

---

# FIT2004 S1/2015: Lab questions for week 4

THIS PRAC IS **ASSESSED!** (6 Marks)

**CLASS:** This programming exercise has to be prepared in advance before you enter your assigned lab in week 4. Your lab time in that week should be mainly used to iron out bugs, clarify any final doubts by speaking with your demonstrator/tutor, and finalize your submission which the demonstrator will check. He will ask you a series of questions to assess your understanding of this exercise, and gauge how you implemented it. It is required that you implement this exercise strictly using **Python programming language**. Practical work is marked on the performance of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! “Forgetting” is not an acceptable explanation for lack of understanding. You must write all the code yourself, and should not use any external library routines, except those that are considered standard. The usual input/output and other unavoidable routines are exempted.

**DEMONSTRATORS** are not obliged to mark programs that do not run or that crash. Time allowing, they will try to help you track down errors, but they are NOT required to mark programs in such a state, particularly those that do not work. Therefore keep backup copies of working partial-solutions.

**OBJECTIVE:** This practical exercise is to help you develop insights into new ways of generating permutations of elements, your understanding of iteration and recursion, taking an algorithm and writing a program from it, and analysis of time-complexity of sorting algorithms.

## Background

The number of permutations of  $N$  distinct elements is  $N$  factorial, written in mathematics as  $N!$ . The factorial of  $N$  is simply the product of all positive integers less than or equal to  $N$ :  $N! = 1 \times 2 \times 3 \times \dots \times N$ .

Define a permutation set  $S_N$  as the set of all permutations of the first  $N$  letters from the English alphabet. (Assume  $N \leq 26$ .) For example, the permutation set  $S_4$ , made up of letters a, b, c, d has  $4! = 24$  permutations, which are:

0 abcd	6 bacd	12 cabd	18 dabc
1 abdc	7 badc	13 cadb	19 dacb
2 acbd	8 bcad	14 cbad	20 dbac
3 acdb	9 bcda	15 cbda	21 dbca
4 adbc	10 bdac	16 cdab	22 dcab
5 adcb	11 bdca	17 cdba	23 dcba

Observe that these permutations are printed in the lexicographic order of the letters. For each permutation in this list, its permutation (decimal) number is listed to its left. Permutation number 0 represents a fully-ordered permutation: abcd in the above example.

There are several ways to print all the permutations in  $S_N$ , for any given  $N$ , in the lexicographic order. One way relies on converting the permutation number from a decimal system (or base-10) into a number in a *factorial number system* (or base-!). A factorial number system is specifically useful to numbering permutations in any  $S_N$ . That is, given  $N$ , any permutation number  $D$  base 10 (or  $(D)_{10}$  for short) in  $S_N$  can be **uniquely** represented as:

$$(D)_{10} = C_{N-1} \times (N-1)! + C_{N-2} \times (N-2)! + \cdots + C_1 \times 1! + C_0 \times 0! \quad (1)$$

where  $C_{N-1}, C_{N-2}, \dots, C_0$  form the number in the factorial system, or  $(C_{N-1}C_{N-2} \cdots C_0)!$  for short.

If the property in Equation 1 holds, then the number  $(C_{N-1}C_{N-2} \cdots C_0)!$  in the factorial number system is equivalent to the permutation number  $(D)_{10}$  in the decimal number system. As a concrete set of examples, below is a table that shows the entire set of decimal permutation numbers and their corresponding factorial numbers for each permutation in  $S_4$  over the letters  $\{a, b, c, d\}$ .

