# Scientific Computation
# Autumn 2024
# Review exercises

---

1. Consider the application of insertion sort to the list, `L = [15,10,16,9]`. How many comparisons will the algorithm require? List the pairs of integers that will be compared in the order that the comparisons take place.

2. Consider the following list ordered as a binary heap: `L = [0, 4, 3, 14, 10, 11, 9, 17, 16, 16, 13]`. How many comparisons will be required if 3 is inserted into the heap? List the pairs of integer that will be compared in the order that the comparisons take place.

3. You are given an *adjacency list*, `L`, for a directed graph with nodes numbered from 1 to $N$. Here, `L[i]` is a list, and if there is a link from node $i$ to node $j$, then $j$ will be in `L[i]`. Provide an efficient algorithm for computing the adjacency list for the *reverse* of the directed graph where any link from $i$ to $j$ is replaced with a link from $j$ to $i$. Briefly explain what the big-O cost of your algorithm is.

4. Consider the application of the Rabin-Karp method to find 4-character patterns in a gene sequence, $S$, which only contains Adenine, Cytosine, or Guanine. Let $S_i$ be the $i$th length-4 sequence in $S$ (containing `S[i:i+4]`). Provide a clear and concise explanation of how to compute the hash of $S_{i+1}$ given the hash of $S_i$. Hashes are not computed modulo a prime.

5. Consider the following algorithm for finding the shortest path from node $s$ to node $x$ in a weighted directed graph with some negative edge weights: add a large constant to each edge weight so that all edge weights are positive and then apply Dijkstra's algorithm setting $s$ as the source node. Explain if this is a generally valid method. Either provide a sketch of a proof establishing its correctness, or provide a counterexample.

6. The code below was generated by claude sonnet 3.5. Provide a clear and concise explanation of the problem the code is solving and the strategy being used to solve the problem.

```
import numpy as np
def code6(A, y0, t0, tf, N):
    """
    Parameters:
```

```
    A (numpy.ndarray): 2 x 2 matrix
    y0 (numpy.ndarray): 2 x 1 array
    t0 (float): Initial time
    tf (float): Final time
    N (int): Number of time steps

    """
    dt = (tf - t0) / N
    t = np.linspace(t0, tf, N+1)
    y = np.zeros((N+1, y0.shape[0]))
    y[0] = y0

    for i in range(1, N+1):
        y[i] = np.linalg.solve(np.eye(2) - dt * A, y[i-1])

    return y
```

7. Consider the function below:

```
import numpy as np
import matplotlib.pyplot as plt
def rwalk_new(Nt=200,M=100,display=False):

    X = np.zeros((Nt+1,M))
    R = np.random.choice([-1,0,1],size=(Nt,M))

    for i in range(Nt):
        X[i+1,:] = X[i,:] + R[i,:]

    Xave = X.mean(axis=1)
    Xsd = X.std(axis=1)

    if display:
        plt.figure()
        plt.plot(Xave,'k-',label='ave, comp')
        plt.plot(Xsd,'k--',label='sd, comp')
        i = np.arange(Nt+1)
        plt.plot(i,0*i,'b:',label='expected val, theory')
        plt.plot(i,np.sqrt(i),'r:',label='sd, theory')
        plt.legend()
```

Describe the problem that this code is solving. Carefully explain what changes, if any, are needed to the 3rd and 4th calls to `plt.plot`

8. In this question, you will analyze *func1* in the code below.

```
def merge(L,R):
    """Merge 2 sorted lists provided as input
    into a single sorted list
    """
    M = [] #Merged list, initially empty
    indL,indR = 0,0 #start indices
    nL,nR = len(L),len(R)

    #Add one element to M per iteration until an entire sublist
    #has been added
    for i in range(nL+nR):
        if L[indL]<R[indR]:
            M.append(L[indL])
            indL = indL + 1
            if indL>=nL:
                M.extend(R[indR:])
                break
        else:
            M.append(R[indR])
            indR = indR + 1
            if indR>=nR:
                M.extend(L[indL:])
                break
    return M


def func1(L_all):
    """Function to be analyzed for question 1
    Input: L_all: A list of N length-M lists. Each element of
    L_all is a list of integers sorted in ascending order.
    Example input: L_all = [[1,3],[2,4],[6,7]]
    """
    if len(L_all)==1:
        return L_all[0]
    else:
        L_all[-1] = merge(L_all[-2],L_all.pop())
        return func1A(L_all)
```

(a) Provide a brief, clear description of the functionality and correctness of `func1` (analysis of `merge` is not required, you may state results from lecture or elsewhere). Include a clear explanation of the function's output and the strategy the function uses to produce the output.

(b) Analyze the time complexity of `func1` and explain how the cost depends on the input. As part of your analysis, include an estimate of the big-O cost of the function (again, analysis of `merge` is not required, and you may simply state results from lecture).