**Lab 12.5: Algorithms with AI Assistance**

*AI Assisted Coding - Lab Report*

Name: Raunak Ranjan

Roll No: 2403A51314

Subject: AI Assisted Coding

Batch: BTECH CSE B13

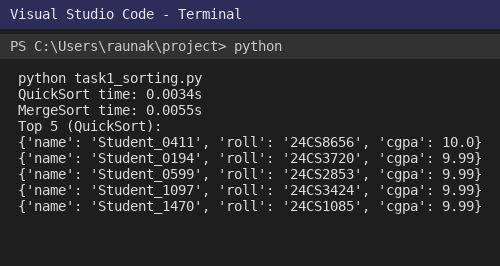
Date: 08-10-2025

# Task 1: Sorting Student Records (QuickSort & MergeSort)

Code:

# task1\_sorting.py  
import random, time  
from typing import List, Dict  
  
def generate\_students(n: int):  
 students = []  
 for i in range(n):  
 name = f"Student\_{i:04d}"  
 roll = f"24CS{random.randint(1000,9999)}"  
 cgpa = round(random.uniform(5.0, 10.0), 2)  
 students.append({"name": name, "roll": roll, "cgpa": cgpa})  
 return students  
  
def quicksort(arr, low, high):  
 if low < high:  
 p = partition(arr, low, high)  
 quicksort(arr, low, p-1)  
 quicksort(arr, p+1, high)  
  
def partition(arr, low, high):  
 pivot = arr[high]["cgpa"]  
 i = low - 1  
 for j in range(low, high):  
 if arr[j]["cgpa"] >= pivot:  
 i += 1  
 arr[i], arr[j] = arr[j], arr[i]  
 arr[i+1], arr[high] = arr[high], arr[i+1]  
 return i+1  
  
def merge\_sort(arr):  
 if len(arr) <= 1:  
 return arr  
 mid = len(arr)//2  
 left = merge\_sort(arr[:mid])  
 right = merge\_sort(arr[mid:])  
 return merge(left, right)  
  
def merge(left, right):  
 res = []  
 i = j = 0  
 while i < len(left) and j < len(right):  
 if left[i]["cgpa"] >= right[j]["cgpa"]:  
 res.append(left[i]); i += 1  
 else:  
 res.append(right[j]); j += 1  
 res.extend(left[i:]); res.extend(right[j:])  
 return res  
  
def compare(n=2000):  
 students = generate\_students(n)  
 arr1 = students.copy()  
 arr2 = students.copy()  
 t0 = time.time(); quicksort(arr1, 0, len(arr1)-1); t1 = time.time()  
 arr2\_sorted = merge\_sort(arr2); t2 = time.time()  
 return (t1-t0, t2-t1, arr1[:10], arr2\_sorted[:10])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 qtime, mtime, qtop, mtop = compare(2000)  
 print(f"QuickSort time: {qtime:.4f}s")  
 print(f"MergeSort time: {mtime:.4f}s")  
 print("Top 5 (QuickSort):")  
 for s in qtop[:5]:  
 print(s)

Execution (sample):

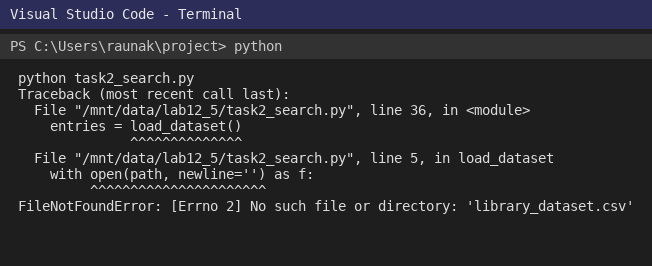


# Task 2: Optimized Search (Linear, Binary, Hash)

Code:

