BER2013 Algorithm Design and Analysis

Student Information System

Asst. Prof. Dr. Deiva Sigamani

Due date: 27th August

Student Name	Student ID
Tuba Ahmad	1002268808
Mohammed Tariq	1002371596
Hamzeh Kareh	1002372768
Raghd Abdullah	1002268809
Malak Mohammed Fouad	1002162667

Table of Contents

1. List of Abbreviations..... 2
2. Abstract.....2
3. Introduction.....3
4. Problem Statement and Objective.....3
5. System Design and Algorithm Description.....5
Figure 1: Main Interface for student registration.....5
5.1. Architecture (Block Diagram)..... 5
5.2. Data Structures Used and Why.....6
5.3. Core Algorithms.....6
5.4. Justification.....6
6. Implementation Details..... 8
Figure 2: Academics tab after enrolling a subject and adding a grade..... 8
6.1. Language & Libraries..... 8
6.2. Modules.....8
6.3. Functional Walkthrough.....9
7. Testing and Evaluation..... 10
Figure 3: Error after adding the same ID..... 10
Figure 4: Search by an existing ID vs. non-existing ID.....10
7.1. Test Cases.....11
8. Result and Analysis.....14
8.1. Complexity Performance..... 14
8.2. Accuracy & Efficiency..... 14
8.3. Challenges Faced..... 14
8.4. Limitations & Future Work.....14
9. Conclusion..... 15
10. References.....15
11. Appendix..... 15

Table of Figures

Figure 1: Main Interface for student registration**.....5**
Figure 2: Academics tab after enrolling a subject and adding a grade**.....8**
Figure 3: Error after adding the same ID**.....10**
Figure 4: Search by an existing ID vs. non-existing ID**.....10**

1. List of Abbreviations

BST – Binary Search Tree
DP – Dynamic Programming
GPA – Grade Point Average
FIFO – First-In-First-Out
LIFO – Last-In-First-Out
GUI – Graphical User Interface

2. Abstract

The objective of this project is to design and implement a student management system that can efficiently organize and handle academic records, including student registration, grades tracking, attendance monitoring, and automated report generation. The system was developed with the aim of combining theoretical knowledge of algorithms and data structures with practical application, resulting in a program that is both functional and educationally meaningful.

To achieve this, the project incorporates several fundamental data structures and algorithms. Stacks were utilized in the management of undo operations for grades, ensuring that the last entered grade could be removed when necessary. Queues were applied in the handling of attendance records, maintaining a first-in, first-out process that reflects the chronological order of attendance. Linked lists were employed to store subject enrollments dynamically, providing flexibility for insertion and deletion operations. A binary search tree (BST) was implemented to manage and search student records efficiently, particularly when dealing with larger datasets, enabling lookups with logarithmic time complexity.

The system was built using Python as the primary programming language, with tools such as the VS Code editor for development, Tkinter for the graphical user interface, and a local JSON file for persistent data storage. Together, these components enabled the integration of both data management and user interaction.

The resulting program demonstrates core object-oriented programming principles, including abstraction, encapsulation, and polymorphism. It provides a straightforward yet complete solution to

the problem of managing student records. Testing confirmed the system's accuracy and efficiency, making it a valuable platform for both academic learning and practical application.

3. Introduction

Data structures and algorithms lay the foundation when it comes to problem solving in Computer Science. Search functionality for programs is an example of how and where data structures and algorithms greatly optimise programming. Without the functionalities of Trees and Hash tables, a search that takes milliseconds could take minutes. Search algorithms like A* and Breath-First-Search (BFS) efficiently finds the shortest paths which improves user functionality, reduces cost, and makes for better and faster apps. Priority queues and heaps allow for system scalability by handling large quantities of data. Such examples are why data structures and algorithms are so prevalent and baked into Python and Programming in general.

This project's objective is to build a record keeping system for admin and school staff to use which keeps a record of student grades, registration, attendance, and produces reports. We have achieved this result using a variety of data structures and algorithms in the Python language such as: Singly Linked List, Stack (LIFO), Queue (FIFO), Binary Search Tree (BST), Python list, Python dict, Tuple and BST

Over the course of this project, our team has learned some key aspects of Python programming and algorithms such as:

- Learning the role of data structures in real world applications
- Implement Search and Sorting algorithms
- Practicing algorithmic problem solving
- Strengthening programming skills
- Learning error handling and debugging skills
- Teamwork and software design skills

This project uses the Python language with its libraries such as dataclasses, typing (List, Dict, Optional, Tuple, Generic, TypeVar, Iterator), json, os, datetime, and collections(Deque).

4. Problem Statement and Objective

Problem: Process student data (ID, Name, Department, Gender, Year, Subjects, Grades, and Attendance) to provide effective insert, lookup, sort and report functions.

Inputs: Student data entered through the GUI; grade records and attendance operations; sort/search query.

Outputs: Alphabetically and ID/GPA-organized tabular lists, student-per-student breakout reports (subjects, grade history, attendance rates), stored data.json file.

Scope: Stand-alone desktop application; no network. Validation will be 1 Year 1..4 and Gender F or M.

5. System Design and Algorithm Description

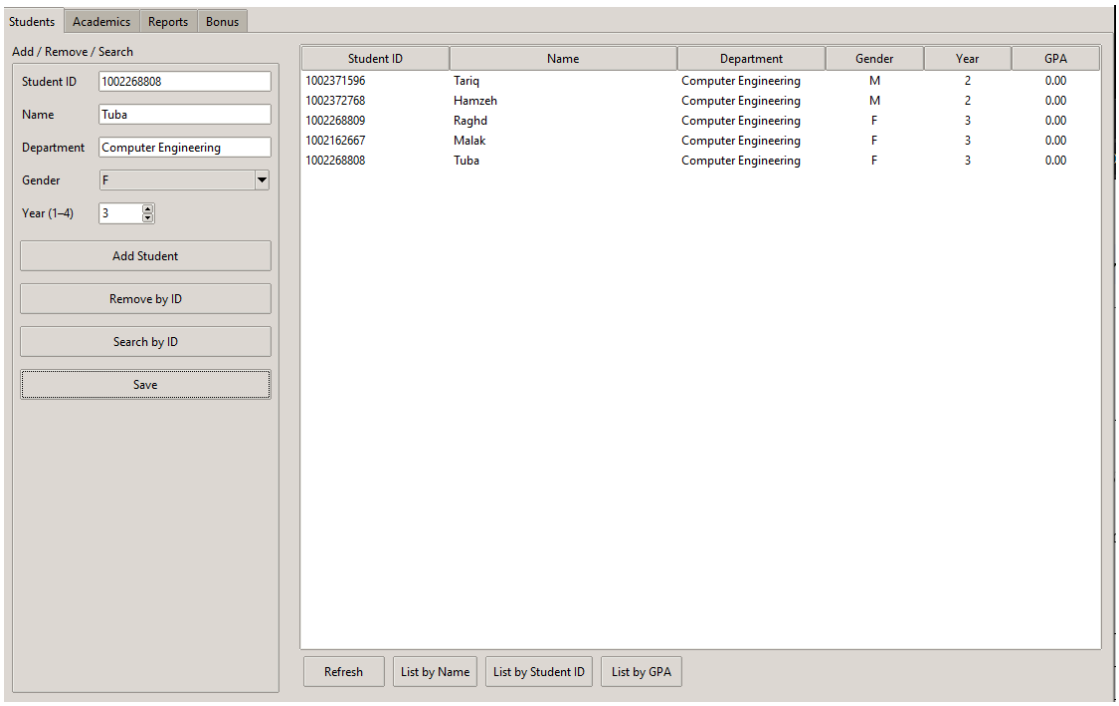
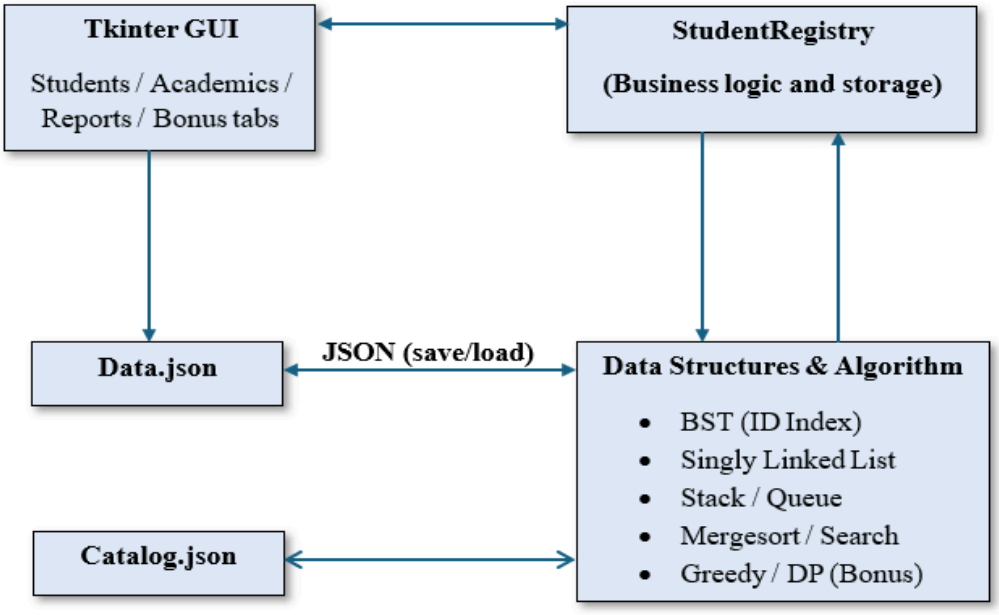


Figure 1: Main Interface for student registration

5.1. Architecture (Block Diagram)



The GUI calls StudentRegistry methods. The registry holds students in memory, indexes them in a Binary Search Tree (BST) for ID lookups, and persists everything to data.json. Subject time slots for the timetable optimizer are stored in catalog.json.

5.2. Data Structures Used and Why

Binary Search Tree (BST): fast searching student pairings. In order traversal automatically orders students in their ID sequence without need of a sor

Singly Linked List: each student subject is a linked list; append/drop are simply address modifications and it is intended to illustrate node-based structures as requested by the assignments.

LIFO Stack: allows returning to the previous grade via the Undo Last Grade operation.

FIFO Queue: attendance processing in a batch; asynchronous attendance batch processing of first in first out.

Lists/Dicts (built-in): to hold student list and grade maps, and subject-slot catalog.

