Discrete Computational Structure

Project 1: Binary/Linear Search

Graphs:

Chart, line chart

Description automatically generated

These graphs show the relationship between the theoretical and calculated values from our linear and binary searches. I was unable to get the Log base 2 theoretical graph to plot properly however, using a standard graphing calculator, I was able to surmise the approximate deviation from the calculated plot shown in the picture above. Starting off with the linear search theoretical plot, we can see that the expected output (orange) is not too dissimilar from the calculated values from my code (blue). I believe that with a better method and more refined values, I could get even closer to the expected values that the orange plot line demonstrates. As for the binary graph, again, using a standard graphing calculator, I found that the origin for the logarithmic graph on the calculator was different than that of my code. This is most likely due to the selection of values chosen as the samples as seen in the code. Slightly modifying the values used in the code would likely result in a better approximation of the logarithmic scale shown on the graph. I believe it is normal for calculated values and theoretical values to differ by a certain margin, however, I am not sure of what percentage deviation is allowable for this comparison. From the graphs I was able to get, it seems that the values from the code and the theoretical values do differ, but not such that it would render the code useless.

I have attached my code for reference, and will be submitting a separate file with the code as well.

import random

import matplotlib.pyplot as plt

import numpy as np

#Binary search function

def binary\_search(array, val):

#n is the length of the array

n = len(array)

#index i and j

i = 0

j = n - 1

#we must keep count of comparisons

cnt = 0

#this is the part of the code that actually searches throught the indexes

while (i <= j):

m = (i+j)/2

compare = array[i] - val

cnt = cnt + 1

#as soon as our value is found, we want to exit the loop

if compare == 0:

break

#if not found and below, then its on the left side of the binary

elif compare < 0:

j = m - 1

else:

#else on the right side of the binary

i = m + 1

return cnt

#create an array with our n values prewritten from prompt

test\_size = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

#create an empty array so we can store the averages and plot later

average\_array = []

#begin the loop to look for averages

for n in test\_size:

array = random\_array(n)

#we must sort the array

array = sorted(array)

cmp\_count = 0

sample\_count = 100

#calculation to find averages

for \_ in range(sample\_count):

val = random.randint(1, n + 3)

count = binary\_search(array, val)

cmp\_count = cmp\_count + count

avg = cmp\_count / sample\_count

#editing the blank array we made earlier in code with new found values

average\_array.append(avg)

# plot the graph

plt.plot(test\_size, average\_array)

plt.title('Binary Searching')

plt.ylabel('Average')

plt.xlabel('n')

plt.show()

#superimposing log\_2 graph

#x\_log = log(i \* base)

#y = []

#plt.plot(x\_log, y)

#plt.xlabel('x')

###############################################################################

# 2). function to create an array of random numbers of size 'n'

def random\_array(n):

array = []

#Going through the array in order to fill the blank array created above

while(len(array) < n):

num = random.randint(1, n + 1)

if num not in array:

#array[] = []

array.append(num)

return array

def linear\_search(array, x):

n = len(array)

i = 0

while(i < n and array[i] != x):

i = i + 1

return i + 1

print('Linear Searching')

#Repeating 2 - 4 for a test size of 10 n 10-100

test\_size = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# 7). Need an array of average values

avg\_array = []

#Using for loop to test all the values and find the averages

for n in test\_size:

array = random\_array(n)

cnt\_comp = 0

#cnt\_val = cnt\_comp + n

cnt\_sample = 100

#Performing the search using our function created above

for \_ in range(cnt\_sample):

var = random.randint(1, 2 + n + 1)

count = linear\_search(array, var)

#

cnt\_comp = cnt\_comp + count

average = cnt\_comp / cnt\_sample

avg\_array.append(average)

#Graphing results

plt.plot(test\_size, avg\_array)

plt.title('Linear Searching')

plt.xlabel('n')

plt.ylabel('Average')

plt.show

#Superimposed f(n) = n

x\_val = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

y\_val = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

plt.plot(x\_val, y\_val)

############################################################################