BASE-10	BASE-!	PERMUTATION	BASE-10	BASE-!	PERMUTATION
-----					
( 0 ) <sub>10</sub>	(0000) <sub>!</sub>	abcd	(12) <sub>10</sub> =	(2000) <sub>!</sub>	cabd
( 1 ) <sub>10</sub>	(0010) <sub>!</sub>	abdc	(13) <sub>10</sub> =	(2010) <sub>!</sub>	cadb
( 2 ) <sub>10</sub>	(0100) <sub>!</sub>	acbd	(14) <sub>10</sub> =	(2100) <sub>!</sub>	cbad
( 3 ) <sub>10</sub>	(0110) <sub>!</sub>	acdb	(15) <sub>10</sub> =	(2110) <sub>!</sub>	cbda
( 4 ) <sub>10</sub>	(0200) <sub>!</sub>	adbc	(16) <sub>10</sub> =	(2200) <sub>!</sub>	cdab
( 5 ) <sub>10</sub>	(0210) <sub>!</sub>	adcb	(17) <sub>10</sub> =	(2210) <sub>!</sub>	cdba
( 6 ) <sub>10</sub>	(1000) <sub>!</sub>	bacd	(18) <sub>10</sub> =	(3000) <sub>!</sub>	dabc
( 7 ) <sub>10</sub>	(1010) <sub>!</sub>	badc	(19) <sub>10</sub> =	(3010) <sub>!</sub>	dacb
( 8 ) <sub>10</sub>	(1100) <sub>!</sub>	bcad	(20) <sub>10</sub> =	(3100) <sub>!</sub>	dbac
( 9 ) <sub>10</sub>	(1110) <sub>!</sub>	bcda	(21) <sub>10</sub> =	(3110) <sub>!</sub>	dbca
(10) <sub>10</sub>	(1200) <sub>!</sub>	bdac	(22) <sub>10</sub> =	(3200) <sub>!</sub>	dcab
(11) <sub>10</sub>	(1210) <sub>!</sub>	bdca	(23) <sub>10</sub> =	(3210) <sub>!</sub>	dcba

The below works out one of the decimal-to-factorial system conversion from above explicitly,

$$(15)_{10} = 2 \times 3! + 1 \times 2! + 1 \times 1! + 0 \times 0! = (2110)!$$

These factorial digits of a permutation  $C_{N-1}, C_{N-2}, \dots, C_0$  have a very special meaning. Notice first that there are as many factorial digits as there are the elements in the set. Each digit in this system gives the number of *inversions* of each element in the permutation as compared with the fully-ordered permutation. What does this mean? Using the same example as above:

$$(15)_{10} = (2110)_{!} \quad \quad \quad \text{c b d a}$$

For better visual understanding, the factorial digits are arranged right under this permutation:

$$\begin{array}{cccc} & \text{c} & \text{b} & \text{d} & \text{a} \\ & 2 & 1 & 1 & 0 \end{array}$$

Then, the digits imply the following:

For the 1st element 'c', there are 2 elements to the right of it ('b' and 'a') that are lexicographically LESS than this element.

For the 2nd element 'b', there is 1 element to the right of it ('a') that are lexicographically LESS than this element.

For the 3rd element 'd', there is 1 element to the right of it ('a') that are lexicographically LESS than this element.

For the 4th element 'a', there are 0 elements to the right of it that are lexicographically LESS than this element.

In fact, the sum of these factorial digits gives the total number of **transpositions** (or swaps between adjacent elements) needed to convert the permutation into a fully sorted order. The minimum of this sum over all permutations is clearly 0 (when the permutation is already fully sorted), and the maximum is  $\frac{N(N-1)}{2}$  (when the permutation is fully reversed). In the example we have been considering, the sum of factorial digits in (2110)<sub>4</sub>! = 2 + 1 + 1 + 0 = 4, implying that this permutation needs 4 transpositions to convert cbda to abcd. Let see how! Working from right to left and applying the number of successive swaps as indicated by the factoradic digit under each symbol:

```
c b d a
2 1 1 0
```

```
SWAPS:          #1          #2          #3          #4
               cbda --> cbad --> cabd --> acbd --> abcd
```

It is also straightforward to construct a permutation string from just the factorial digits (and the total ordering of the  $N$  letters in the permutation). Let's follow this with the same example:

$N = 4$

Below is the total ordering of the elements in  $S_4$

```
Index:          0 1 2 3
Total ordering:  a b c d
```

Our goal now is to generate the permutation given the following  
Factorial digits : 2 1 1 0

