

LAB 12.5: IMPLEMENTATION OF SORTING AND SEARCHING ALGORITHMS USING AI ASSISTANCE

N.Goutham

2303A51209

B-04

TASK 1 - Merge Sort Implementation using AI Assistance

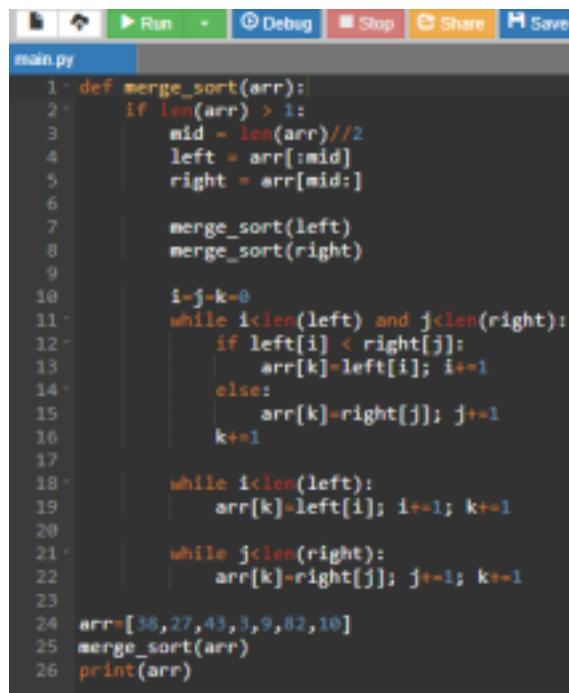
QUESTION

Generate a Python program that implements Merge Sort to sort a list in ascending order including time & space complexity.

PROMPT

Generate Python code for Merge Sort with function merge_sort(arr) including complexity.

CODE



The screenshot shows a Python code editor with a dark theme. The file is named 'main.py'. The code implements the Merge Sort algorithm. It defines a function 'merge_sort' that takes an array 'arr' as input. If the length of 'arr' is greater than 1, it calculates the midpoint 'mid', splits the array into 'left' (elements from index 0 to mid) and 'right' (elements from index mid to end). It then recursively calls 'merge_sort' on both 'left' and 'right'. After the recursive calls, it merges the sorted arrays back into 'arr'. The merge process involves three nested loops: one to iterate through the left array, one to iterate through the right array, and one to copy elements back into 'arr'. Finally, it prints the sorted array. The code is numbered from 1 to 26.

```
1 def merge_sort(arr):
2     if len(arr) > 1:
3         mid = len(arr)//2
4         left = arr[:mid]
5         right = arr[mid:]
6
7         merge_sort(left)
8         merge_sort(right)
9
10        i=j=k=0
11        while i<len(left) and j<len(right):
12            if left[i] < right[j]:
13                arr[k]=left[i]; i+=1
14            else:
15                arr[k]=right[j]; j+=1
16                k+=1
17
18            while i<len(left):
19                arr[k]=left[i]; i+=1; k+=1
20
21            while j<len(right):
22                arr[k]=right[j]; j+=1; k+=1
23
24 arr=[38,27,43,3,9,82,10]
25 merge_sort(arr)
26 print(arr)
```

```

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i=j=k=0
        while i<len(left) and j<len(right):
            if left[i] < right[j]:
                arr[k]=left[i]; i+=1
            else:
                arr[k]=right[j]; j+=1
            k+=1
            while i<len(left):
                arr[k]=left[i]; i+=1; k+=1
            while j<len(right):
                arr[k]=right[j]; j+=1; k+=1

arr=[38,27,43,3,9,82,10]
merge_sort(arr)
print(arr)

```

OUTPUT

```

[3, 9, 10, 27, 38, 43, 82]

...Program finished with exit code 0
Press ENTER to exit console.

```

[3, 9, 10, 27, 38, 43, 82]

EXPLANATION

Merge Sort divides the list and merges sorted halves efficiently.

TASK-02

Binary Search Implementation

QUESTION

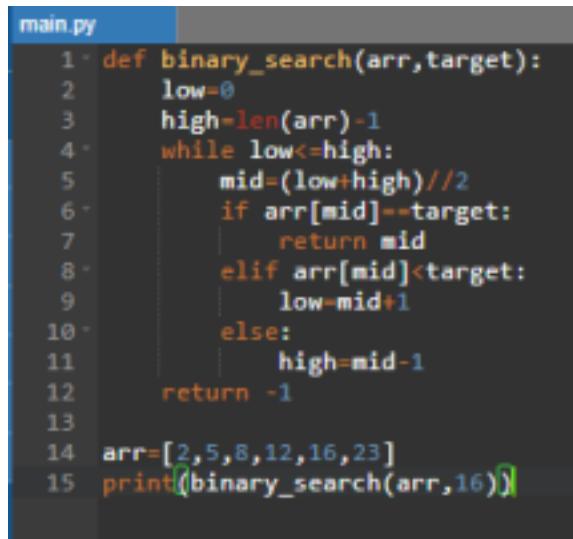
Create a binary search function returning index or -1.