5.3. Core Algorithms

Mergesort: list students by Name and by GPAh/delete by Student ID; in order for ID-sorted view.

Binary search: backend-only by Name over a mergesorted list..

BST operations: insert/searc

Greedy selection: pick class representatives per department using weighted score:
Score = $\alpha \times (GPA/4 \times 100) + \beta \times$ Attendance%.

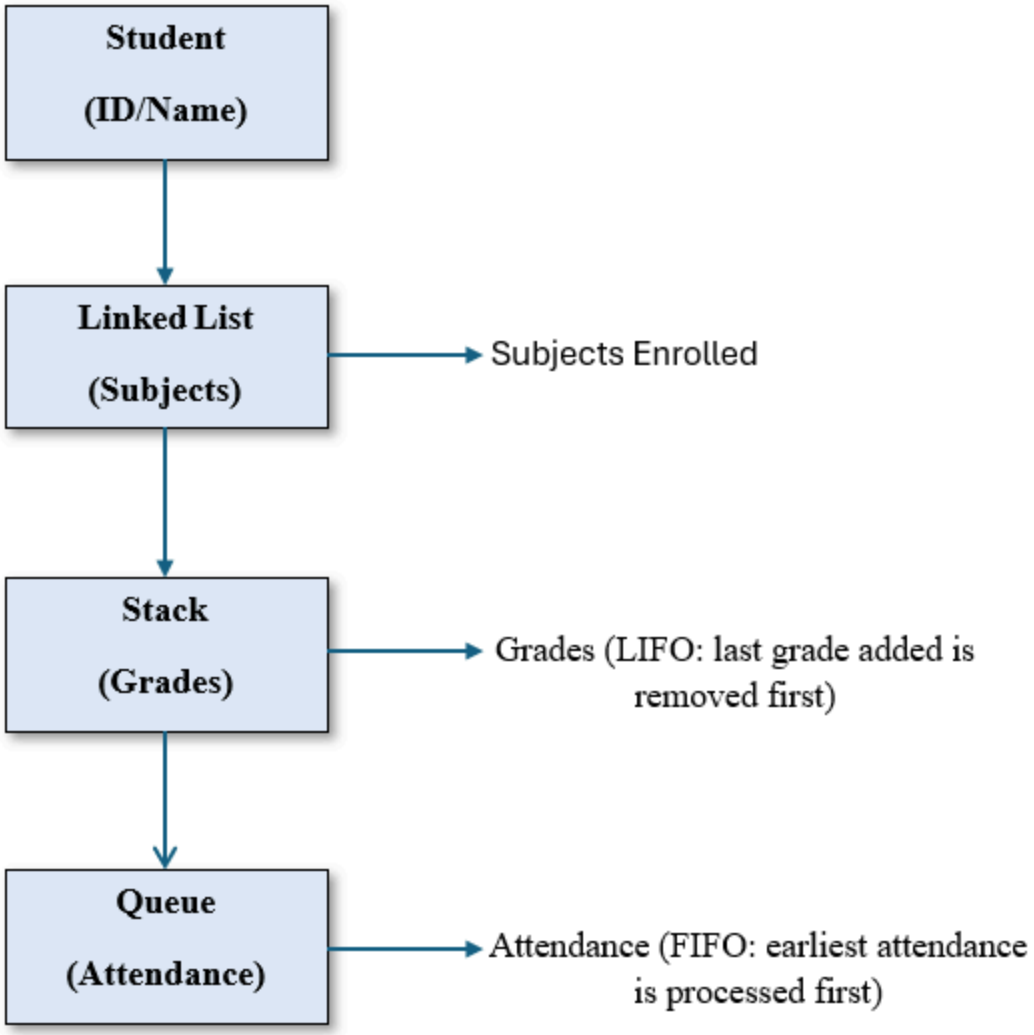
Dynamic Programming – timetable optimizer picks a max-weight set of non-overlapping subjects for a student.

5.4. Justification

BST vs. array search – We insert/search/delete by ID frequently; BST gives good expected performance and a sorted ID listing for free. A balanced tree (AVL/RB) could improve the worst-case but adds complexity beyond scope.

Mergesort vs. quicksort for GUI lists – Mergesort is stable and O(n log n) worst case; stable order is helpful when names tie.

Stack/Queue/Linked List – These mirror real actions: undo (LIFO), attendance stream (FIFO), and a simple, explicit structure for subjects.


Student
(ID/Name)
Linked List
(Subjects)
Subjects Enrolled
Stack
(Grades)
Grades (LIFO: last grade added is removed first)
Queue
(Attendance)
Attendance (FIFO: earliest attendance is processed first)

6. Implementation Details

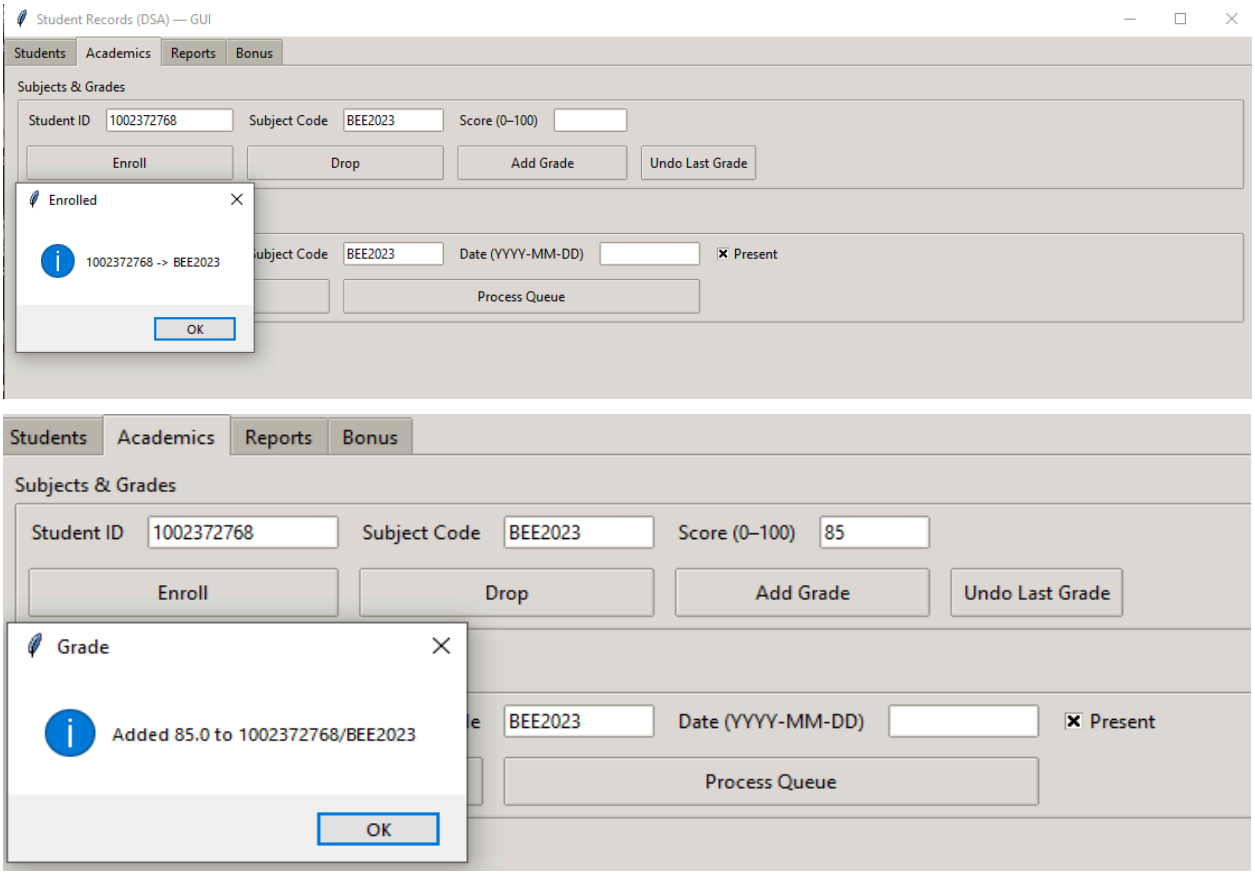


Figure 2: Academics tab after enrolling a subject and adding a grade

6.1. Language & Libraries

Python 3.13 standard library only: dataclasses, typing, collections.deque, json, os, datetime, tkinter/ttk

6.2. Modules

- student_records.py — data models, BST, linked list, stack, queue, mergesort/binary search, greedy reps, DP optimizer, JSON persistence.
- student_records_gui.py — Tkinter GUI with four tabs and event-driven callbacks.

6.3. Functional Walkthrough

1. **Registration**: form validates Student ID, Year 1..4, Gender F,M; appends list and BST index; on Save persists.
2. **Search and Listing**: Search by ID search uses BST; list by Name/ID/GPA uses mergesort or BST bstorder.
3. **Subjects and Grades**: Enroll/Drop maintain the linked list; Add Grade adds to the list and pushes to grade_history; Undo pops.
4. **Attendance**: Enqueue records (ID, date, subject, present); Process uses them, calculating the per-subject and overall rates.
5. **Reports**: displays a readable per-student overview: department/gender/year, GPA, subjects, attendance %, and raw grades.
6. **Bonus task**: Representatives - per department, by score, show top K; alpha/beta are adjustable to emphasize GPA vs. attendance.
7. **Bonus task**: Timetable - users specify Start/End (HH:MM) and Weight per subject in a catalog, and the optimizer chooses a max-weight non-overlapping set of subjects to assign to the selected student.

