Question 1

import random

import matplotlib.pyplot as plt

import numpy as np

def insertion\_sort(arr):

comparisons = 0

for i in range(1, len(arr)):

key = arr[i]

j = i - 1

while j >= 0 and key < arr[j]:

arr[j+1] = arr[j]

j -= 1

comparisons += 1

arr[j+1] = key

return comparisons

# Generate arrays of varying sizes and sort them with insertion sort

n\_values = np.arange(30, 1001, 10)

comparison\_values = []

for n in n\_values:

my\_arr = [random.randint(1, 1000) for i in range(n)]

num\_comparisons = insertion\_sort(my\_arr)

comparison\_values.append(num\_comparisons)

# Plot the graph of number of comparisons versus input size

plt.plot(n\_values, comparison\_values, label='Insertion Sort')

# Plot the graph of nlogn for comparison

nlogn\_values = [n\*np.log2(n) for n in n\_values]

plt.plot(n\_values, nlogn\_values, label='nlogn')

plt.title('Insertion Sort Comparisons')

plt.xlabel('Input Size (n)')

plt.ylabel('Number of Comparisons')

plt.legend()

plt.show()



Question 1 (ii)

import random

import matplotlib.pyplot as plt

import numpy as np

def merge\_sort(arr):

comparisons = 0

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

comparisons += merge\_sort(left\_half)

comparisons += merge\_sort(right\_half)

i = j = k = 0

while i < len(left\_half) and j < len(right\_half):

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

comparisons += 1

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

return comparisons

# Generate arrays of varying sizes and sort them with Merge Sort

n\_values = np.arange(30, 1001, 10)

merge\_sort\_comparisons = []

for n in n\_values:

my\_arr = [random.randint(1, 1000) for i in range(n)]

num\_comparisons = merge\_sort(my\_arr)

merge\_sort\_comparisons.append(num\_comparisons)

# Generate a graph of the number of comparisons made by Merge Sort versus n

plt.plot(n\_values, merge\_sort\_comparisons, label="Merge Sort")

# Generate a graph of nlogn for comparison

nlogn\_values = n\_values \* np.log(n\_values)

plt.plot(n\_values, nlogn\_values, label="nlogn")

# Add axis labels and a legend

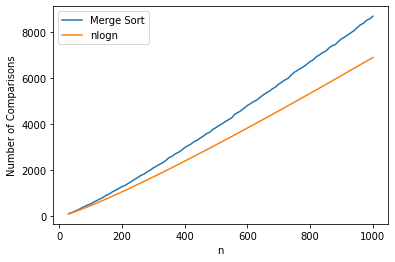
plt.xlabel("n")

plt.ylabel("Number of Comparisons")

plt.legend()

# Display the graph

plt.show()



Question 2

import random

import matplotlib.pyplot as plt

import numpy as np

def heapify(arr, n, i):

comparisons = 0

largest = i

left = 2 \* i + 1

right = 2 \* i + 2

if left < n and arr[i] < arr[left]:

largest = left

comparisons += 1

if right < n and arr[largest] < arr[right]:

largest = right

comparisons += 1

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i]

comparisons += 1

comparisons += heapify(arr, n, largest)

return comparisons

def heap\_sort(arr):

comparisons = 0

n = len(arr)

for i in range(n // 2 - 1, -1, -1):

comparisons += heapify(arr, n, i)

for i in range(n - 1, 0, -1):

arr[i], arr[0] = arr[0], arr[i]

comparisons += 1

comparisons += heapify(arr, i, 0)

return comparisons

# Generate arrays of varying sizes and sort them with Heap Sort

n\_values = np.arange(30, 1001, 10)

heap\_sort\_comparisons = []

for n in n\_values:

my\_arr = [random.randint(1, 1000) for i in range(n)]

num\_comparisons = heap\_sort(my\_arr)

heap\_sort\_comparisons.append(num\_comparisons)

# Generate a simple line graph of the number of comparisons made by Heap Sort versus n

fig, ax = plt.subplots()

ax.plot(n\_values, heap\_sort\_comparisons, label="Heap Sort")

# Generate a graph of nlogn for comparison

nlogn\_values = n\_values \* np.log(n\_values)

ax.plot(n\_values, nlogn\_values, label="nlogn")

# Add axis labels and a legend

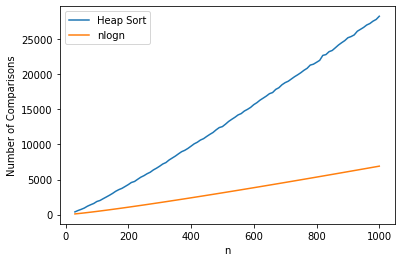
ax.set\_xlabel("n")

ax.set\_ylabel("Number of Comparisons")

ax.legend()

# Display the graph

plt.show()



Question 3

import random

import matplotlib.pyplot as plt

import numpy as np

def partition(arr, low, high):

pivot = arr[high]

i = low - 1

comparisons = 0

for j in range(low, high):

comparisons += 1

if arr[j] <= pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

comparisons += 1

arr[i+1], arr[high] = arr[high], arr[i+1]

comparisons += 1

return i+1, comparisons

def randomized\_partition(arr, low, high):

i = random.randint(low, high)

arr[i], arr[high] = arr[high], arr[i]

return partition(arr, low, high)

def quick\_sort(arr, low, high):

comparisons = 0

if low < high:

pi, comp1 = randomized\_partition(arr, low, high)

comparisons += comp1

comp2 = quick\_sort(arr, low, pi-1)

comparisons += comp2

comp3 = quick\_sort(arr, pi+1, high)

comparisons += comp3

return comparisons

# Generate arrays of varying sizes and sort them with Quick Sort

n\_values = np.arange(30, 1001, 10)

quick\_sort\_comparisons = []

for n in n\_values:

my\_arr = [random.randint(1, 1000) for i in range(n)]

num\_comparisons = quick\_sort(my\_arr, 0, n-1)

quick\_sort\_comparisons.append(num\_comparisons)

