Exercise 1

Q1:

Memoization: is the algorithm that some programming language use to store the current results so we can used this for later uses to speed up the future function calls.

Q3:

The code provided in question 2 is used to generate the Fibonacci function by using recursive method. The function recalls itself until it approaches the base cases and stop.

Q4:

Yes, the code is an example of divide-and-conquer algorithm. The way this code work is follow recursive method where they code into smaller branches and each branch keeps breaking into smaller branches until it reaches the base case.

Q5: The time complexity of Fibonacci sequence in recursive call is a complexity of O(2^n)

Q6:

import timeit

def memoize(k):

list\_memory = {}

def check\_function(l):

if l not in list\_memory:

list\_memory[l] = k(l)

return list\_memory[l]

return check\_function

def fibonacci(n):

if (n<2):

return 1

return (fibonacci(n-1) + fibonacci(n-2))

fibonacci =memoize(fibonacci)

print(time1)

def fibonacci2(n):

if (n<2):

return 1

return (fibonacci2(n-1) + fibonacci2(n-2))

Q7: The time complexity of memoize function is O(n) where n is the input of the function memorize. One the result of a specific input is computed, it will be store in the list\_memory if the list\_memory doesn’t have the value.

Q8:

import timeit

from matplotlib import pyplot as plt

import pandas as pd

def memoize(k):

list\_memory = {}

def check\_function(l):

if l not in list\_memory:

list\_memory[l] = k(l)

return list\_memory[l]

return check\_function

def fibonacci(n):

if (n<2):

return 1

return (fibonacci(n-1) + fibonacci(n-2))

fibonacci =memoize(fibonacci)

def fibonacci2(n):

if (n<2):

return 1

return (fibonacci2(n-1) + fibonacci2(n-2))

list\_optimize=[]

list\_origninal=[]

for i in range(36):

tm\_op=timeit.timeit(lambda:fibonacci(i), number =1)

list\_optimize.append(tm\_op)

tm2\_or=timeit.timeit(lambda:fibonacci2(i), number =1)

list\_origninal.append(tm2\_or)

list\_number=[]

for i in range (36):

list\_number.append(i)

print(list\_optimize)

print(list\_origninal)

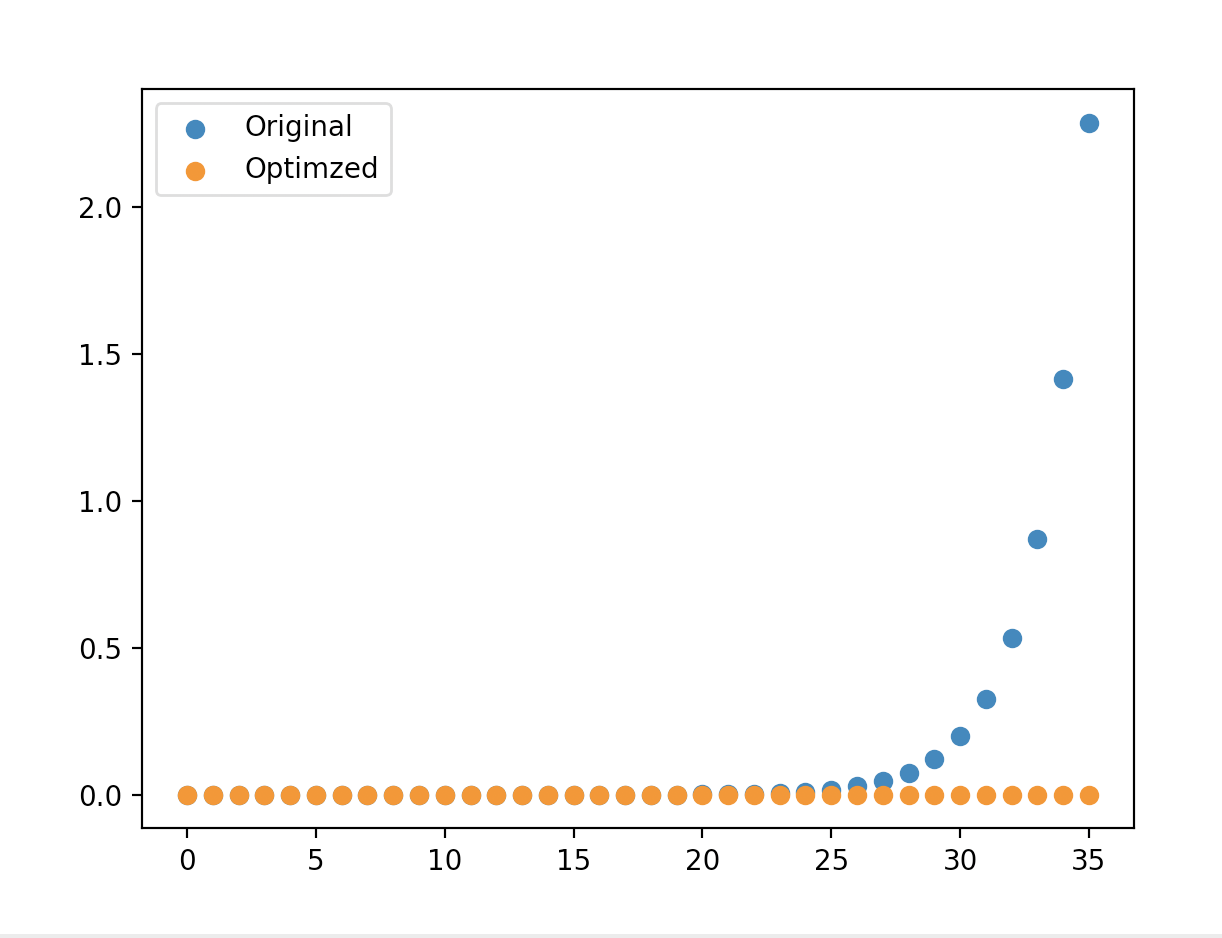
fix, ax=plt.subplots()