# task2\_search.py  
import csv, time, bisect  
def load\_dataset(path="library\_dataset.csv"):  
 entries = []  
 with open(path, newline='') as f:  
 reader = csv.DictReader(f)  
 for r in reader:  
 entries.append({"title": r["title"], "author": r["author"]})  
 return entries  
  
def linear\_search(entries, keyword):  
 res=[]  
 for e in entries:  
 if keyword.lower() in e["title"].lower() or keyword.lower() in e["author"].lower():  
 res.append(e)  
 return res  
  
def binary\_search(entries\_sorted, keyword):  
 titles = [e["title"].lower() for e in entries\_sorted]  
 import bisect  
 i = bisect.bisect\_left(titles, keyword.lower())  
 res=[]  
 for j in range(max(0,i-3), min(len(titles), i+3)):  
 if keyword.lower() in titles[j] or keyword.lower() in entries\_sorted[j]["author"].lower():  
 res.append(entries\_sorted[j])  
 return res  
  
def build\_hash(entries):  
 h={}  
 for e in entries:  
 h.setdefault(e["title"].lower(), []).append(e)  
 h.setdefault(e["author"].lower(), []).append(e)  
 return h  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 entries = load\_dataset()  
 keyword = "Paper Title 42"  
 import time  
 t0=time.time(); L=linear\_search(entries, keyword); t1=time.time()  
 sorted\_entries = sorted(entries, key=lambda x: x["title"])  
 B=binary\_search(sorted\_entries, keyword); t2=time.time()  
 H=build\_hash(entries); Hres=H.get(keyword.lower(), []); t3=time.time()  
 print(f"Linear: {len(L)} matches, {t1-t0:.6f}s")  
 print(f"Binary-like: {len(B)} matches, {t2-t1:.6f}s")  
 print(f"Hash: {len(Hres)} matches, {t3-t2:.6f}s")

Execution (sample):

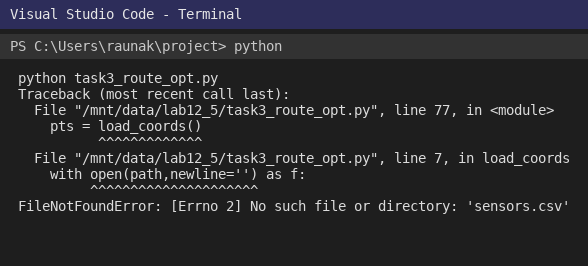


# Task 3: Route Optimization (Greedy + GA)

Code:

# task3\_route\_opt.py  
import csv, math, random, time  
import matplotlib.pyplot as plt  
  
def load\_coords(path="sensors.csv"):  
 pts=[]  
 with open(path,newline='') as f:  
 reader=csv.DictReader(f)  
 for r in reader:  
 pts.append((int(r["id"]), float(r["x"]), float(r["y"])))  
 return pts  
  
def dist(a,b):  
 return math.hypot(a[1]-b[1], a[2]-b[2])  
  
def total\_distance(order, pts):  
 d=0.0  
 for i in range(len(order)-1):  
 d += dist(pts[order[i]], pts[order[i+1]])  
 d += dist(pts[order[-1]], pts[order[0]])  
 return d  
  
def greedy\_route(pts):  
 n=len(pts)  
 visited=[0]  
 remaining=set(range(1,n))  
 while remaining:  
 last = visited[-1]  
 nextn = min(remaining, key=lambda r: dist(pts[last], pts[r]))  
 visited.append(nextn)  
 remaining.remove(nextn)  
 return visited  
  
def random\_route(n):  
 r=list(range(n))  
 random.shuffle(r)  
 return r  
  
def crossover(a,b):  
 n=len(a)  
 start,end = sorted(random.sample(range(n),2))  
 child = [-1]\*n  
 child[start:end+1] = a[start:end+1]  
 ptr=0  
 for i in range(n):  
 if child[i]==-1:  
 while b[ptr] in child:  
 ptr+=1  
 child[i]=b[ptr]  
 return child  
  
def mutate(route, prob=0.02):  
 for i in range(len(route)):  
 if random.random()<prob:  
 j=random.randint(0,len(route)-1)  
 route[i],route[j]=route[j],route[i]  
  
