Objectives of this assignment:

* to explore time complexity and “real time” of a well-known algorithm

Use This File

* USE THIS FILE AS THE STARTING DOCUMENT YOU WILL TURN IN. **DO NOT DELETE ANYTHING FROM THIS FILE:** JUST **INSERT** EACH ANSWER **RIGHT AFTER ITS QUESTION/PROMPT**.
* IF USING HAND WRITING (STRONGLY DISCOURAGED), **USE THIS FILE** BY CREATING SUFFICIENT SPACE AND WRITE IN YOUR ANSWERS.
* FAILING TO FOLLOW TURN IN DIRECTIONS /GUIDELINES WILL COST **A 30% PENALTY.**

What you need to do: (Use this file to INSERT your answers as indicated below)

1. Implement the ***QuickSort*** algorithm to sort an array. (See Lecture or Textbook for the *QuickSort* algorithm)
2. Collect the execution time T(n) as a function of n
3. Plot the functions , and as a function of n on three separate graphs. We assume that:
4. In Module 5, we establish that the average running time T(n) of *QuickSort* is Θ(n.log(n)). Discuss T(n) in light of the graphs you plotted above. Use the prediction techniques learned in M1: Programming Assignment (See Early questions trying to infer the shape of the plots)

**Objective**: The objective of this programming assignment is to implement in your preferred language the QuickSort algorithm to sort a list of numbers. We are interested in exploring the relationship between the time complexity and the “real time”. For this exploration, you will collect the execution time T(n) as a function of n and plot the , and on the same graph (*If you cannot see clearly the shape of the plots, feel free to separate plots.*). Try to predict ahead the shapes of , and to check whether your plots are correct. Finally, discuss your results.

**Program to implement**

collectData()

Generate an array G of **HUGE** length L (as huge as your language allows) with **random** values capped at some max value (as supported by your chosen language). Try to reach at least the value L for 250,000.

for n = 7,500 to L (with step 2,500)

copy in Array A **n** first values from Array G // **(declare Array A only ONCE out of the loop)**

Take current time ***Start*** // We time the sorting of Array A of length n

// (**Use nanoseconds resolution if possible**)

QuickSort(A,0,n-1)

Take current time ***End*** // ***T(n) = End - Start*** (**Use nanoseconds**)

Store the value n and the values , and in a file **F** where T(n) is the execution time

**Advice:**

**1)** The pseudocode assumes arrays that start with index 1. So, an array A with n elements is an array A[1], A[2]..., A[n-1], A[n]. With most programming languages, an array A with n elements is an array A[0], A[2]..., A[n-1], A[n-1]. When implementing pseudocode that uses some array A with elements, I advise you to declare an array with elements and just ignore (not use) A[0]. This way, you can directly implement the algorithm without worrying about indices changes.

**2)** When plotting, **ignore the first values of n= 1000, to 5000**. When a program starts, there will be some overhead execution time not related to the algorithms. That overhead may skew T(n).

**Data Analysis**

Use any plotting software (e.g., Excel) to plot the values , and in File F as a function of n. File F is the file produced by the program you implemented. Discuss your results based on the plots. (**Hint**: is T(n) closer to K., or K. where K is a constant? See M1: Programming Assignment).

Answer where indicated below. Recall that answers must be well written, documented, justified, and presented to get full credit.

1. (25 points) Implement the ***QuickSort*** algorithm to sort an array. (See Appendix for the *QuickSort* algorithm)

.... answer here

I chose to program the QuickSort algorithm in python for this programming assignment, as it is my other course for the term. I had started a project file that contains this file and one other file but I believe you should only need this one .py file in order to compile and run.

import random

import time

import csv

import math

# Jonathan Elder

# CPSC 3270 Programming Assignment 2

# quicksort\_analyzer.py

class QuickSortAnalyzer:

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

self.data\_file = 'data.csv' # Filename for storing collected data

self.max\_val = 1000 # Maximum value for random elements in arrays

self.L = 250000 # Maximum length of arrays

def partition(self, arr, p, r):

"""

Perform partitioning step of the QuickSort algorithm.

:param arr: List to be partitioned.

:param p: Starting index of the partition.

:param r: Ending index of the partition.

:return: Index of current p element after partitioning.

"""

x = arr[r] # Pivot element

i = p - 1 # Index of smaller element

for j in range(p, r):

if arr[j] <= x:

i += 1

arr[i], arr[j] = arr[j], arr[i] # Swap elements

arr[i + 1], arr[r] = arr[r], arr[i + 1] # Swap pivot to correct position

return i + 1

def quicksort(self, arr, p, r):

"""

Perform the QuickSort algorithm recursively.

:param arr: List to be sorted.

:param p: Starting index of the array/subarray.

:param r: Ending index of the array/subarray.

"""

if p < r:

q = self.partition(arr, p, r)

self.quicksort(arr, p, q - 1) # Sort left subarray.

self.quicksort(arr, q + 1, r) # Sort right subarray.

def collect\_data(self):

"""

Collect execution data for sorting arrays of different sizes using QuickSort.