# Generate a simple line graph of the number of comparisons made by Quick Sort versus n

fig, ax = plt.subplots()

ax.plot(n\_values, quick\_sort\_comparisons, label="Quick Sort")

# Generate a graph of nlogn for comparison

nlogn\_values = n\_values \* np.log(n\_values)

ax.plot(n\_values, nlogn\_values, label="nlogn")

# Add axis labels and a legend

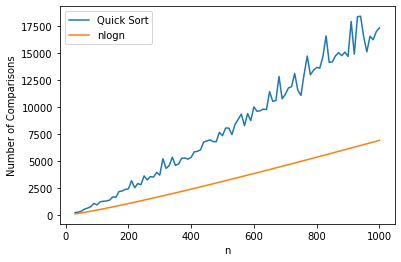
ax.set\_xlabel("n")

ax.set\_ylabel("Number of Comparisons")

ax.legend()

# Display the graph

plt.show()



Question 4

def radix\_sort(arr):

max\_val = max(arr)

exp = 1

while max\_val // exp > 0:

# Create buckets for each digit (0-9)

buckets = [0] \* 10

# Count the number of times each digit appears

for num in arr:

digit = (num // exp) % 10

buckets[digit] += 1

# Modify the count to represent the index of each digit in the sorted array

for i in range(1, 10):

buckets[i] += buckets[i-1]

# Sort the array based on the digit at the current exponent

sorted\_arr = [0] \* len(arr)

for num in reversed(arr):

digit = (num // exp) % 10

index = buckets[digit] - 1

sorted\_arr[index] = num

buckets[digit] -= 1

# Update the original array with the sorted array

for i in range(len(arr)):

arr[i] = sorted\_arr[i]

exp \*= 10

return arr

arr = [170, 45, 75, 90, 802, 24, 2, 66]

sorted\_arr = radix\_sort(arr)

print(sorted\_arr)

Output

[2, 24, 45, 66, 75, 90, 170, 802]

Question 5

def bucket\_sort(arr):

# Determine the range of values in the input array

min\_val = min(arr)

max\_val = max(arr)

bucket\_size = 10

# Determine the size of each bucket

bucket\_range = (max\_val - min\_val + 1) / bucket\_size

buckets = [[] for \_ in range(bucket\_size)]

# Assign each element to its corresponding bucket

for num in arr:

bucket\_index = int((num - min\_val) // bucket\_range)

buckets[bucket\_index].append(num)

# Sort each bucket

for i in range(bucket\_size):

buckets[i].sort()

# Combine the sorted buckets into a single array

sorted\_arr = []

for bucket in buckets:

sorted\_arr += bucket

return sorted\_arr

arr = [4, 10, 1, 5, 2, 7, 8, 3, 6, 9]

sorted\_arr = bucket\_sort(arr)

print(sorted\_arr)

Output

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Question 6

import random

def randomized\_select(arr, p, r, i):

if p == r:

return arr[p]

q = randomized\_partition(arr, p, r)

k = q - p + 1

if i == k:

return arr[q]

elif i < k:

return randomized\_select(arr, p, q-1, i)

else:

return randomized\_select(arr, q+1, r, i-k)

def randomized\_partition(arr, p, r):

i = random.randint(p, r)

arr[r], arr[i] = arr[i], arr[r]

return partition(arr, p, r)

def partition(arr, p, r):

x = arr[r]

i = p - 1

for j in range(p, r):

if arr[j] <= x:

i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i+1], arr[r] = arr[r], arr[i+1]

return i + 1

arr = [3, 7, 9, 5, 1, 10, 6, 8, 2, 4]

k = 4

elem = randomized\_select(arr, 0, len(arr)-1, k)

print("The", k, "smallest element is:", elem)

Output

The 4 smallest element is: 4

Question 7

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list)

def add\_edge(self, u, v):

self.graph[u].append(v)

def bfs(self, start):

visited = set()

queue = []

visited.add(start)

queue.append(start)

while queue:

s = queue.pop(0)

print(s, end=' ')

for neighbor in self.graph[s]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

g = Graph()

g.add\_edge(0, 1)

g.add\_edge(0, 2)

g.add\_edge(1, 2)

g.add\_edge(2, 0)

g.add\_edge(2, 3)

g.add\_edge(3, 3)

print("BFS Traversal starting from vertex 2:")

g.bfs(2)

Output

BFS Traversal starting from vertex 2:

2 0 3 1

Question 8

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list)

def add\_edge(self, u, v):

self.graph[u].append(v)

def dfs\_util(self, v, visited):

visited.add(v)

print(v, end=' ')

for neighbor in self.graph[v]:

if neighbor not in visited:

self.dfs\_util(neighbor, visited)

def dfs(self, start):

visited = set()

self.dfs\_util(start, visited)

g = Graph()

g.add\_edge(0, 1)

g.add\_edge(0, 2)

g.add\_edge(1, 2)

g.add\_edge(2, 0)

g.add\_edge(2, 3)

g.add\_edge(3, 3)

print("DFS Traversal starting from vertex 2:")

g.dfs(2)

Output

DFS Traversal starting from vertex 2:

2 0 1 3