**Homework 1**

**GithubLink**: <https://github.com/naveenkumarkk/algorithmics>

***NOTE: FOR EXERCISES 2 AND 3, CODE, DATASET & RESULTS AND GRAPH HAVE BEEN INCLUDED IN THE ABOVE GIT HUB LINK AND ALSO ADDED IN THE ZIP FILE  
  
FILE: algorithmics\homework\hw-1\exercise2-3***

**Exercises 2 and 3:**

1. **Question**: Generate random integers/numbers in your choice of programming language and write a reusable code to generate a list/array of various input data sets varying in size and type of data. Generate cases for N integers (e.g. N in 1000, 10 000, 100 000, 1000 000 in a given range: positive integers from 0..(2\*\*8-1), 0..(2\*\*16-1), 0..(2\*\*32-1), 0..(2\*\*64-1) or in other words, 1 byte, 2 bytes, 4 bytes, 8 bytes, and signed integers in ranges -100..+100, -10000..+10000, -1 000 000 000..+1 000 000 000 ).  
   **Answer**:   
    1. Used Python language for implementation.  
    2. A ***random***  library is used to generate the random integers
2. **Question:** Figure out how to perform a timing experiment  
   **Answer**: By using the ***time*** library I have calculated the execution time of the program
3. **Question:**  Measure the time it takes to run as the input size increases or the data type changes.   
   **Answer**: I have executed the program 10 times with input of size 1byte,2byte,8byte,16 bytes, 32 bytes, 64 bytes, -100 to 100, -10000 to 10000 and -1000000000 to 1000000000
4. **Question:** Analyse runtime variability by running the code many times for each data (for every run, generate a new list/dataset). Does it take a different time to execute on the same input? Why? Write down why the algorithm takes a different time to execute on the same input size.  
   **Answer:**   
   Execution time differs based on various reasons, for the above program the execution time differs based on the following reason  
   1. System Load and Resource Availablity:   
    \* If the CPU is busy with another process, the above program gets less CPU time which will in turn result in slow execution speed.  
    \*Memory usage: If other processes are consuming large amounts of RAM, it will lead to memory contention which will result in slow execution speed.  
     
   2. Randomness in the program:  
    \* Since the program itself is based on random number generation, the execution time is directly based on how long the program takes time to make the random choices.
5. **Question:** Create an output summary table and illustrative plots showing the results of experiments from steps 3 and 4. How do you summarise such complex tables? Which plot should you use? Make sure to add labels to your axes and a title to your plots. Interpret what you see in the plots  
   **Answer:** I have included the Excel sheet under the file path algorithmics\homework\hw-1\exercise2-3\HomeWork 1 - Naveen.xlsx  
     
   I used a simple table to summarize the dataset, with the input size as the column and the minimum and maximum values of randomness as the rows. After executing the program, I recorded the execution time in the respective cells based on the dataset.

For visualization, I utilized a line graph. Since the program demonstrates linear progression, this type of graph provides a clear and detailed visual representation of the data.

Code:

import random

import time

def generateRandomList(sizeN,minValue,maxValue):

beginingOfCounter = time.perf\_counter()

result = [random.randint(minValue, maxValue) for \_ in range(sizeN)]

endCounter= time.perf\_counter()

print(f"Time difference: {endCounter - beginingOfCounter:0.4f} seconds")

***NOTE: FOR EXERCISES 4, CODE, DATASET & RESULTS AND GRAPH HAVE BEEN INCLUDED IN THE ABOVE GIT HUB LINK AND ALSO ADDED IN THE ZIP FILE  
  
FILE: algorithmics\homework\hw-1\exercise4***

**Exercise 4:**

**Question:** Measure a sorting algorithm that is part of your programming language's standard library. Briefly describe the algorithm and state its computational complexity (you might need to dig a bit of documentation for it). Follow similar steps to analyse the sorting algorithm as you did in EX 2/3.  
  
