorithms designed using Divide & Conquer. Use it to calculate the Big-O for your algorithm.

   Here is the Master Method:

$$T(n) \text{ is } \begin{cases} O(n^d \log n), & \text{if } a = b^d \\ O(n^d), & \text{if } a < b^d \\ O(n^{log_b a}) & \text{if } a > b^d \end{cases} \quad (1)$$

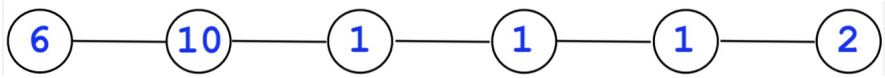
   with a the number of recursive calls, b the shrinking factor for the size of the input and d the exponent of the running time of the combine part.

3. The function `mwis` defined below solves the problem of calculating the total weight of the maximum weighted independent set in a path graph.

```python
def mwis(a):
    n = len(a)
    if n == 0: return 0
    if n == 1: return a[0]
    s1 = mwis(a[0:-1])
    s2 = mwis(a[0:-2])
    return max(s1, s2 + a[n-1])
```

Python note: `a[0:-1]` is an array with all the elements of `a` except the last one. Similarly, `a[0:-2]` is an array with all the elements of `a` except the two last ones.

The argument to the function `mwis` is an array of positive integer numbers that are the weights of consecutive nodes in a path graph. For example, the input `[6, 10, 1, 1, 1, 2]` stands for the path graph



The function calculates the total weight of the set of independent nodes with the maximum total weight. In a set of independent nodes there are no edges between any two nodes. In the example the result is 13 using nodes in places $1, 3, 5$ (with weights $10, 1, 2$). Unfortunately, the program takes too much time due to overlapping recursive calls.

(a) **(1 pts)** What is the base case and what are the recursive calls?

(b) **(1 pts)** Enumerate the recursive calls made by the program when used with input `[1, 4, 5, 4]`? What is the result?

(c) **(1 pts)** What are the smaller problems that need to be solved in order to calculate the maximum total weight of a set of independent nodes up-to the first n nodes?

(d) **(1 pts)** Suggest a way of storing results to smaller problems so that they are available when needed.

(e) **(2 pts)** Use your suggestion to write a dynamic programming algorithm and show how to obtain the result to the problem. You can express your algorithm as a Python program.

(f) **(1 pts)** Explain how you calculate the Big-O for your algorithm. What is the Big-O of your algorithm?

4. Consider the following program that calculates the number of pairs in an array a that add up to a number t:

```
def sums(a, t):
    n = len(a)
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
                if a[i] + a[j] == t: count = count + 1
    return count
```

The arguments are an array of integer numbers a and an integer number t.

Unfortunately this algorithm is quadratic in the size of the array! More formally, $T(n) = O(n^2)$ for n the number of elements in the array.

(a) **(1 pts)** Construct an input with an array of 5 elements with exactly 2 such pairs that add up to 4. Show what the pairs are.

(b) **(1 pts)** Explain how you could improve the execution time. Write an algorithm that implements your idea.

(c) **(1 pts)** Explain how to calculate the Big-O for the execution time of your algorithm. What is the Big-O? Is it better than the original one? If you use a data structure you need to state what the execution times of the operations you use are.

5. In this exercise we would like to solve the problem of finding the largest M items in a stream of many many items. This can be used for example to detect fraud by isolating the 100 largest transactions of all Swedish banks every 24 hours. For this exercise we will use an array instead of a stream so that you do not have to recall how to read from a file or similar.

(a) **(2 pts)** Your task is to write an algorithm (or a Python function that implements the algorithm) that uses a heap to maintain the M largest elements in an array. The input to your algorithm is an array and a number M. The algorithm should end by printing the M largest elements of the array. If there are no more than M elements it should print all the elements in the array. The execution time of your algorithm should be $\mathcal{O}(n \log(M))$ for n the length of the array.
Python note: In Python heaps are lists that are manipulated with the operations `heapq.heappush(h,x)` and `heapq.heappop(h)`.

(b) **(2 pts)** Explain how to calculate the Big-O for the execution time of your algorithm.