7. Testing and Evaluation

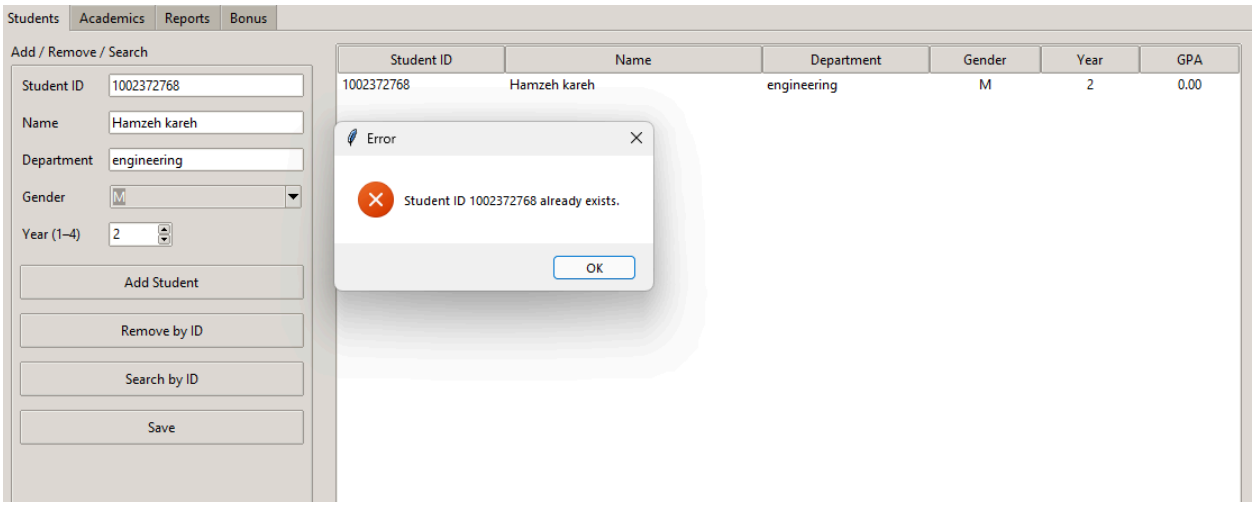


Figure 3: Error after adding the same ID

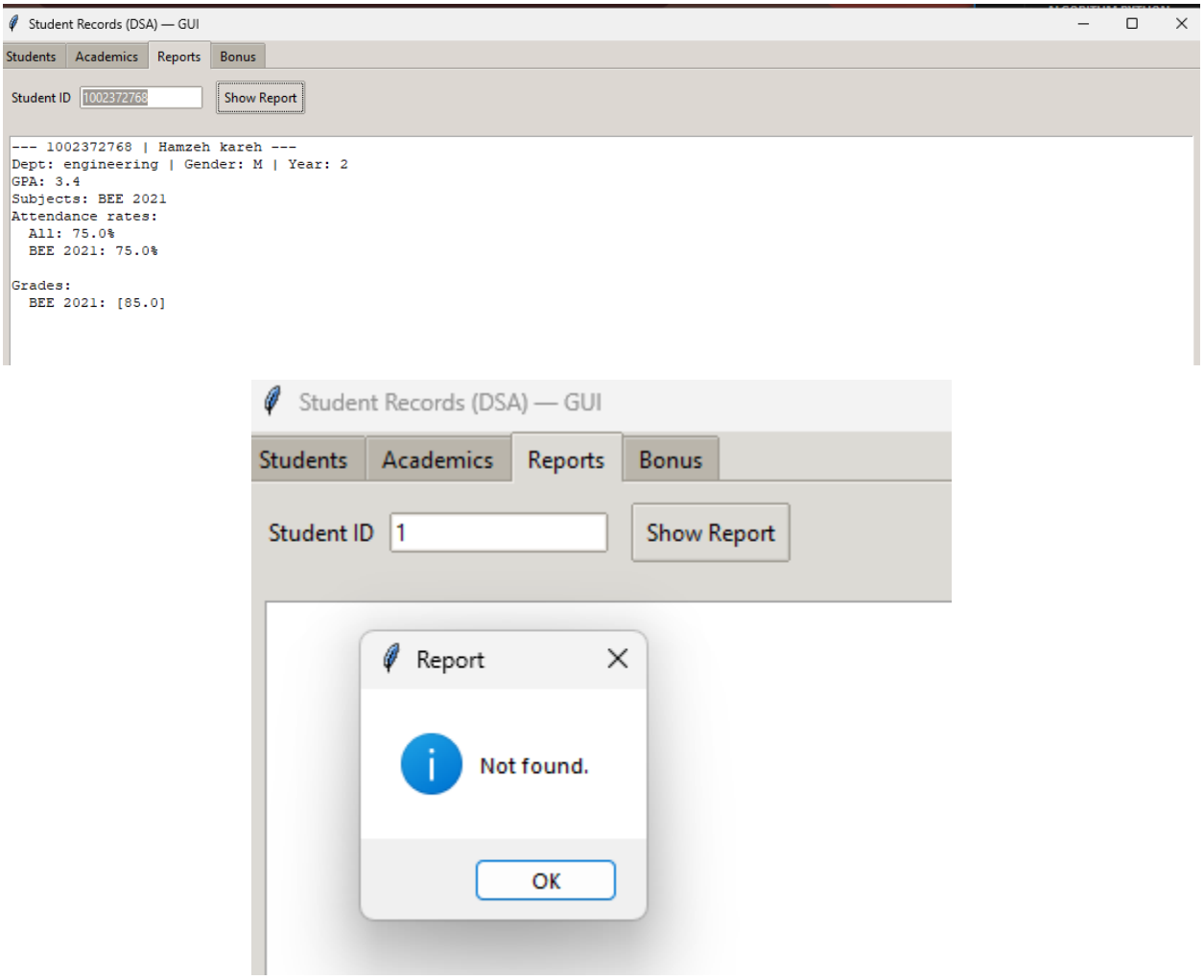


Figure 4: Search by an existing ID vs. non-existing ID.

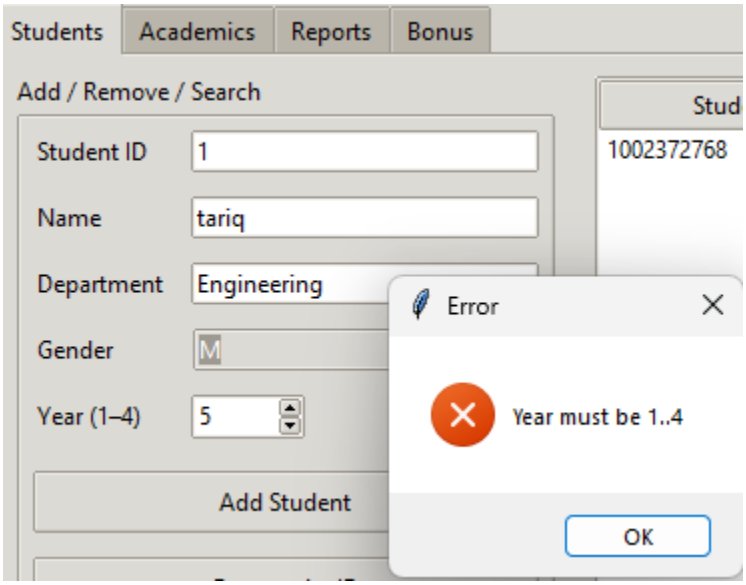
7.1. Test Cases

Registration & Validation:

- Add valid student → appears in table; GPA = 0.00.

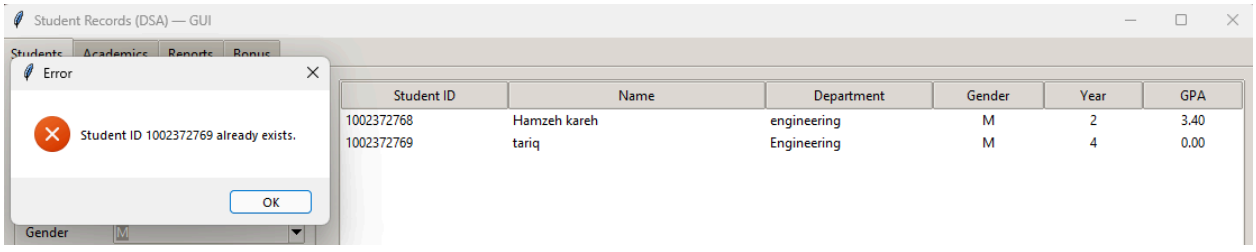
Student ID	Name	Department	Gender	Year	GPA
1002372768	Hamzeh kareh	engineering	M	2	3.40
1002372769	tariq	Engineering	M	4	0.00

- Year outside 1 to 4 → blocked with message.



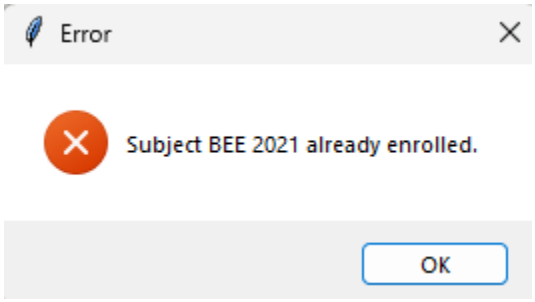
- Gender other than F/M → impossible via GUI (combobox restriction).

- Duplicate ID → error; record not added.

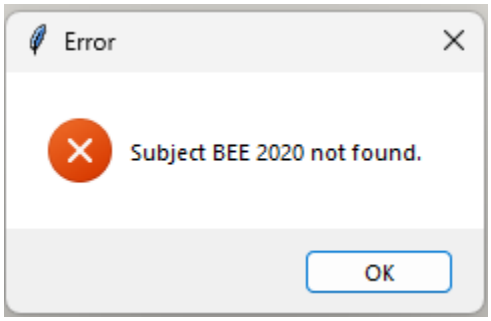


Academics:

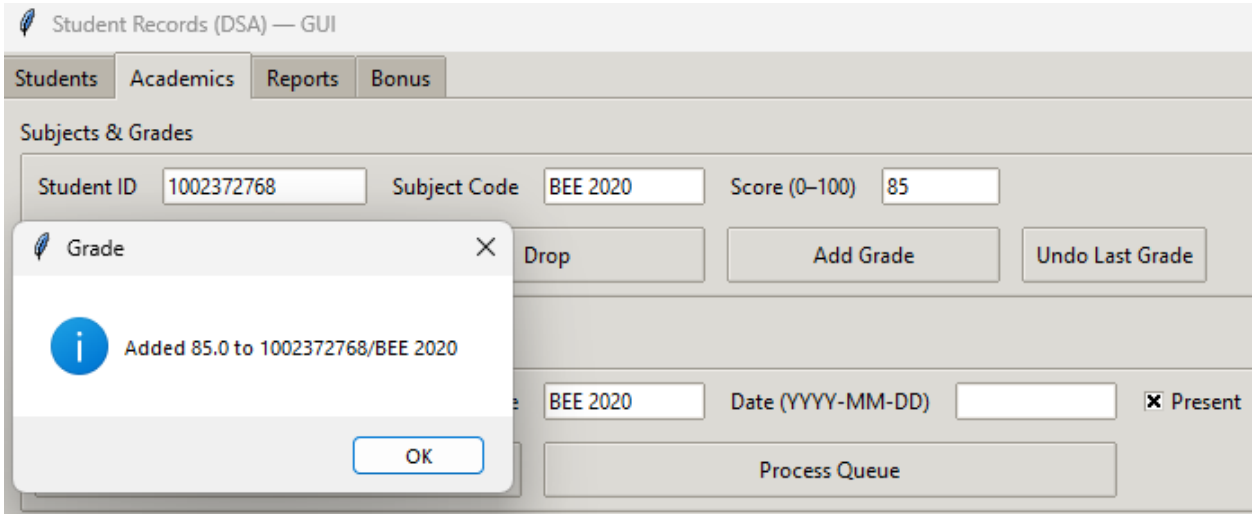
- Enroll subject twice → error.