**Answer:**I am using Python language. The “Sorted” function is the inbuild sorting algorithm used to sort the array list. This function uses the “TimSort” method, which is a hybrid sorting algorithm. It's a stable balanced algorithm which is a mixture of “Merge sort” and “Binary Insertion Sort”. It takes advantage of “run” and also uses the "galloping method. Collecting the data into run stacks over each top of the run once the minimum merge criteria are met, it's merged.  
  
*TIME COMPLEXITY:  
1. Best Case: O(n)*

*2. Worst Case: O(n logn)*

| **Input** | **Minimum** | **Maximum** | **Execution Time (Seconds)** |
| --- | --- | --- | --- |
| 1000 | -1000000000 | 1000000000 | 0.00012737 |
| 10000 | -1000000000 | 1000000000 | 0.0011779 |
| 100000 | -1000000000 | 1000000000 | 0.0154952 |
| 1000000 | -1000000000 | 1000000000 | 0.2205046 |

Since the time complexity in the worst case is “n log(n)”, the overlay function I am using is  
 **T(x) = a.x.log(x)**where a is a constant  
  
To find the constant we are modifying the above formula to  
**a = T(x)/x.log(x)**

For x = 1000  
  
a = T(1000) / (1000 \* log(1000))

log(1000) = 6.907

a = 0.00012737/(1000 \* 6.907)  
a = 1.84407123 e-8

For x = 10000

a = T(10000) / (10000 \* log(10000))

log(10000) = 9.210

a= 0.0011779 / (10000 \* 9.210)

a= 1.27893594 e-8

For x = 100000

a= T(100000) / (100000 \* log(100000))

log(100000) = 11.513  
  
a= 0.0154952 /(100000 \* 11.513)

a= 1.34588726 e-8

For x = 1000000

a= T(1000000) / (1000000 \* log(1000000)

log(1000000) = 13.816

a= 0.2205046 / (1000000 \* 13.816)  
a= 1.59600898e-8

Averaging and creating the final function

a= ((1.27893594 + 1.84407123 + 1.34588726 + 1.5960089) / 4) e-8

a= 1.5162258325 e-8

a=1.516 \* 10^-8

My system can sort the 80000000 data in under a minute.

T(80000000 ) = (1.516 \* 10^-8) \* 80000000 \* log(80000000 )

T(80000000) = 1.516 \* 10 ^ 8 \* 80000000 \* 18.197537193

T(80000000) = 22.07 second  
  
  
  
**Exercise 5:**

**Question:** For each of the following sets of five functions, order them so that if fa appears before fb in your sequence then fa = O(fb). If fa = O(fb) and fb = O(fa) (meaning fa and fb could appear in either order), indicate this by enclosing fa and fb in a set with curly braces.  
  
**Answer:**

1. (f1=(logn)2023,f5=nlogn,f2=n2log(n2023),f3=n3,f4=2.023n)
2. (f5=nC3​,f2=n3,f3=nCn/2​,f1=2n,f4=n!)

**Exercise 6:**

**Question:** Our company needs to decide on which software library to use to process up to n = 10 \*\* 5 data records that exist in a database. The decision lies between two libraries:

* library A has a known processing time of TA = 0.3 \* n \* log(n) milliseconds
* library B has a known processing time of TB = log(n) + 5 \* n milliseconds

**1.** Which library is faster, according to the "Big-O" notation?

Library A:

O = 0.3 \* n \* log(n) = n\*log(n)  
 It grows linearly but has an additional logarithmic factor  
 Library B:

