# BSG Lab 3

Sara Montese

October 2023

1. Python code:
   **Preprocessing**:

```
import numpy as np

#read and process the distance matrics
with open("ultrametric.txt") as f:
    contents = f.read()
content = contents.split("\n")[:-1]
ult_matrix=[]
myfunc = np.vectorize(lambda x: int(x))

for row in content:
    r = row.split(" ")
    r = myfunc(r)
    ult_matrix.append(r)
```

   **Check if the given matrix is ultrametric:**

```
def ultra(matrix):
    ultrametric = True
    length = len(matrix[0])
    for row_num, row in enumerate(matrix):  #Choose a row (node0)
        for c1 in range((row_num+1), length):       #  (node1)
            for c2 in range((row_num+1), length):   #  (node2)
                if(c1!=c2):   #exclude repetitions (eg. A,A)
                    a = row[c1]       #1st dist from row_num to c1
                    b = row[c2]       #2nd dist from row_num to c2
                    c = matrix[c1][c2] #3rd dist from c1 to c2
                    mini = min(a,b,c)
                    #check if 2 maximum distances are equal
                    sum_calc=((a+b+c)-mini)/2
                    if(sum_calc not in {a,b,c}):
                        ultrametric = False
                        return(ultrametric)
    return(ultrametric)

ultrametric = ultra(ult_matrix)
print("The matrix is ultrametric:", ultrametric)
```

   **Output**: The matrix is ultrametric: True
2. The running time of our script, as a function of the number n of species, is $O(n^3)$.
3. The best possible running time of an algorithm to test for an ultrametric distance matrix is $O(n^2)$.
4. Python code:
   **Preprocessing**:

```
import numpy as np
#read and process the distance matrics
with open("additive.txt") as f:
    contents = f.read()
content = contents.split("\n")[:-1]
add_matrix=[]
```

```
for row in content:
    r = row.split(" ")
    r = myfunc(r)
    add_matrix.append(r)
```

**Check if the given matrix is additive:**

```
def is_additive(matrix):
    additive = True
    length = len(matrix[0])
    for i in range(length):
        for j in range(i + 1, length):
            for k in range(j + 1, length):
                for l in range(length):
                    if l != i and l != j and l != k:
                        dist_i_j = matrix[i][j]
                        dist_i_k = matrix[i][k]
                        dist_i_l = matrix[i][l]
                        dist_j_k = matrix[j][k]
                        dist_j_l = matrix[j][l]
                        dist_k_l = matrix[k][l]
                        sum_1 = dist_i_j + dist_k_l
                        sum_2 = dist_i_k + dist_j_l
                        sum_3 = dist_i_l + dist_j_k

                        min_sum = min(sum_1, sum_2, sum_3)
                        sum_calc↩
                            = (sum_1 + sum_2 + sum_3 - min_sum) / 2
                        if sum_calc not in {sum_1, sum_2, sum_3}:
                            additive = False
                            return additive
    return additive

additive_result = is_additive(add_matrix)
print("The matrix is additive:", additive_result)
```

**Output**: The matrix is additive: True

5. The running time of our script, as a function of the number n of species, is $O(n^4)$.
6. The best possible running time of an algorithm to test for an additive distance matrix is $O(n^3)$.