ax.scatter(list\_number,list\_origninal,label="Original")

ax.scatter(list\_number,list\_optimize,label="Optimzed")

plt.legend()

plt.show()



Q9:

As we can see clearly from the plot, the time of computing the fib(35) is way faster when using the memorization where the original one is getting slower every increment. Also the time complexities is match with what I states in question 5 and question 7 where the original method follow the O(2^n) complexity (the graph and slower 2 times each increment) and the memorization followed the O(n) complexity (the time is nearly same even is looking for fib(0) or fib(35).

Exercise 3

Q1:

Interpolation is better when the list is uniformly distributed where the estimated position of the target is gotten faster compared to binary search where we must divide and search for the element in smaller block. Also, the interpolation works more efficient if the space between each element is large, the estimation of target position faster than we must start from the middle of the list.

Q2:

If the data follows a different distribution that not uniformly distribution, the performance may be affected. So, if the data set is not followed uniform distributed, for instance the data follow the normal distribution, the space between each element is not guaranteed to space equally. Some space maybe small, some space may be large between elements which is affected the position of the target value that we are looking for. Also, as we know the normal distribution, the area between the mean value larger than the rest of the distribution which can cause the interpolation search repeatedly to search as the same portion of the list.

Q3:

If we want to modify the Interpolation Search to follow a different distribution, the position calculation part needs to be changed since the data is not uniformly distributed so we cannot use the same formula in the code. Therefore, we cannot use that formula for different distribution. If we want to find the estimate the position of the value in the normal distribution, we can use CDF (cumulative distribution function to estimative the position of the target value in the list. The formula is position = low + (((x - mean) / std) \*(high - low).

Q4:

A. The linear search can be the only option to use when the list has few elements, the list is an unordered list, the data contains duplicate values, or the list does not have a well-define distribution. The binary search or interpolation search required an order list to perform the search.

B. The case when Linear search will outperform both Binary search and Interpolation is when the size of list is small or unorder list. To address this issue, we can divide the problem into 2 cases, if the data small use the Linear Search. Otherwise, if the data is big, first we need to sort the array if it has not, and then search the data by using Binary Search or Interpolation. By dividing the problem into 2 cases, we will ensure that the algorithms are always efficient no matter the size of the data.