PROMPT

Generate Python function

binary_search(arr,target).

CODE



The image shows a code editor window with a dark theme. The tab bar at the top has a single tab labeled "main.py". The main area contains Python code for a binary search function. The code is numbered from 1 to 15. It defines a function "binary_search" that takes two parameters: "arr" and "target". The function initializes "low" to 0 and "high" to the length of the array minus one. It then enters a loop where it calculates the midpoint ("mid") as the average of "low" and "high". If the element at "mid" is equal to the target, it returns "mid". If the element is less than the target, it sets "low" to "mid+1". If the element is greater than the target, it sets "high" to "mid-1". If the loop exits without finding the target, it returns -1. Below the function definition, there is a line of code that creates an array "arr" with elements [2, 5, 8, 12, 16, 23] and prints the result of calling the "binary_search" function with "arr" and the target value 16.

```
main.py
1 def binary_search(arr,target):
2     low=0
3     high=len(arr)-1
4     while low<=high:
5         mid=(low+high)//2
6         if arr[mid]==target:
7             return mid
8         elif arr[mid]<target:
9             low=mid+1
10        else:
11            high=mid-1
12    return -1
13
14 arr=[2,5,8,12,16,23]
15 print(binary_search(arr,16))
```

def binary_search(arr,target):

 low=0

 high=len(arr)-1

 while low<=high:

 mid=(low+high)//2

 if arr[mid]==target:

 return mid

 elif arr[mid]<target:

 low=mid+1

 else:

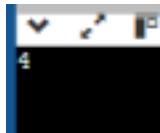
 high=mid-1

 return -1

arr=[2,5,8,12,16,23]

print(binary_search(arr,16))

OUTPUT



4

EXPLANATION

Binary search works on sorted lists and halves searches space.

TASK 3 – HEALTHCARE APPOINTMENT SYSTEM

Healthcare Appointment System

QUESTION

Search using ID and sort by fee/time.

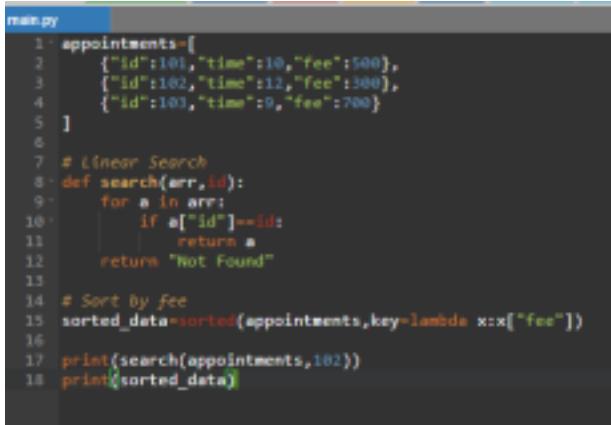
PROMPT

Suggest searching and sorting algorithms.

CODE

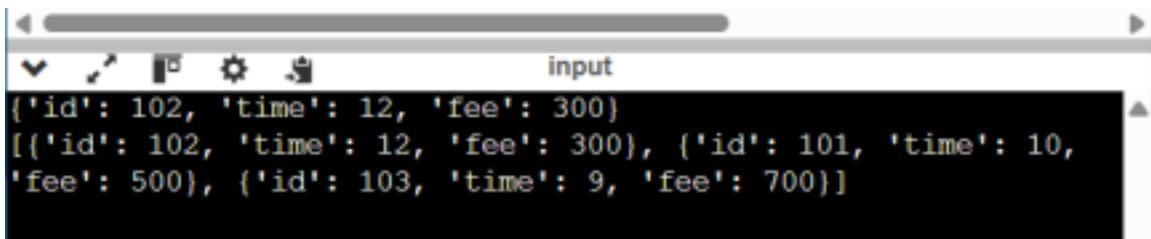
```
appointments=[  
    {"id":101,"time":10,"fee":500},  
    {"id":102,"time":12,"fee":300},  
    {"id":103,"time":9,"fee":700}  
]  
  
def search(arr,id):  
    for a in arr:  
        if a["id"]==id:  
            return a  
sorted_data=sorted(appointments,key=lambda x:x["fee"])
```

```
print(search(appointments,102))
print(sorted_data)
```



```
main.py
1 appointments=[
2     {"id":101,"time":10,"fee":500},
3     {"id":102,"time":12,"fee":300},
4     {"id":103,"time":9,"fee":700}
5 ]
6
7 # Linear Search
8 def search(arr,id):
9     for a in arr:
10         if a["id"]==id:
11             return a
12     return "Not Found"
13
14 # Sort by fee
15 sorted_data=sorted(appointments,key=lambda x:x["fee"])
16
17 print(search(appointments,102))
18 print(sorted_data)
```

OUTPUT



```
input
{'id': 102, 'time': 12, 'fee': 300}
[{'id': 102, 'time': 12, 'fee': 300}, {'id': 101, 'time': 10, 'fee': 500}, {'id': 103, 'time': 9, 'fee': 700}]
```

{'id': 102, 'time': 12, 'fee': 300}

EXPLANATION

Linear Search for ID, Sorting by fee using built-in-sort.