- Drop non-enrolled subject → error.

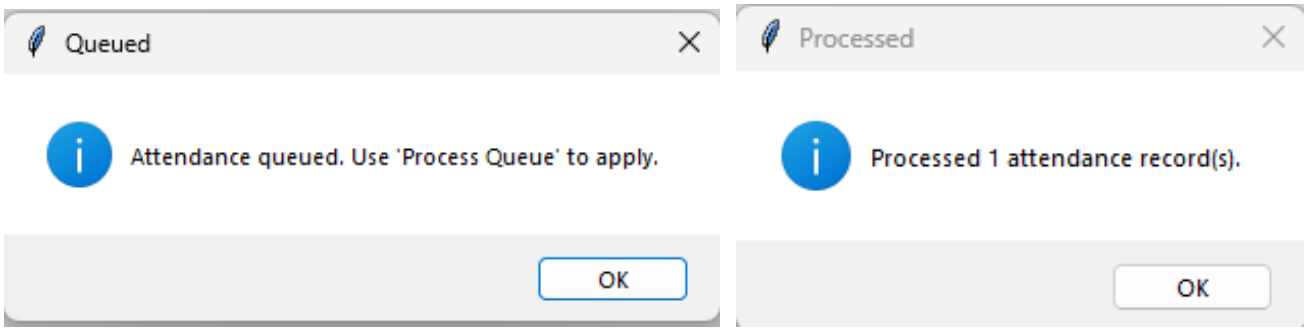


- Add Grade within 0–100 → accepted; Undo removes last one.



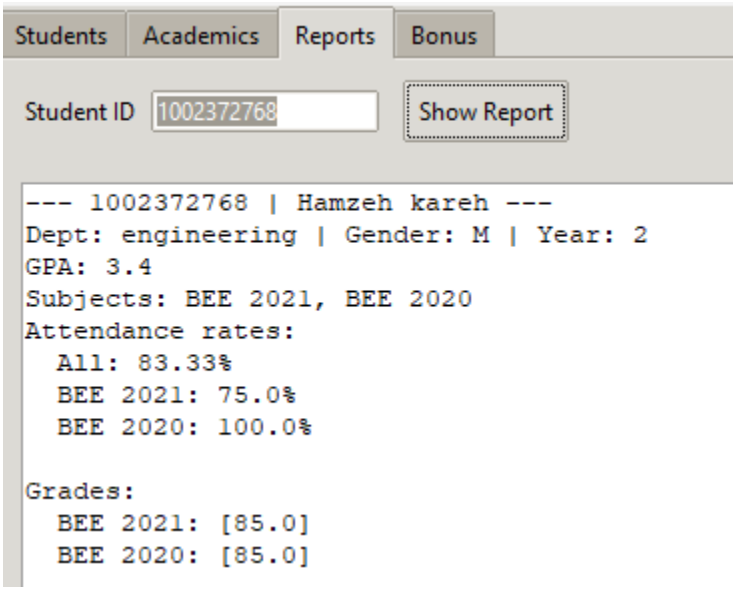
Attendance:

- Queue multiple records → Process applies all; per-subject and overall attendance % update.



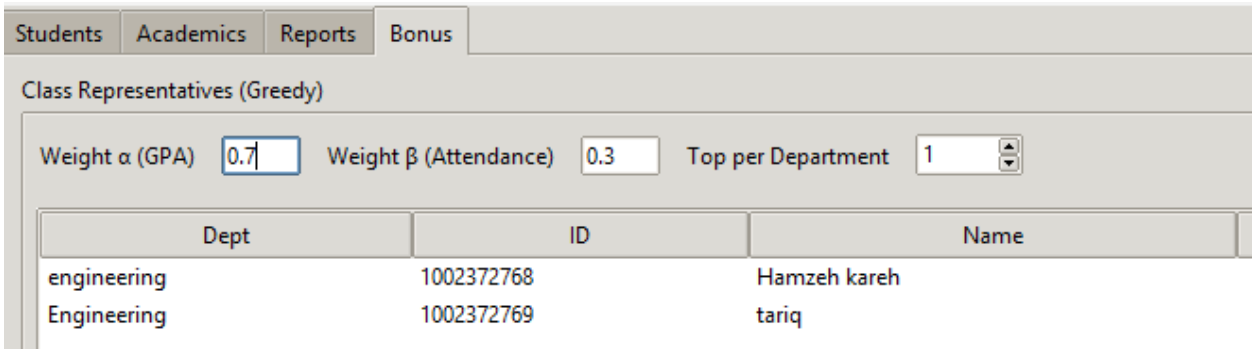
Search & Lists:

- Search by ID → correct dialog.
- List by Name/ID/GPA → order verified (merge stability for tied names).

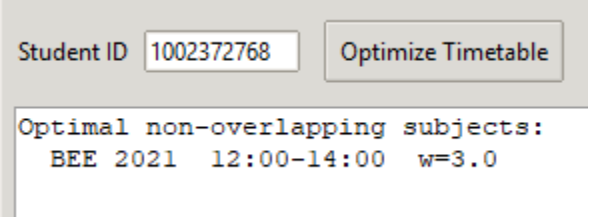


Bonus tacks:

- Representatives: change α/β and confirm ordering changes accordingly.



- Timetable: create overlapping slots and verify that the chosen set is conflict-free and maximizes weight



8. Result and Analysis

8.1. Complexity Performance

- O(n log n) time, O(n) extra space; stable - does not change the order of equal-key items.
- **Theoretical**: O(log n) insert/search/delete; inorder listing O(n). O(n) worst case, skewed; acceptable at assignment scale.
- **Stack/Queue**: constant push/pop, enqueue/dequeue.
- **Optimizer**: O(n 2) on simple p(j) calculation; could be O(n log n) by computing p(j) using a binary search after sorting by end times.

8.2. Accuracy & Efficiency

- **Representatives**: Weighted scoring balances academic performance and reliability. Sensitivity tests (vary α/β) show expected rank shifts.
- **Timetable**: Always returns a non-overlapping set with maximum total weight under the model.

8.3. Challenges Faced

- Mapping GUI actions to model operations cleanly (avoid state drift).
- Designing persistence that never crashes even when files are missing or corrupted.
- Explaining algorithm choices clearly for the report while keeping code readable.

8.4. Limitations & Future Work

- BST is not self-balancing; AVL/RB would remove worst-case.

- Timetable assumes single-day, single room per subject; multi-day scheduling is future work.
- No authentication/roles.

9. Conclusion

This project shows how classic data structures (linked list, stack, queue, BST) and algorithms (mergesort, binary search, greedy and dynamic programming) can be used in combination in a feasible Student Information System that has a nice Tkinter Graphical User Interface. The system satisfies all core requirements: registration, attendance, grades search, report generation, and has both bonus features: the greedy selection of class representatives and DP-style timetable optimization. The outcome is flexible, friendly and heightens. The balancing, richer analytics and multi-day scheduling will be concentrated in the future work.

10. References

1. *Python Software Foundation, "The Python Standard Library Documentation," Python.org, 2024.*
2. *Tkinter 8.6 Reference, "Tkinter — Python interface to Tcl/Tk," Python.org, 2024.*
3. *M. Roseman, TkDocs: Modern Tkinter for Busy Python Developers, 2024.*
4. *U. of Illinois (J. Erickson), "Greedy Algorithms and Interval Scheduling," course notes, 2019.*

11. Appendix

Source Files:

Student_records.py – full backend (DSA, algorithms, registry, JSON, bonus features).
Student_records_gui.py – full Tkinter GUI (Students/Academics/Reports/Bonus tabs).