First digit is 2; find the symbol at index=2 in the total ordering set and place it as the first symbol of the permutation:

```
      c
Remove c from the total ordering:
Index:          0 1 2
Total ordering:  a b d
```

Next digit is 1; find the symbol at index=1 in this new total ordering set and place it as the second symbol of the permutation:

```
      c b
Remove b from the total ordering:
Index          0 1
Total ordering:  a d
```

Next digit is 1 again; find and place the symbol at index=1  
as the third symbol of the permutation:

c b d

Remove d from the total ordering:

Index 0

Total ordering: a

Next digit is 0; find and place the symbol at index = 0  
as the fourth symbol of the permutation:

c b d a

Based on this background, your lab assignment is to address the following questions:

## Lab Assignment Questions

1. Write a Python program called `permute` which accepts an integer  $N$  as its argument. For the given  $N$ , your program should iterate over  $N!$  numbers, and at each step:
  - Convert each (permutation) number from the base-10 (decimal) representation to its corresponding base-! (factorial) representation.
  - Convert the factorial number into a permutation string, assuming we are dealing with the first  $N$  letters of the English alphabet. (It is okay to assume that  $N \leq 10$  for this exercise.)
  - Compute the sum of these factorial digits for the given permutation

Having written this program:

- (a) Print to an output file (`fit2004.lab4.output.txt`) all permutations in the lexicographic order containing the following information:
  - Column 1: The decimal number of the permutation
  - Column 2: The factorial digits of that permutation
  - Column 3: The sum of the factorial digits
  - Column 4: The corresponding permutation string using the first  $N$  letters from the English alphabet as the total ordering.

(Your output should roughly be similar to the sample output given overleaf.)
- (b) Print a frequency table (refer output format overleaf) showing the number of permutations (or frequency) as a function of sum of factoradic digits.
- (c) Calculate and print the *weighted average* of the sum factoradic digits over all permutations.
- (d) How is this weighted average growing asymptotically as a function of  $N$ ? Justify your answer with a clear reasoning.
- (e) This growth informs the complexity of a certain sorting algorithm we discussed specifically in weeks 2 and 3. Which one and what case – Best, Average or Worst – does your answer to (d) illuminate?

2. Using the background information, write another script that takes any two permutations of a finite set of distinct elements and returns as an answer the **smallest number of (adjacent) transpositions** required to convert one permutation to another. (E.g.: for converting the permutation **cadb** to **bacd** or vice versa, the answer would be 4.)

## GENERAL FORMAT OF YOUR OUTPUT FOR Q1:

```
INPUT TO THE SCRIPT: N = 4
TOTAL NUMBER OF PERMUTATIONS = 24
decimal      factoradic    sum    permutation
( 0)_10      (0000)_!        0      abcd
( 1)_10      (0010)_!        1      abdc
( 2)_10      (0100)_!        1      acbd
( 3)_10      (0110)_!        2      acdb
( 4)_10      (0200)_!        2      adbc
( 5)_10      (0210)_!        3      adcb
( 6)_10      (1000)_!        1      bacd
( 7)_10      (1010)_!        2      badc
( 8)_10      (1100)_!        2      bcad
( 9)_10      (1110)_!        3      bcda
(10)_10      (1200)_!        3      bdac
(11)_10      (1210)_!        4      bdca
(12)_10      (2000)_!        2      cabd
(13)_10      (2010)_!        3      cadb
(14)_10      (2100)_!        3      cbad
(15)_10      (2110)_!        4      cbda
(16)_10      (2200)_!        4      cdab
(17)_10      (2210)_!        5      cdba
(18)_10      (3000)_!        3      dabc
(19)_10      (3010)_!        4      dacb
(20)_10      (3100)_!        4      dbac
(21)_10      (3110)_!        5      dbca
(22)_10      (3200)_!        5      dcab
(23)_10      (3210)_!        6      dcba
```

### FREQUENCY TABLE

SUM	FREQ.
0	1
1	3
2	5
3	6
4	5
5	3
6	1

Weighted average of sum = 3

```
--o0o--
END
--o0o--
```