Assignment 4

Himanshu, MDS202327

In [ ]:
```
# Importing Libraries
from random import random, randrange, seed
import time
from numpy import mean, std, sqrt, log
import sys

sys.setrecursionlimit(2**31 - 1)
```

In [ ]:
```
# Helper Functions
def selectionSort(L):
    n = len(L)
    if n < 1:
        return L
    for i in range(n):
        mpos = i
        for j in range(i + 1, n):
            if L[j] < L[mpos]:
                mpos = j
        (L[i], L[mpos]) = (L[mpos], L[i])
    return L


def insertionSort(L):
    n = len(L)
    if n < 1:
        return L
    for i in range(n):
        j = i
        while j > 0 and L[j] < L[j - 1]:
            L[j], L[j - 1] = L[j - 1], L[j]
            j = j - 1
    return L


def merge(A, B):
    (m, n) = (len(A), len(B))
    (C, i, j, k) = ([], 0, 0, 0)
    while k < m + n:
        if i == m:
            C.extend(B[j:])
            k = k + (n - j)
        elif j == n:
            C.extend(A[i:])
            k = k + (m - i)
        elif A[i] < B[j]:
            C.append(A[i])
            (i, k) = (i + 1, k + 1)
        else:
            C.append(B[j])
            (j, k) = (j + 1, k + 1)
    return C


def hybridMergeSort(A, cutoff):
    n = len(A)
    if n <= 1:
        return A
    if n <= cutoff:
        return insertionSort(A)
    L = hybridMergeSort(A[: n // 2], cutoff)
    R = hybridMergeSort(A[n // 2:], cutoff)
    B = merge(L, R)
    return B
```

```
def hybridRandQuickSort(L, l, r, cutoff):
    if r - l <= cutoff:
        return insertionSort(L[l:r])
    if r - l <= 1:
        return L
    randpivot = randrange(r - l)
    (L[l], L[l + randpivot]) = (L[l + randpivot], L[l])
    (pivot, lower, upper) = (L[l], l + 1, l + 1)

    for i in range(l + 1, r):
        if L[i] > pivot:
            upper = upper + 1
        else:
            (L[i], L[lower]) = (L[lower], L[i])
            (lower, upper) = (lower + 1, upper + 1)

    (L[l], L[lower - 1]) = (L[lower - 1], L[l])
    lower = lower - 1

    hybridRandQuickSort(L, l, lower, cutoff)
    hybridRandQuickSort(L, lower + 1, upper, cutoff)
    return L
```

In [ ]:

```
def function12(K, N, M, sortingAlgo):
    seed(12)
    mean_run_time = []
    std_deviation = []
    L = [[round(random() * 100, 3) for _ in range(N)] for _ in range(K)]
    for _ in range(M):
        timeVals = []
        for l in L:
            start = time.perf_counter()
            sortingAlgo(l)
            elapsed = time.perf_counter() - start
            timeVals.append(elapsed)
        mean_run_time.append(round(mean(timeVals), 5))
        std_deviation.append(round(std(timeVals), 5))
    return list(zip(mean_run_time, std_deviation))


def function3(K, N, M, sortingAlgo, cutoff):
    seed(34)
    mean_run_time = []
    std_deviation = []
    L = [[round(random() * 100, 3) for _ in range(N)] for _ in range(K)]
    for _ in range(M):
        timeVals = []
        for l in L:
            start = time.perf_counter()
            sortingAlgo(l, cutoff)
            elapsed = time.perf_counter() - start
            timeVals.append(elapsed)
        mean_run_time.append(round(mean(timeVals), 5))
        std_deviation.append(round(std(timeVals), 5))
    return list(zip(mean_run_time, std_deviation))


def function4(K, N, M, sortingAlgo, l, r, cutoff):
    seed(34)
    mean_run_time = []
    std_deviation = []
    L = [[round(random() * 100, 3) for _ in range(N)] for _ in range(K)]
    for _ in range(M):
        timeVals = []
        for lis in L:
            start = time.perf_counter()
            sortingAlgo(lis, l, r, cutoff)
            elapsed = time.perf_counter() - start
            timeVals.append(elapsed)
        mean_run_time.append(round(mean(timeVals), 6))
        std_deviation.append(round(std(timeVals), 6))
    return list(zip(mean_run_time, std_deviation))
```

In [ ]:
```
# Q1
function12(100, 5000, 5, selectionSort)
```

Out[ ]:
```
[(0.74852, 0.08998),
 (1.25346, 0.35363),
 (0.90271, 0.26732),
 (1.25925, 0.27655),
 (0.95346, 0.29903)]
```

In [ ]:
```
# Q2
function12(100, 5000, 5, insertionSort)
```

Out[ ]:
```
[(1.70091, 0.36835),
 (0.00062, 0.00019),
 (0.00064, 0.00022),
 (0.00063, 0.00024),
 (0.00068, 0.00023)]
```

In [ ]:
```
# Q3
for cutoff in [0, 10, 40, 50, 80, 90]:
    print(f"Cutoff: {cutoff}")
    print(function3(100, 50000, 5, hybridMergeSort, cutoff), "\n")
```

```
Cutoff: 0
[(0.25121, 0.05155), (0.29607, 0.06386), (0.28988, 0.03841), (0.24397, 0.04083), (0.1827, 0.03309)]

Cutoff: 10
[(0.18891, 0.03331), (0.21625, 0.04486), (0.20843, 0.04243), (0.20744, 0.04116), (0.22315, 0.04811)]

Cutoff: 40
[(0.16116, 0.02933), (0.1791, 0.04785), (0.23167, 0.04751), (0.23258, 0.04977), (0.23863, 0.04278)]

Cutoff: 50
[(0.35954, 0.06533), (0.33722, 0.07767), (0.26499, 0.04503), (0.28409, 0.06159), (0.23504, 0.05811)]

Cutoff: 80
[(0.24371, 0.0646), (0.29612, 0.06259), (0.27351, 0.06128), (0.28086, 0.05922), (0.27965, 0.05228)]

Cutoff: 90
[(0.26833, 0.04347), (0.23131, 0.04289), (0.31386, 0.06715), (0.30173, 0.06228), (0.23973, 0.05886)]
```

In [ ]:
```
# Q4
for cutoff in [0, 10, 40, 50, 80, 90]:
    print(f"Cutoff: {cutoff}")
    print(function4(100, 50000, 5, hybridRandQuickSort, 0, 1000, cutoff), "\n")
```

```
Cutoff: 0
[(0.002457, 0.000766), (0.00218, 0.000675), (0.002146, 0.00071), (0.002413, 0.000772), (0.002959, 0.0
00845)]

Cutoff: 10
[(0.00265, 0.001299), (0.00182, 0.000626), (0.002255, 0.001137), (0.002761, 0.001512), (0.001482, 0.0
00558)]

Cutoff: 40
[(0.002278, 0.000759), (0.001613, 0.000528), (0.001387, 0.000424), (0.001659, 0.000877), (0.001445,
0.000585)]

Cutoff: 50
[(0.003464, 0.001183), (0.002606, 0.000928), (0.001996, 0.000633), (0.002125, 0.000886), (0.00194, 0.
000882)]

Cutoff: 80
[(0.004571, 0.00145), (0.00354, 0.001153), (0.002398, 0.000736), (0.002003, 0.000576), (0.00184, 0.00
0551)]

Cutoff: 90
[(0.006672, 0.001929), (0.004584, 0.00158), (0.003724, 0.001363), (0.002816, 0.000802), (0.002269, 0.
000677)]
```