Files made at Run time

data.json — for saving student records

catalog.json — for timetable optimizer

Student_records.py:

```python
from __future__ import annotations
from dataclasses import dataclass, field, asdict
from typing import Any, Callable, Dict, Iterable, Iterator, List, Optional, Tuple, TypeVar, Generic, List as PyList
import json, os, datetime
from collections import deque

#Subject adding system (Linked List)
class _LLNode:
    __slots__ = ("value","next")
    def __init__(self, value: Any, nxt: Optional["__class__"]=None):
        self.value = value
        self.next = nxt

class SinglyLinkedList:
    def __init__(self, values: Optional[Iterable[Any]]=None):
        self._head: Optional[_LLNode] = None
        self._size = 0
        if values:
            for v in values:
                self.append(v)
    def __len__(self) -> int: return self._size
    def __iter__(self) -> Iterator[Any]:
        cur = self._head
        while cur is not None:
            yield cur.value
            cur = cur.next
    def __contains__(self, item: Any) -> bool:
        for v in self:
            if v == item: return True
        return False
    def to_list(self) -> List[Any]: return list(iter(self))
    def append(self, value: Any) -> None:
        n = _LLNode(value)
        if not self._head: self._head = n
        else:
            cur = self._head
            while cur.next is not None: cur = cur.next
            cur.next = n
        self._size += 1
    def remove(self, value: Any) -> bool:
        prev = None; cur = self._head
        while cur is not None:
            if cur.value == value:
                if prev is None: self._head = cur.next
                else: prev.next = cur.next
                self._size -= 1; return True
            prev, cur = cur, cur.next
        return False
```

```python
# Grading system (Stack)
T = TypeVar("T")
class Stack(Generic[T]):
    def __init__(self): self._data: PyList[T] = []
    def push(self, item: T) -> None: self._data.append(item)
    def pop(self) -> T:
        if not self._data: raise IndexError("pop from empty stack")
        return self._data.pop()
    def is_empty(self) -> bool: return len(self._data) == 0
    def peek(self) -> T:
        if not self._data: raise IndexError("peek from empty stack")
        return self._data[-1]
    def __iter__(self): return iter(self._data)
    def __len__(self): return len(self._data)

# Attandance (Queue)
class Queue(Generic[T]):
    def __init__(self): self._dq = deque()
    def enqueue(self, item: T) -> None: self._dq.append(item)
    def dequeue(self) -> T:
        if not self._dq: raise IndexError("dequeue from empty queue")
        return self._dq.popleft()
    def peek(self) -> T:
        if not self._dq: raise IndexError("peek from empty queue")
        return self._dq[0]
    def is_empty(self) -> bool: return len(self._dq) == 0
    def __len__(self): return len(self._dq)
```

```python
# Report system (BST)
@dataclass
class _BSTNode:
    key: Any
    value: Any
    left: Optional["_BSTNode"]=None
    right: Optional["_BSTNode"]=None

class BinarySearchTree:
    def __init__(self): self._root: Optional[_BSTNode]=None; self._size=0
    def __len__(self): return self._size
    def insert(self, key: Any, value: Any) -> None:
        def _ins(n: Optional[_BSTNode], k: Any, v: Any) -> _BSTNode:
            if n is None: return _BSTNode(k,v)
            if k < n.key: n.left = _ins(n.left,k,v)
            elif k > n.key: n.right = _ins(n.right,k,v)
            else: n.value = v
            return n
        self._root = _ins(self._root, key, value)
        self._size = self._compute_size(self._root)
    def search(self, key: Any) -> Optional[Any]:
        n = self._root
        while n is not None:
            if key < n.key: n = n.left
            elif key > n.key: n = n.right
            else: return n.value
        return None
    def delete(self, key: Any) -> None:
```

```python
    def delete(self, key: Any) -> None:
        def _min(n: _BSTNode) -> _BSTNode:
            while n.left is not None: n = n.left
            return n
        def _del(n: Optional[_BSTNode], k: Any) -> Optional[_BSTNode]:
            if n is None: return None
            if k < n.key: n.left = _del(n.left,k)
            elif k > n.key: n.right = _del(n.right,k)
            else:
                if n.left is None: return n.right
                if n.right is None: return n.left
                t = _min(n.right)
                n.key, n.value = t.key, t.value
                n.right = _del(n.right, t.key)
            return n
        self._root = _del(self._root, key)
        self._size = self._compute_size(self._root)
    def inorder(self) -> Iterator[Tuple[Any,Any]]:
        def _in(n):
            if n:
                yield from _in(n.left); yield (n.key,n.value); yield from _in(n.right)
        yield from _in(self._root)
    def _compute_size(self, n: Optional[_BSTNode]) -> int:
        return 0 if n is None else 1 + self._compute_size(n.left) + self._compute_size(n.right)

```

```python
  # Sorting & Searching
def mergesort(arr: List[Any], key: Optional[Callable[[Any], Any]]=None) -> List[Any]:
    if key is None: key = lambda x: x
    if len(arr) <= 1: return arr[:]
    mid = len(arr)//2
    left = mergesort(arr[:mid], key=key)
    right = mergesort(arr[mid:], key=key)
    return _merge(left, right, key)

def _merge(a: List[Any], b: List[Any], key: Callable[[Any], Any]) -> List[Any]:
    i=j=0; out: List[Any]=[]
    while i<len(a) and j<len(b):
        if key(a[i]) <= key(b[j]): out.append(a[i]); i+=1
        else: out.append(b[j]); j+=1
    out.extend(a[i:]); out.extend(b[j:]); return out

def quicksort(arr: List[Any], key: Optional[Callable[[Any], Any]]=None) -> List[Any]:
    if key is None: key = lambda x: x
    if len(arr)<=1: return arr[:]
    pivot = arr[len(arr)//2]; pv = key(pivot)
    less = [x for x in arr if key(x) < pv]
    equal = [x for x in arr if key(x) == pv]
    greater = [x for x in arr if key(x) > pv]
    return quicksort(less, key=key) + equal + quicksort(greater, key=key)

def binary_search(sorted_arr: List[Any], target: Any, key: Optional[Callable[[Any], Any]]=None) -> int:
    if key is None: key = lambda x: x
    lo, hi = 0, len(sorted_arr)-1
    while lo<=hi:
        mid = (lo+hi)//2; km = key(sorted_arr[mid])
        if km == target:
            while mid>0 and key(sorted_arr[mid-1])==target: mid -= 1
            return mid
        elif km < target: lo = mid+1
        else: hi = mid-1
    return -1

```

```python
# Models
@dataclass
class AttendanceRecord:
    date: str
    subject: str
    present: bool

@dataclass
class Student:
    student_id: str
    name: str
    department: str
    gender: str
    year: int
    subjects: SinglyLinkedList = field(default_factory=SinglyLinkedList)
    grades: Dict[str, List[float]] = field(default_factory=dict)
    grade_history: Stack[Tuple[str, float]] = field(default_factory=Stack)
    attendance_log: List[AttendanceRecord] = field(default_factory=list)

    def __post_init__(self) -> None:
        if self.year < 1 or self.year > 4:
            raise ValueError("Year must be 1..4")
    def enroll_subject(self, code: str) -> None:
        code = code.strip().upper()
        if code in self.subjects: raise ValueError(f"Subject {code} already enrolled.")
        self.subjects.append(code)
    def drop_subject(self, code: str) -> None:
        code = code.strip().upper()
        if not self.subjects.remove(code): raise ValueError(f"Subject {code} not found.")
    def add_grade(self, subject: str, score: float) -> None:
        subject = subject.strip().upper()
        if subject not in self.subjects: raise ValueError(f"Not enrolled in {subject}.")
        if not (0 <= score <= 100): raise ValueError("Score must be 0..100")
        self.grades.setdefault(subject, []).append(score)
        self.grade_history.push((subject, score))
    def undo_last_grade(self) -> None:
        if self.grade_history.is_empty(): raise ValueError("No grades to undo.")
        subj, score = self.grade_history.pop()
        lst = self.grades.get(subj, [])
        if lst and lst[-1] == score: lst.pop()
    def record_attendance(self, date: str, subject: str, present: bool) -> None:
        subject = subject.strip().upper()
        if subject not in self.subjects: raise ValueError(f"Not enrolled in {subject}.")
        self.attendance_log.append(AttendanceRecord(date=date, subject=subject, present=present))
    def gpa(self) -> float:
        if not self.grades: return 0.0
        avgs = [sum(v)/len(v) for v in self.grades.values() if v]
        if not avgs: return 0.0
```

```
        avgs = [sum(v)/len(v) for v in self.grades.values() if v]
        if not avgs: return 0.0
        pct = sum(avgs)/len(avgs)
        return round((pct/100)*4, 2)
    def attendance_rate(self, subject: Optional[str]=None) -> float:
        logs = self.attendance_log if subject is None else [r for r in self.attendance_log if r.subject == subject.upper()]
        if not logs: return 0.0
        present = sum(1 for r in logs if r.present)
        return round(100.0 * present / len(logs), 2)
    def to_dict(self) -> dict:
        d = asdict(self)
        d["subjects"] = self.subjects.to_list()
        d["grade_history"] = list(self.grade_history)
        d["attendance_log"] = [asdict(r) for r in self.attendance_log]
        return d
    @staticmethod
    def from_dict(d: dict) -> "Student":
        s = Student(d["student_id"], d["name"], d["department"], d["gender"], int(d["year"]))
        for sub in d.get("subjects", []): s.subjects.append(sub)
        s.grades = {k: list(v) for k,v in d.get("grades", {}).items()}
        for subj,score in d.get("grade_history", []): s.grade_history.push((subj, float(score)))
        for rec in d.get("attendance_log", []): s.attendance_log.append(AttendanceRecord(**rec))
        return s
```

```
# Registry
DATA_FILE = "data.json"
CATALOG_FILE = "catalog.json"

class StudentRegistry:
    def __init__(self) -> None:
        self._students: List[Student] = []
        self._index = BinarySearchTree()
        self._attendance_q: Queue[Tuple[str, str, str, bool]] = Queue()
        self._subject_catalog: Dict[str, Dict[str, float]] = {}
    def add_student(self, s: Student) -> None:
        if self.get_by_id(s.student_id) is not None:
            raise ValueError(f"Student ID {s.student_id} already exists.")
        self._students.append(s)
        self._index.insert(s.student_id, s)
    def get_by_id(self, sid: str) -> Optional[Student]:
        return self._index.search(sid)
    def remove_student(self, sid: str) -> None:
        if self.get_by_id(sid) is None: raise ValueError("Not found.")
        self._students = [x for x in self._students if x.student_id != sid]
        self._index.delete(sid)
    def enqueue_attendance(self, sid: str, date: str, subject: str, present: bool) -> None:
        self._attendance_q.enqueue((sid, date, subject, present))
    def process_attendance(self) -> int:
        cnt=0
        while not self._attendance_q.is_empty():
            sid,date,subj,present = self._attendance_q.dequeue()
            s = self.get_by_id(sid)
            if s is None:
                print(f"[WARN] Unknown student {sid}"); continue
            try:
                if not date: date = str(datetime.date.today())
                s.record_attendance(date, subj, present)
                cnt += 1
            except Exception as e:
                print(f"[WARN] attendance failed for {sid}: {e}")
        return cnt
```

```
274     def list_students(self) -> List[Student]: return list(self._students)
275     def sorted_by_name(self) -> List[Student]:
276         key = lambda s: s.name.lower()
277         return mergesort(self._students, key=key)
278     def sorted_by_id(self) -> List[Student]:
279         return [v for _, v in self._index.inorder()]
280     def sorted_by_gpa(self, descending: bool=True) -> List[Student]:
281         arr = self._students[:]; key = lambda s: s.gpa()
282         out = mergesort(arr, key=key)
283         return list(reversed(out)) if descending else out
284     def binary_search_by_name(self, name: str) -> List[Student]:
285         arr = mergesort(self._students, key=lambda s: s.name.lower())
286         idx = binary_search(arr, name.lower(), key=lambda s: s.name.lower())
287         if idx == -1: return []
288         out=[]; i=idx
289         while i<len(arr) and arr[i].name.lower()==name.lower(): out.append(arr[i]); i+=1
290         return out
291     def save(self, path: str=DATA_FILE) -> None:
292         with open(path,"w",encoding="utf-8") as f:
293             json.dump([s.to_dict() for s in self._students], f, indent=2)
294         self.save_catalog()
295     def load(self, path: str=DATA_FILE) -> None:
296         if not os.path.exists(path):
297             with open(path, "w", encoding="utf-8") as f: json.dump([], f)
298         try:
299             with open(path, "r", encoding="utf-8") as f:
300                 raw = f.read()
301             if not raw.strip():
302                 data = []
303             else:
304                 data = json.loads(raw)
305         except Exception:
306             try:
307                 os.replace(path, path + ".corrupt")
308             except Exception:
309                 pass
310             data = []
311         self._students.clear(); self._index = BinarySearchTree()
312         for d in data:
313             try:
314                 self.add_student(Student.from_dict(d))
315             except Exception:
316                 continue
317         self.load_catalog()
318     def save_catalog(self, path: str=CATALOG_FILE) -> None:
319         try:
320             with open(path, "w", encoding="utf-8") as f:
```

```
322         except Exception as e:
323             print("[WARN] could not save catalog:", e)
324     def load_catalog(self, path: str=CATALOG_FILE) -> None:
325         if not os.path.exists(path):
326             with open(path, "w", encoding="utf-8") as f: json.dump({}, f)
327         try:
328             with open(path, "r", encoding="utf-8") as f:
329                 raw = f.read().strip()
330                 self._subject_catalog = json.loads(raw) if raw else {}
331         except Exception:
332             try:
333                 os.replace(path, path + ".corrupt")
334             except Exception:
335                 pass
336             self._subject_catalog = {}
337
338     # Class Representatives (Greedy)
339     def choose_class_representatives(self, top_per_dept: int=1, alpha: float=0.7, beta: float=0.3) -> Dict[str, List[Tuple[Student, float]]]:
340         buckets: Dict[str, List[Tuple[Student, float]]] = {}
341         for s in self._students:
342             score = alpha * (s.gpa()/4.0*100.0) + beta * s.attendance_rate()
343             buckets.setdefault(s.department or "-", []).append((s, round(score, 2)))
344         chosen: Dict[str, List[Tuple[Student, float]]] = {}
345         for dept, lst in buckets.items():
346             lst.sort(key=lambda t: t[1], reverse=True)
347             chosen[dept] = lst[:max(1, int(top_per_dept))]
348         return chosen
349
350     # Timetable Optimizer
351     def set_subject_slot(self, code: str, start_min: int, end_min: int, weight: float=1.0) -> None:
352         code = code.strip().upper()
353         if end_min <= start_min: raise ValueError("End must be after start")
354         self._subject_catalog[code] = {"start": int(start_min), "end": int(end_min), "weight": float(weight)}
355     def get_subject_slot(self, code: str) -> Optional[Dict[str, float]]:
356         return self._subject_catalog.get(code.strip().upper())
357     def list_subject_slots(self) -> Dict[str, Dict[str, float]]:
358         return dict(self._subject_catalog)
359     def optimize_timetable_for(self, student_id: str) -> List[str]:
360
361         s = self.get_by_id(student_id)
362         if s is None: raise ValueError("Student not found")
363         items: List[Tuple[int,int,float,str]] = []
364         for code in s.subjects:
365             slot = self._subject_catalog.get(code)
366             if slot:
367                 items.append((int(slot["start"]), int(slot["end"]), float(slot.get("weight",1.0)), code))
```

```
368         if not items: return []
369         items.sort(key=lambda x: x[1])
370         n = len(items)
371 
372         p = [0]*n
373         for j in range(n):
374             p[j] = 0
375             sj_start = items[j][0]
376             for i in range(j-1, -1, -1):
377                 if items[i][1] <= sj_start:
378                     p[j] = i+1
379                     break
380 
381         dp = [0.0]*(n+1)
382         keep = [False]*(n+1)
383         for j in range(1, n+1):
384             incl = items[j-1][2] + dp[p[j-1]]
385             excl = dp[j-1]
386             if incl > excl:
387                 dp[j] = incl; keep[j] = True
388             else:
389                 dp[j] = excl; keep[j] = False
390 
391         chosen_codes: List[str] = []
392         j = n
393         while j > 0:
394             if keep[j]:
395                 chosen_codes.append(items[j-1][3])
396                 j = p[j-1]
397             else:
398                 j -= 1
399         chosen_codes.reverse()
400         return chosen_codes
401 
```