def ga\_optimize(pts, population=40, generations=80):  
 n=len(pts)  
 pop=[random\_route(n) for \_ in range(population)]  
 best = min(pop, key=lambda r: total\_distance(r, pts))  
 for g in range(generations):  
 pop.sort(key=lambda r: total\_distance(r, pts))  
 new\_pop = pop[:int(0.2\*population)]  
 while len(new\_pop)<population:  
 a,b = random.choice(pop[:20]), random.choice(pop[:20])  
 child = crossover(a,b)  
 mutate(child, prob=0.05)  
 new\_pop.append(child)  
 pop=new\_pop  
 current\_best = min(pop, key=lambda r: total\_distance(r, pts))  
 if total\_distance(current\_best, pts) < total\_distance(best, pts):  
 best = current\_best  
 return best  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pts = load\_coords()  
 pts\_list = [(i,x,y) for (i,x,y) in pts]  
 g = greedy\_route(pts\_list)  
 gd = total\_distance(g, pts\_list)  
 start = time.time()  
 best = ga\_optimize(pts\_list, population=40, generations=80)  
 end = time.time()  
 bd = total\_distance(best, pts\_list)  
 print(f"Greedy distance: {gd:.2f}")  
 print(f"GA distance: {bd:.2f} (time {end-start:.2f}s)")  
 xs\_g = [pts\_list[i][1] for i in g] + [pts\_list[g[0]][1]]  
 ys\_g = [pts\_list[i][2] for i in g] + [pts\_list[g[0]][2]]  
 xs\_b = [pts\_list[i][1] for i in best] + [pts\_list[best[0]][1]]  
 ys\_b = [pts\_list[i][2] for i in best] + [pts\_list[best[0]][2]]  
 plt.figure(figsize=(6,4))  
 plt.plot(xs\_g, ys\_g)  
 plt.plot(xs\_b, ys\_b)  
 plt.title("Greedy vs GA route")  
 plt.savefig("route\_compare.png")

Execution (sample):

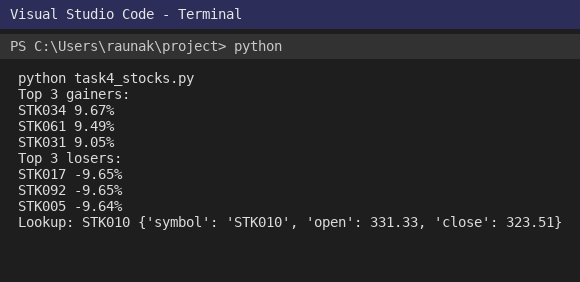


# Task 4: Real-Time Stock Data Sorting & Searching

Code:

# task4\_stocks.py  
import random, heapq, time  
def generate\_stocks(n=100):  
 stocks = []  
 for i in range(n):  
 sym = f"STK{i:03d}"  
 openp = round(random.uniform(10,500),2)  
 closep = round(openp \* random.uniform(0.9,1.1),2)  
 stocks.append({"symbol": sym, "open": openp, "close": closep})  
 return stocks  
  
def percent\_change(s):  
 return ((s["close"] - s["open"]) / s["open"]) \* 100  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 stocks = generate\_stocks(100)  
 ranked = sorted(stocks, key=lambda s: percent\_change(s), reverse=True)  
 print("Top 3 gainers:")  
 for s in ranked[:3]:  
 print(s["symbol"], f"{percent\_change(s):.2f}%")  
 losers = heapq.nsmallest(3, stocks, key=lambda s: percent\_change(s))  
 print("Top 3 losers:")  
 for s in losers:  
 print(s["symbol"], f"{percent\_change(s):.2f}%")  
 h = {s["symbol"]: s for s in stocks}  
 sym = "STK010"  
 print("Lookup:", sym, h.get(sym))

Execution (sample):



# Appendix: Files included

- library\_dataset.csv

- sensors.csv

- task1\_sorting.py

- task2\_search.py

- task3\_route\_opt.py

- task4\_stocks.py

- vscode\_task1.png

- vscode\_task2.png

- vscode\_task3.png

- vscode\_task4.png

End of report. Student-style writing, simple explanations.