"""

print("Collecting data...")

with open(self.data\_file, mode='w') as file:

writer = csv.writer(file)

writer.writerow(['n', 'T(n)', 'T(n)/g\_1(n)', 'T(n)/g\_2(n)', 'T(n)/g\_3(n)'])

G = [random.randint(0, self.max\_val) for \_ in range(self.L)]

for n in range(7500, self.L + 1, 2500):

A = G[:n]

start\_time = time.time() # Use time() instead of time\_ns()

self.quicksort(A, 0, len(A) - 1)

end\_time = time.time() # Use time() instead of time\_ns()

execution\_time = end\_time - start\_time

g1 = 10 \* math.sqrt(n)

g2 = 2 \* n \*\* 2

g3 = 5 \* n \* math.log(n)

writer.writerow([n, execution\_time, execution\_time / g1, execution\_time / g2, execution\_time / g3])

print("Data collection completed.")

def demo\_sorting(self):

"""

Demonstrate sorting of a random array with 10 elements using QuickSort.

"""

print("Demonstrating sorting...")

# Generate a random array with 10 elements

original\_array = [random.randint(0, 100) for \_ in range(10)]

# Print the original array

print("Original array:", original\_array)

# Sort the array using QuickSort

self.quicksort(original\_array, 0, len(original\_array) - 1)

# Print the sorted array

print("Sorted array:", original\_array)

print("Sorting demonstration completed.")

# This will create and save the usage data needed for the assignment/graphs.

analyzer = QuickSortAnalyzer()

analyzer.collect\_data()

analyzer.demo\_sorting()

Actions to complete:

1) ssh into a **Tux** machine

2) clear the screen on the Tux machine (use the command clear)

3) Display the current date (use the command date)

4) Compile your program

5) Execute your program showing that your algorithm sorts correctly an array with 10 elements. We must see the original array and the sorted array.

6) Take a screenshot of the Tux terminal and insert here (below). Your screenshot should look like this template screenshot (we should see the **username**, the **date**, the **commands typed**, and the **results**): a **50 points penalty will be applied if there is no screenshot or if the screenshot does not include ALL the required information. Screenshot should be as readable as the screenshot below. If this task is not completed on a Tux machine, no credit will be awarded.**



.... (10 points for the screenshot if your program works and produces correct data) Insert here YOUR screenshot with Tux terminal. Without screenshot with all required information 50 points will be taken off.

...(15 points if screenshot **PLUS** program works and produces correct data)

Screenshot should include everything needed.

A screenshot of a computer

Description automatically generated

2. (10 points) Collect the execution time T(n) as a function of n. Record the values n, T(n), , and in a csv (comma-separated-values) file.

Turn in this csv file with your submission

I included the data.csv file from running the program from the TUX machine in my assignment, when I turned it in.

3. (3x15 points) Plot the functions , and as a function of n on three separate graphs (15 points per graph)*.*

Insert here the three graphs/plots

A graph with a line drawn on it

Description automatically generated

A graph with a line

Description automatically generated

A graph with a green line

Description automatically generated

4. (20 points) In Module 4, we will establish that the running time T(n) of *QuickSort* is Θ(n.log(n)).

Discuss here T(n) in light of the graphs you plotted above. Use the prediction techniques learned in M1: Programming Assignment (See Early questions trying to infer the shape of T(n) and determine the asymptotic growth). Discuss whether your plots confirm what we will learn in Module M4, i.e., that the running time T(n) of *QuickSort* is Θ(n.log(n)).

Answer/elaborate/Justify.

As n increases, the execution as described as T(n) also increases, which is the expected general output for when sorting larger arrays. When you compare the graphs and ratios of , and with the actual execution time of T(n), it becomes obvious that that which is equal to 5n.log(n) are the closest to 1 when compared to the other two ratios. As n increases for this function, it seems to stabilize around a consistent value, which would indicate that it has a similar growth rate as the T(n). Also it seems the growth rate of T(n) is slightly faster than n.log(n) but slower than n2 for very large values of n. This shows what we will see in module 5, I am to assume, with the idea that the quicksort algorithm is a tight bound to n.log(n).

**What you need to turn in:**

* Electronic copy of your source program of *collectData* program
* Electronic copy of the csv file recording the values n, T(n), , and
* Electronic copy of this file (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.

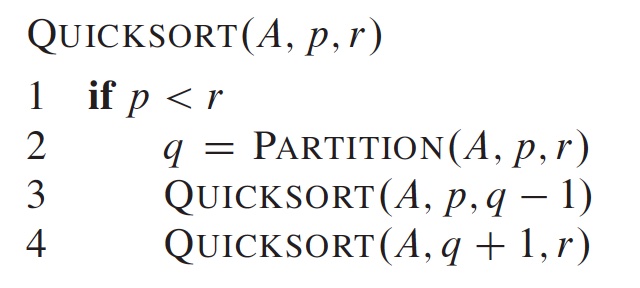
**Grading**

* See points distribution assigned to each task/question

Appendix: QuickSort Algorithm.

At this stage, you do NOT need to fully understand QuickSort (It will be presented and explained in Module 5)). Implement QuickSort exactly the way it is described below. Read about it.

**Appendix (Algorithm To Implement)**

****