Student_records_gui.py:

```python
import tkinter as tk
from tkinter import ttk, messagebox
import datetime
from student_records import StudentRegistry, Student
#timetable
def time_to_minutes(hhmm: str) -> int:
    hhmm = hhmm.strip()
    if not hhmm:
        raise ValueError("Empty time")
    if ":" not in hhmm:
        raise ValueError("Time must be HH:MM")
    h, m = hhmm.split(":", 1)
    h = int(h); m = int(m)
    if not (0 <= h < 24 and 0 <= m < 60):
        raise ValueError("Time out of range")
    return h*60 + m

def minutes_to_time(mins: int) -> str:
    h = mins // 60; m = mins % 60
    return f"{h:02d}:{m:02d}"

class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Student Records (DSA) — GUI")
        self.geometry("1120x700")
        self.minsize(1000, 640)
        self.reg = StudentRegistry()
        self.reg.load()

        style = ttk.Style()
        try: style.theme_use("clam")
        except Exception: pass

        nb = ttk.Notebook(self)
        self.tab_students = ttk.Frame(nb)
        self.tab_academics = ttk.Frame(nb)
        self.tab_reports = ttk.Frame(nb)
        self.tab_bonus = ttk.Frame(nb)
        nb.add(self.tab_students, text="Students")
        nb.add(self.tab_academics, text="Academics")
        nb.add(self.tab_reports, text="Reports")
        nb.add(self.tab_bonus, text="Bonus")
        nb.pack(fill="both", expand=True)
```

```
45
46          self._build_students_tab()
47          self._build_academics_tab()
48          self._build_reports_tab()
49          self._build_bonus_tab()
50
51          self._refresh_student_table()
52          self._refresh_catalog_table()
53
```

```
#Students Tab
def _build_students_tab(self):
    f = self.tab_students
    form = ttk.LabelFrame(f, text="Add / Remove / Search")
    form.pack(side=tk.LEFT, fill=tk.Y, padx=10, pady=10)

    self.sid_var = tk.StringVar()
    self.name_var = tk.StringVar()
    self.dept_var = tk.StringVar()
    self.gender_var = tk.StringVar(value="F")
    self.year_var = tk.IntVar(value=1)

    row=0
    ttk.Label(form, text="Student ID").grid(row=row, column=0, sticky="w", padx=6, pady=6)
    ttk.Entry(form, textvariable=self.sid_var, width=28).grid(row=row, column=1, padx=6, pady=6); row+=1
    ttk.Label(form, text="Name").grid(row=row, column=0, sticky="w", padx=6, pady=6)
    ttk.Entry(form, textvariable=self.name_var, width=28).grid(row=row, column=1, padx=6, pady=6); row+=1
    ttk.Label(form, text="Department").grid(row=row, column=0, sticky="w", padx=6, pady=6)
    ttk.Entry(form, textvariable=self.dept_var, width=28).grid(row=row, column=1, padx=6, pady=6); row+=1
    ttk.Label(form, text="Gender").grid(row=row, column=0, sticky="w", padx=6, pady=6)
    ttk.Combobox(form, textvariable=self.gender_var, values=["F","M"], width=25, state="readonly").grid(row=row, column=1, padx=6, pady=6); row+=1
    ttk.Label(form, text="Year (1-4)").grid(row=row, column=0, sticky="w", padx=6, pady=6)
    ttk.Spinbox(form, from_=1, to=4, textvariable=self.year_var, width=5).grid(row=row, column=1, sticky="w", padx=6, pady=6); row+=1

    ttk.Button(form, text="Add Student", command=self._add_student).grid(row=row, column=0, columnspan=2, sticky="ew", padx=6, pady=(10,6)); row+=1
    ttk.Button(form, text="Remove by ID", command=self._remove_student).grid(row=row, column=0, columnspan=2, sticky="ew", padx=6, pady=6); row+=1
    ttk.Button(form, text="Search by ID", command=self._search_by_id).grid(row=row, column=0, columnspan=2, sticky="ew", padx=6, pady=6); row+=1
    ttk.Button(form, text="Save", command=self._save).grid(row=row, column=0, columnspan=2, sticky="ew", padx=6, pady=(6,12))

    table_frame = ttk.Frame(f); table_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10, pady=10)
    cols = ("id","name","dept","gender","year","gpa")
    self.tree = ttk.Treeview(table_frame, columns=cols, show="headings", height=18)
    for c, txt, w, anchor in [("id","Student ID",120,"w"),("name","Name",200,"w"),("dept","Department",120,"w"),("gender","Gender",70,"center"),("year","Year",60,"center"),("gpa","GPA
        self.tree.heading(c, text=txt); self.tree.column(c, width=w, anchor=anchor)
    self.tree.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

    btns = ttk.Frame(table_frame); btns.pack(side=tk.TOP, fill=tk.X)
    ttk.Button(btns, text="Refresh", command=self._refresh_student_table).pack(side=tk.LEFT, padx=4, pady=6)
    ttk.Button(btns, text="List by Name", command=self._list_by_name).pack(side=tk.LEFT, padx=4)
    ttk.Button(btns, text="List by Student ID", command=self._list_by_id).pack(side=tk.LEFT, padx=4)
    ttk.Button(btns, text="List by GPA", command=self._list_by_gpa).pack(side=tk.LEFT, padx=4)
```

```
    def _add_student(self):
        try:
            sid = self.sid_var.get().strip()
            if not sid: raise ValueError("Student ID is required.")
            s = Student(student_id=sid, name=self.name_var.get().strip() or "Unnamed", department=self.dept_var.get().strip() or "-", gender=self.gender_var.get().strip() or "F", year=int(self.year_var.get()))
            self.reg.add_student(s); self._refresh_student_table()
            messagebox.showinfo("Added", f"Student {sid} added.")
        except Exception as e:
            messagebox.showerror("Error", str(e))

    def _remove_student(self):
        sid = self.sid_var.get().strip() or self._selected_student_id()
        if not sid: messagebox.showwarning("Select", "Enter or select a Student ID to remove."); return
        try:
            self.reg.remove_student(sid); self._refresh_student_table()
            messagebox.showinfo("Removed", f"Student {sid} removed.")
        except Exception as e: messagebox.showerror("Error", str(e))

    def _search_by_id(self):
        sid = self.sid_var.get().strip()
        if not sid: messagebox.showwarning("Input", "Enter a Student ID to search."); return
        s = self.reg.get_by_id(sid)
        if s is None: messagebox.showinfo("Search", "Not found.")
        else:
            info = f"ID: {s.student_id}\nName: {s.name}\nDept: {s.department}\nGender: {s.gender}\nYear: {s.year}\nGPA: {s.gpa()}"
            messagebox.showinfo("Found", info)

    def _save(self):
        try:
            self.reg.save(); messagebox.showinfo("Saved", "Data + catalog saved.")
        except Exception as e: messagebox.showerror("Error", str(e))

    def _fill_table(self, items):
        for i in self.tree.get_children(): self.tree.delete(i)
        for s in items:
            self.tree.insert("", tk.END, values=(s.student_id, s.name, s.department, s.gender, s.year, f"{s.gpa():.2f}"))
```

```
def _refresh_student_table(self):
    self._fill_table(self.reg.list_students())

def _list_by_name(self):
    self._fill_table(self.reg.sorted_by_name())

def _list_by_id(self):
    self._fill_table(self.reg.sorted_by_id())

def _list_by_gpa(self):
    self._fill_table(self.reg.sorted_by_gpa(descending=True))

def _selected_student_id(self):
    sel = self.tree.selection()
    if not sel: return ""
    vals = self.tree.item(sel[0], "values")
    return vals[0] if vals else ""
```

```python
    #Academics Tab
    def _build_academics_tab(self):
        f = self.tab_academics
        frm = ttk.LabelFrame(f, text="Subjects & Grades"); frm.pack(side=tk.TOP, fill=tk.X, padx=10, pady=10)
        self.sid2_var = tk.StringVar(); self.subj_var = tk.StringVar(); self.score_var = tk.StringVar()
        ttk.Label(frm, text="Student ID").grid(row=0, column=0, sticky="w", padx=6, pady=6)
        ttk.Entry(frm, textvariable=self.sid2_var, width=18).grid(row=0, column=1, padx=6, pady=6)
        ttk.Label(frm, text="Subject Code").grid(row=0, column=2, sticky="w", padx=6, pady=6)
        ttk.Entry(frm, textvariable=self.subj_var, width=14).grid(row=0, column=3, padx=6, pady=6)
        ttk.Label(frm, text="Score (0□100)").grid(row=0, column=4, sticky="w", padx=6, pady=6)
        ttk.Entry(frm, textvariable=self.score_var, width=10).grid(row=0, column=5, padx=6, pady=6)
        ttk.Button(frm, text="Enroll", command=self._enroll_subject).grid(row=1, column=0, columnspan=2, sticky="ew", padx=6, pady=6)
        ttk.Button(frm, text="Drop", command=self._drop_subject).grid(row=1, column=2, columnspan=2, sticky="ew", padx=6, pady=6)
        ttk.Button(frm, text="Add Grade", command=self._add_grade).grid(row=1, column=4, columnspan=2, sticky="ew", padx=6, pady=6)
        ttk.Button(frm, text="Undo Last Grade", command=self._undo_grade).grid(row=1, column=6, columnspan=2, sticky="ew", padx=6, pady=6)

        att = ttk.LabelFrame(f, text="Attendance Queue"); att.pack(side=tk.TOP, fill=tk.X, padx=10, pady=10)
        self.date_var = tk.StringVar(); self.present_var = tk.BooleanVar(value=True)
        ttk.Label(att, text="Student ID").grid(row=0, column=0, sticky="w", padx=6, pady=6)
        ttk.Entry(att, textvariable=self.sid2_var, width=18).grid(row=0, column=1, padx=6, pady=6)
        ttk.Label(att, text="Subject Code").grid(row=0, column=2, sticky="w", padx=6, pady=6)
        ttk.Entry(att, textvariable=self.subj_var, width=14).grid(row=0, column=3, padx=6, pady=6)
        ttk.Label(att, text="Date (YYYY-MM-DD)").grid(row=0, column=4, sticky="w", padx=6, pady=6)
        ttk.Entry(att, textvariable=self.date_var, width=14).grid(row=0, column=5, padx=6, pady=6)
        ttk.Checkbutton(att, text="Present", variable=self.present_var).grid(row=0, column=6, padx=6, pady=6)
        ttk.Button(att, text="Queue Attendance", command=self._queue_attendance).grid(row=1, column=0, columnspan=3, sticky="ew", padx=6, pady=6)
        ttk.Button(att, text="Process Queue", command=self._process_attendance).grid(row=1, column=3, columnspan=3, sticky="ew", padx=6, pady=6)

    def _get_student_or_warn(self, sid: str):
        s = self.reg.get_by_id(sid)
        if s is None:
            messagebox.showwarning("Not Found", f"Student {sid} not found.")
            return None
        return s

    def _enroll_subject(self):
        sid, subj = self.sid2_var.get().strip(), self.subj_var.get().strip()
        if not sid or not subj:
            messagebox.showwarning("Input", "Enter Student ID and Subject Code."); return
        s = self._get_student_or_warn(sid)
        if not s: return
        try:
            s.enroll_subject(subj); messagebox.showinfo("Enrolled", f"{sid} -> {subj}")
        except Exception as e: messagebox.showerror("Error", str(e))
```

```
    def _enroll_subject(self):
        sid, subj = self.sid2_var.get().strip(), self.subj_var.get().strip()
        if not sid or not subj:
            messagebox.showwarning("Input", "Enter Student ID and Subject Code."); return
        s = self._get_student_or_warn(sid)
        if not s: return
        try:
            s.enroll_subject(subj); messagebox.showinfo("Enrolled", f"{sid} -> {subj}")
        except Exception as e: messagebox.showerror("Error", str(e))

    def _drop_subject(self):
        sid, subj = self.sid2_var.get().strip(), self.subj_var.get().strip()
        if not sid or not subj:
            messagebox.showwarning("Input", "Enter Student ID and Subject Code."); return
        s = self._get_student_or_warn(sid)
        if not s: return
        try:
            s.drop_subject(subj); messagebox.showinfo("Dropped", f"{sid} -/-> {subj}")
        except Exception as e: messagebox.showerror("Error", str(e))

    def _add_grade(self):
        sid, subj = self.sid2_var.get().strip(), self.subj_var.get().strip()
        if not sid or not subj:
            messagebox.showwarning("Input", "Enter Student ID and Subject Code."); return
        s = self._get_student_or_warn(sid)
        if not s: return
        try:
            score = float(self.score_var.get())
        except Exception:
            messagebox.showwarning("Score", "Enter a valid number (0..100)"); return
        try:
            s.add_grade(subj, score); messagebox.showinfo("Grade", f"Added {score} to {sid}/{subj}")
        except Exception as e: messagebox.showerror("Error", str(e))

    def _undo_grade(self):
        sid = self.sid2_var.get().strip()
        if not sid:
            messagebox.showwarning("Input", "Enter Student ID."); return
        s = self._get_student_or_warn(sid)
        if not s: return
        try:
            s.undo_last_grade(); messagebox.showinfo("Undo", "Last grade undone.")
        except Exception as e: messagebox.showerror("Error", str(e))

    def _queue_attendance(self):
        sid, subj = self.sid2_var.get().strip(), self.subj_var.get().strip()
        date = self.date_var.get().strip(); present = bool(self.present_var.get())
```

```
    def _queue_attendance(self):
        sid, subj = self.sid2_var.get().strip(), self.subj_var.get().strip()
        date = self.date_var.get().strip(); present = bool(self.present_var.get())
        if not sid or not subj:
            messagebox.showwarning("Input", "Enter Student ID and Subject Code."); return
        self.reg.enqueue_attendance(sid, date or str(datetime.date.today()), subj, present)
        messagebox.showinfo("Queued", "Attendance queued. Use 'Process Queue' to apply.")

    def _process_attendance(self):
        try:
            cnt = self.reg.process_attendance(); messagebox.showinfo("Processed", f"Processed {cnt} attendance record(s).")
        except Exception as e: messagebox.showerror("Error", str(e))