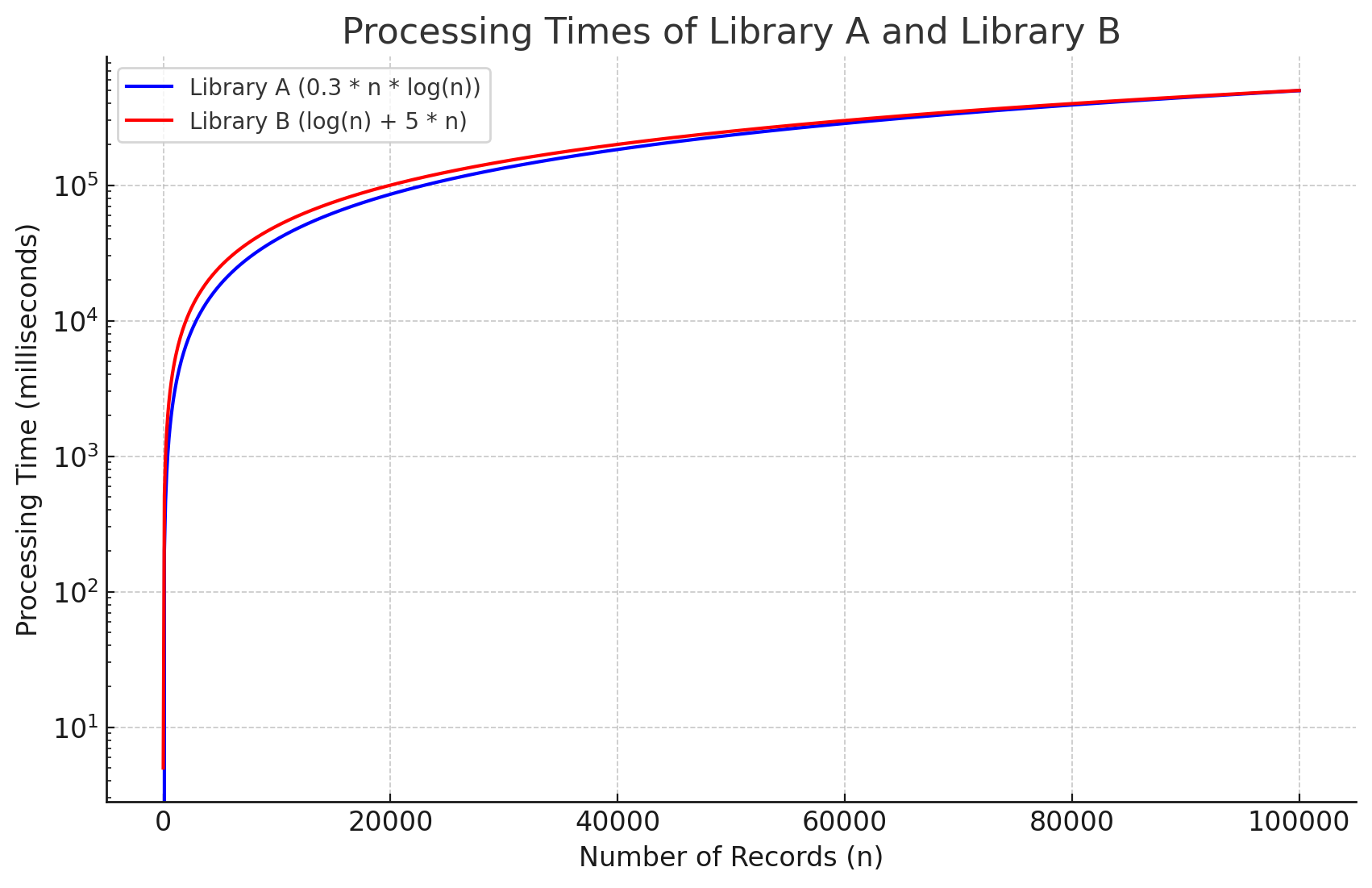
O = log(n) + 5 \* n = log(n)

It grows linearly  
  
 Library B is faster for large n as log(n) < n\*log(n)

2. Find out in which conditions the two libraries outperform each other. Use plots to show your findings.

**Answer**:

If “n” is small: library A is faster as n is the smaller component  
 If “n” is large: library B is faster



3. Finally, given the initial circumstances, choose the best library for the company to use.  
  
**Answer**:  
 Given data = 10\*\*5

Library B is faster for the data, it has lower processing time