TASK 4 – RAILWAY RESERVATION SYSTEM

Search tickets and sort bookings

QUESTION

Search using Ticket ID and sort by date.

PROMPT

Recommend efficient algorithms.

CODE

```
main.py
1 tickets=[{"id":1,"date":5}, {"id":2,"date":2}, {"id":3,"date":8}]
2
3 def search_ticket(arr,id):
4     for t in arr:
5         if t["id"]==id:
6             return t
7
8 sorted_tickets=sorted(tickets,key=lambda x:x["date"])
9
10 print(search_ticket(tickets,2))
11 print(sorted_tickets)
```

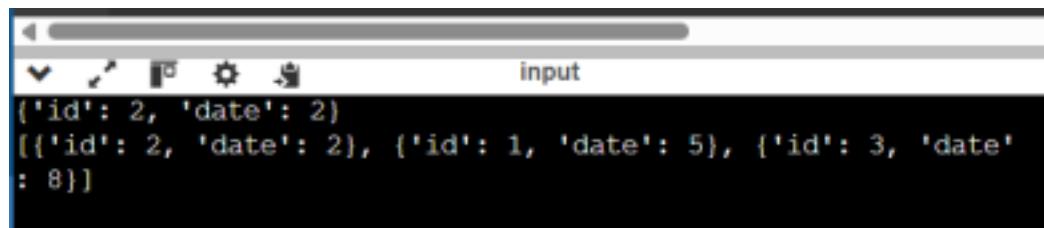
```
tickets=[{"id":1,"date":5}, {"id":2,"date":2}, {"id":3,"date":8}]

def search_ticket(arr,id):
    for t in arr:
        if t["id"]==id:
            return t
```

```
sorted_tickets=sorted(tickets,key=lambda x:x["date"])

print(search_ticket(tickets,2))
print(sorted_tickets)
```

OUTPUT



A screenshot of a terminal window titled "input". The window shows Python code being run. The code defines a list of dictionaries `tickets` and two functions: `search_ticket` and `sorted_tickets`. It then prints the result of `search_ticket(2)` and `sorted_tickets`. The output is: `{'id': 2, 'date': 2}` followed by the sorted list: `[{'id': 2, 'date': 2}, {'id': 1, 'date': 5}, {'id': 3, 'date': 8}]`.

```
{'id': 2, 'date': 2}
```

EXPLANATION

Efficient for small datasets using linear search

TASK 5

Hostel Allocation System

QUESTION

Search by student ID and sort by room.

PROMPT

Suggest optimized algorithms.

CODE

```
main.py
1 rooms=[{"id":11,"room":302}, {"id":12,"room":101}, {"id":13,"room":205}]
2
3 def search(arr,id):
4     for r in arr:
5         if r["id"]==id:
6             return r
7
8 sorted_rooms=sorted(rooms,key=lambda x:x["room"])
9
10 print(search(rooms,12))
11 print(sorted_rooms)
12
13
14
15
16
```

```
rooms=[{"id":11,"room":302}, {"id":12,"room":101}, {"id":13,"room":205}]
```

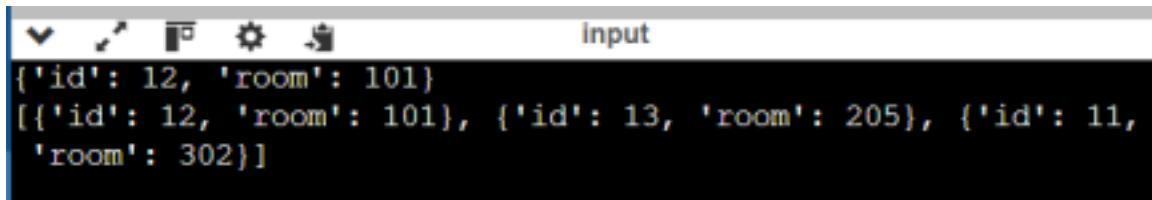
```
def search(arr,id):
    for r in arr:
        if r["id"]==id:
```

```
return r

sorted_rooms=sorted(rooms,key=lambda x:x["room"])

print(search(rooms,12))
print(sorted_rooms)
```

OUTPUT



```
{'id': 12, 'room': 101}
[{'id': 12, 'room': 101}, {'id': 13, 'room': 205}, {'id': 11, 'room': 302}]
```

```
{'id': 12, 'room': 101}
```

EXPLANATION

Simple search and sorting improves allocation visibility.