```

```python
    #Reports Tab
    def _build_reports_tab(self):
        f = self.tab_reports
        top = ttk.Frame(f); top.pack(side=tk.TOP, fill=tk.X, padx=10, pady=10)
        self.report_sid = tk.StringVar()
        ttk.Label(top, text="Student ID").pack(side=tk.LEFT)
        ttk.Entry(top, textvariable=self.report_sid, width=18).pack(side=tk.LEFT, padx=6)
        ttk.Button(top, text="Show Report", command=self._show_report).pack(side=tk.LEFT, padx=6)
        self.report_text = tk.Text(f, height=20)
        self.report_text.pack(side=tk.TOP, fill=tk.BOTH, expand=True, padx=10, pady=10)

    def _show_report(self):
        sid = self.report_sid.get().strip()
        if not sid: messagebox.showwarning("Input", "Enter Student ID."); return
        s = self.reg.get_by_id(sid)
        if s is None: messagebox.showinfo("Report", "Not found."); return
        lines = [
            f"--- {s.student_id} | {s.name} ---",
            f"Dept: {s.department} | Gender: {s.gender} | Year: {s.year}",
            f"GPA: {s.gpa()}",
            f"Subjects: {', '.join(s.subjects.to_list()) or '-'}",
            "Attendance rates:",
            f"  All: {s.attendance_rate()}%",
        ]
        for sub in s.subjects:
            lines.append(f"  {sub}: {s.attendance_rate(sub)}%")
        lines.append("\nGrades:")
        for sub, scores in s.grades.items():
            lines.append(f"  {sub}: {scores}")
        self.report_text.delete("1.0", tk.END)
        self.report_text.insert(tk.END, "\n".join(lines))
```

```python
    #Bonus chalanges tab
    def _build_bonus_tab(self):
        f = self.tab_bonus

        reps = ttk.LabelFrame(f, text="Class Representatives (Greedy)")
        reps.pack(side=tk.TOP, fill=tk.X, padx=10, pady=10)

        self.alpha_var = tk.DoubleVar(value=0.7)
        self.beta_var = tk.DoubleVar(value=0.3)
        self.topk_var = tk.IntVar(value=1)

        ttk.Label(reps, text="Weight α (GPA)").grid(row=0, column=0, padx=6, pady=6, sticky="w")
        ttk.Entry(reps, textvariable=self.alpha_var, width=6).grid(row=0, column=1, padx=6, pady=6)
        ttk.Label(reps, text="Weight β (Attendance)").grid(row=0, column=2, padx=6, pady=6, sticky="w")
        ttk.Entry(reps, textvariable=self.beta_var, width=6).grid(row=0, column=3, padx=6, pady=6)
        ttk.Label(reps, text="Top per Department").grid(row=0, column=4, padx=6, pady=6, sticky="w")
        ttk.Spinbox(reps, from_=1, to=5, textvariable=self.topk_var, width=5).grid(row=0, column=5, padx=6, pady=6)
        ttk.Button(reps, text="Pick Representatives", command=self._pick_reps).grid(row=0, column=6, padx=10, pady=6)

        cols = ("dept","id","name","gpa","attend","score")
        self.reps_tree = ttk.Treeview(reps, columns=cols, show="headings", height=8)
        for c, txt, w, anchor in [("dept","Dept",120,"w"),("id","ID",100,"w"),("name","Name",180,"w"),("gpa","GPA",60,"center"),("attend","Attend %",80,"center"),("score","Score",70,"center")]:
            self.reps_tree.heading(c, text=txt); self.reps_tree.column(c, width=w, anchor=anchor)
        self.reps_tree.grid(row=1, column=0, columnspan=7, sticky="nsew", padx=6, pady=6)
        reps.grid_columnconfigure(6, weight=1)
        tt = ttk.LabelFrame(f, text="Timetable Optimizer (DP: Weighted Interval Scheduling)")
        tt.pack(side=tk.TOP, fill=tk.BOTH, expand=True, padx=10, pady=10)

        editor = ttk.Frame(tt); editor.pack(side=tk.TOP, fill=tk.X, padx=6, pady=6)
        self.slot_code = tk.StringVar(); self.slot_start = tk.StringVar(); self.slot_end = tk.StringVar(); self.slot_weight = tk.StringVar(value="1")
        ttk.Label(editor, text="Subject Code").grid(row=0, column=0, padx=4, pady=4, sticky="w")
        ttk.Entry(editor, textvariable=self.slot_code, width=10).grid(row=0, column=1, padx=4, pady=4)
        ttk.Label(editor, text="Start (HH:MM)").grid(row=0, column=2, padx=4, pady=4, sticky="w")
        ttk.Entry(editor, textvariable=self.slot_start, width=8).grid(row=0, column=3, padx=4, pady=4)
        ttk.Label(editor, text="End (HH:MM)").grid(row=0, column=4, padx=4, pady=4, sticky="w")
        ttk.Entry(editor, textvariable=self.slot_end, width=8).grid(row=0, column=5, padx=4, pady=4)
        ttk.Label(editor, text="Weight").grid(row=0, column=6, padx=4, pady=4, sticky="w")
        ttk.Entry(editor, textvariable=self.slot_weight, width=6).grid(row=0, column=7, padx=4, pady=4)
        ttk.Button(editor, text="Add/Update Slot", command=self._add_update_slot).grid(row=0, column=8, padx=8, pady=4)

        cols2 = ("code","start","end","weight")
        self.catalog_tree = ttk.Treeview(tt, columns=cols2, show="headings", height=8)
        for c, txt, w, anchor in [("code","Subject",100,"w"),("start","Start",80,"center"),("end","End",80,"center"),("weight","Weight",70,"center")]:
            self.catalog_tree.heading(c, text=txt); self.catalog_tree.column(c, width=w, anchor=anchor)
        self.catalog_tree.pack(side=tk.TOP, fill=tk.X, padx=6, pady=6)
```

```
        # Optimizer
        opt = ttk.Frame(tt); opt.pack(side=tk.TOP, fill=tk.X, padx=6, pady=6)
        self.opt_sid = tk.StringVar()
        ttk.Label(opt, text="Student ID").pack(side=tk.LEFT)
        ttk.Entry(opt, textvariable=self.opt_sid, width=12).pack(side=tk.LEFT, padx=6)
        ttk.Button(opt, text="Optimize Timetable", command=self._optimize_timetable).pack(side=tk.LEFT, padx=6)

        self.opt_output = tk.Text(tt, height=6)
        self.opt_output.pack(side=tk.TOP, fill=tk.BOTH, expand=True, padx=6, pady=6)

    def _pick_reps(self):
        try:
            alpha = float(self.alpha_var.get())
            beta = float(self.beta_var.get())
            if alpha < 0 or beta < 0: raise ValueError("Weights must be non-negative")
            if alpha + beta == 0: raise ValueError("At least one weight must be > 0")
            topk = int(self.topk_var.get())
            data = self.reg.choose_class_representatives(top_per_dept=topk, alpha=alpha, beta=beta)
            for i in self.reps_tree.get_children(): self.reps_tree.delete(i)
            for dept, lst in data.items():
                for stu, score in lst:
                    self.reps_tree.insert("", tk.END, values=(dept, stu.student_id, stu.name, f"{stu.gpa():.2f}", f"{stu.attendance_rate():.1f}", f"{score:.1f}"))
        except Exception as e:
            messagebox.showerror("Error", str(e))

    def _add_update_slot(self):
        code = (self.slot_code.get() or "").strip().upper()
        start = (self.slot_start.get() or "").strip()
        end = (self.slot_end.get() or "").strip()
        weight = float(self.slot_weight.get() or 1)
        if not code or not start or not end:
            messagebox.showwarning("Input", "Fill Subject, Start and End."); return
        try:
            smin = time_to_minutes(start)
            emin = time_to_minutes(end)
            self.reg.set_subject_slot(code, smin, emin, weight)
            self.reg.save_catalog()
            self._refresh_catalog_table()
            messagebox.showinfo("Saved", f"Slot saved for {code}.")
        except Exception as e:
            messagebox.showerror("Error", str(e))

    def _refresh_catalog_table(self):
        for i in self.catalog_tree.get_children(): self.catalog_tree.delete(i)
        for code, slot in self.reg.list_subject_slots().items():
            self.catalog_tree.insert("", tk.END, values=(code, minutes_to_time(int(slot["start"])), minutes_to_time(int(slot["end"])), f"{slot.get('weight',1.0)}"))
```

```
    def _refresh_catalog_table(self):
        for i in self.catalog_tree.get_children(): self.catalog_tree.delete(i)
        for code, slot in self.reg.list_subject_slots().items():
            self.catalog_tree.insert("", tk.END, values=(code, minutes_to_time(int(slot["start"])), minutes_to_time(int(slot["end"])), f"{slot.get('weight',1.0)}"))

    def _optimize_timetable(self):
        sid = (self.opt_sid.get() or "").strip()
        if not sid:
            messagebox.showwarning("Input", "Enter a Student ID."); return
        try:
            chosen = self.reg.optimize_timetable_for(sid)
            if not chosen:
                self.opt_output.delete("1.0", tk.END)
                self.opt_output.insert(tk.END, "No schedulable subjects found (check catalog and enrollments).")
                return
            self.opt_output.delete("1.0", tk.END)
            self.opt_output.insert(tk.END, "Optimal non-overlapping subjects:\n")
            for code in chosen:
                slot = self.reg.get_subject_slot(code)
                self.opt_output.insert(tk.END, f"  {code}  {minutes_to_time(int(slot['start']))}-{minutes_to_time(int(slot['end']))}  w={slot.get('weight',1.0)}\n")
        except Exception as e:
            messagebox.showerror("Error", str(e))

if __name__ == "__main__":
    try:
        App().mainloop()
    except KeyboardInterrupt:
        print("\nInterrupted.")
```