TASK 6 – MOVIE STREAMING PLATFORM

QUESTION

Search by movie ID and sort by rating.

PROMPT

Recommend algorithms.

CODE

```
movies=[
    {"id":1,"rating":8.2},
    {"id":2,"rating":9.1},
    {"id":3,"rating":7.5}
]
def search(arr,id):
    for m in arr:
```

```

if m["id"]==id:
    return m

sorted_movies=sorted(movies,key=lambda x:x["rating"],reverse=True)

print(search(movies,3))
print(sorted_movies)

```

```

main.py
1 movies=[
2     {"id":1,"rating":8.2},
3     {"id":2,"rating":9.1},
4     {"id":3,"rating":7.5}
5 ]
6
7 def search(arr,id):
8     for m in arr:
9         if m["id"]==id:
10            return m
11
12 sorted_movies=sorted(movies,key=lambda x:x["rating"],reverse=True)
13
14 print(search(movies,3))
15 print(sorted_movies)

```

OUTPUT

```

input
{'id': 3, 'rating': 7.5}
[{'id': 2, 'rating': 9.1}, {'id': 1, 'rating': 8.2}, {'id': 3, 'rating': 7.5}]

```

{'id': 3, 'rating': 7.5}

Explanation

Sorting by rating helps recommendation engines

TASK 7 – AGRICULTURE MONITORING SYSTEM

QUESTION

Search by crop ID and sort by yield.

PROMPT

Use AI reasoning.

CODE

```
main.py
1 crops=[
2     {"id":1,"yield":40},
3     {"id":2,"yield":55},
4     {"id":3,"yield":30}
5 ]
6
7 def search(arr,id):
8     for c in arr:
9         if c["id"]==id:
10             return c
11
12 sorted_crops=sorted(crops,key=lambda x:x["yield"])
13
14 print(search(crops,2))
15 print(sorted_crops)
16
```

```
crops=[
    {"id":1,"yield":40},
    {"id":2,"yield":55},
    {"id":3,"yield":30}
]

def search(arr,id):
    for c in arr:
        if c["id"]==id:
            return c

sorted_crops=sorted(crops,key=lambda x:x["yield"])

print(search(crops,2))
print(sorted_crops)
```

OUTPUT

```
input
{'id': 2, 'yield': 55}
[{"id": 3, "yield": 30}, {"id": 1, "yield": 40}, {"id": 2, "yield": 55}]
{'id': 2, 'yield': 55}
```

EXPLANATION

Helps farming prioritize crop monitoring.

TASK 8 – AIRPORT FLIGHT MANAGEMENT

QUESTION

Search by Flight ID and sort by time.

PROMPT

Recommend algorithms.

CODE

```
main.py
1 flights=[
2     {"id":101,"time":18},
3     {"id":102,"time":10},
4     {"id":103,"time":22}
5 ]
6
7 def search(arr,id):
8     for f in arr:
9         if f["id"]==id:
10             return f
11
12 sorted_flights=sorted(flights,key=lambda x:x["time"])
13
14 print(search(flights,101))
15 print(sorted_flights)
16
```

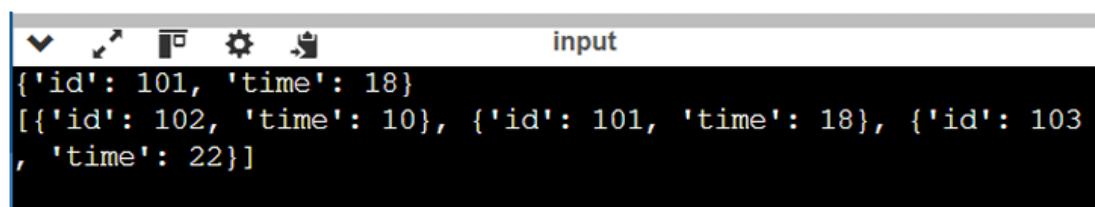
```
flights=[{"id":101,"time":18}, {"id":102,"time":10}, {"id":103,"time":22}]
```

```
def search(arr,id):
    for f in arr:
        if f["id"]==id:
            return f
```

```
sorted_flights=sorted(flights,key=lambda x:x["time"])

print(search(flights,101))
print(sorted_flights)
```

OUTPUT



```
input
{'id': 101, 'time': 18}
[{'id': 102, 'time': 10}, {'id': 101, 'time': 18}, {'id': 103, 'time': 22}]
```

```
{'id': 101, 'time': 18}
```

EXPLANATION

Sorting improves schedule management