Rubrics for CLO2, CLO3, CLO4, and CLO5 Assessment (Aligned with PLO2 and PLO5)

CLO2 – Apply Data Structures in Problem-Solving (PLO2: Problem Analysis)

Criteria	Very Poor	Poor	Good	Very Good	Excellent
Selection and abstraction of data structures	No appropriate structure selected	Inappropriate structure, unclear reasoning	Appropriate structure selected	Well-reasoned choice of structure	Accurate, efficient selection with justification
Implementation and logic correctness	Incomplete or incorrect	Functional with major flaws	Mostly correct implementation	Correct and mostly efficient	Fully accurate and optimized
Presentation and explanation	Unable to explain	Vague explanation	Satisfactory explanation	Clear and logical explanation	Highly professional and insightful
Plagiarism check	>35%	30–35%	20–30%	10–20%	<10%

CLO3 – Apply Recursion and Sorting Algorithms (PLO2: Problem Analysis)

Criteria	Very Poor	Poor	Good	Very Good	Excellent
Use of recursion or brute-force strategy	No usage or incorrect logic	Attempted but flawed	Functional logic, lacks clarity	Correct and appropriate logic	Efficient and elegant implementation
Integration of sorting algorithms	Not attempted	Wrong or inefficient method	Proper but basic implementation	Correct with clear logic	Optimized with advanced sorting techniques
Presentation and justification	Not understandable	Limited explanation	Reasonable explanation	Strong and clear presentation	Excellent delivery with technical insights
Plagiarism check	>35%	30–35%	20–30%	10–20%	<10%

CLO4 – Apply Greedy and Dynamic Algorithms (PLO2: Problem Analysis)

Criteria	Very Poor	Poor	Good	Very Good	Excellent
Application of algorithm strategy	No algorithm applied or incorrect	Poor fit for problem	Functional but lacks depth	Relevant and mostly correct	Innovative and well-suited approach
Problem breakdown and reasoning	No clear breakdown	Weak or inconsistent logic	Partial decomposition	Logical and systematic	Fully structured and insightful analysis
Explanation and presentation	Not coherent	Minimal or confusing	Acceptable and mostly clear	Strong, confident explanation	Excellent articulation and technical clarity
Plagiarism check	>35%	30–35%	20–30%	10–20%	<10%

CLO5 – Analyze Problems Using Tools and Algorithms (PLO5: Tool Usage)

Criteria	Very Poor	Poor	Good	Very Good	Excellent
Tool usage (Python, IDE, etc.)	No tools used	Attempted but ineffective	Correct tool with minor limitations	Proper, effective tool use	Advanced usage with recognition of limitations
Problem analysis and model formulation	No analysis provided	Vague or incorrect	Basic analysis with gaps	Structured and mostly complete	Excellent formulation and clear insights
Demonstration and discussion	Unclear or not demonstrated	Weak demonstration	Satisfactory with partial logic	Clear and well-explained	Excellent articulation and tool demonstration
Plagiarism check	>35%	30–35%	20–30%	10–20%	<10%