**Space-time complexity analysis**

***Algorithm #1: Binary search***

*Time complexity analysis:*

public static <T extends Comparable<T>> int binarySearch(T[] array, T value) {

----------------------------------------------------------------------------- 1

------------------------------------------------------------------ 1

------------------------------------------------------------------- Log₂n

-------------------------------------------------------- Log₂n

------------------------------------ Log₂n

------------------------------------------------------------------- 1

----------------------------------- Log₂n

------------------------------------------------------------------- (Log₂n)/2

------------------------------------------------------------------- (Log₂n)/2

----------------------------------------------------------------------------- 1

**Time complexity:** O(Log n)

int a = 0;

int b = array.length - 1;

while (a <= b) {

int mid = a + (b - a) / 2;

if (value.compareTo(array[mid]) == 0) {

return mid;

} else if (value.compareTo(array[mid]) < 0) {

b = mid - 1;

} else {

a = mid + 1;

}

}

return -1;

}

*Space complexity analysis:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable** | **Size of 1 A.V** | **Quantity of A.V** |
| **Input** | array  value | ¿?  ¿? | n  1 |
| **Auxiliary** | a  b | 32 bits  32 bits | 1  1 |
| **Output** | mid  -1 | 32 bits  32 bits | 1  1 |
| **Total** | | | n+5 |
| **Total space complexity** | | | Θ(n) |

***Algorithm #2: Priority calculation***

*Time complexity analysis:*

public Double valuePerSection(Section[] sections) {

------------------------- n+1

------------------------- n

------------------------- n

------------------------- 1

------------------------- 1

**Time complexity:** O(n)

---------------------- 1 \* n

----------------------------------- 1

------------------------- 1

---- 1

------------------------- 1

----------------------------------- 1

---------------------------------------------- 1

----------------------------------- 1

----------- 1

--------------------------------------------- 1

**Time complexity:** O(n)

for (int i = 1; i < sections.length+1; i++) {

if (getRow() + 1 >= sections[i-1].getStart() &&

getRow() + 1 <= sections[i-1].getEnd()) {

return 10.0 \* i;

}

}

return Double.MIN\_VALUE;

}

public Double getPriority(Long initTime, Section[] sections) {

Double valuePerSection = valuePerSection(sections);

Double priority = valuePerSection;

if (seat.getType() == SeatType.FIRST\_CLASS) {

if (preference == PassengerPreference.MEDICAL\_ATTENTION) {

return valuePerSection \* 1.9999999999;

}

if (getAge() >= 60) {

priority += 2.2;

}

priority += 2.7 - (2.7 / (miles + 1));

}

priority += 5 / ((System.currentTimeMillis() - initTime) / 1000);

return priority;

}

*Space complexity analysis:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable** | **Size of 1 A.V** | **Quantity of A.V** |
| **Input** | initTime  sections | 64 bits  64 bits | 1  n |
| **Auxiliary** | valuePerSection | 64 bits | 1 |
| **Output** | priority | 64 bits | 1 |
| **Total** | | | n+3 |
| **Total space complexity** | | | Θ(n) |

***Algorithm #3: Passengers reading***

*Time complexity analysis:*

public static HashTable<String, Passenger> loadPassengersTxtFrom(String path, LoadToTable tableAction) {

-------------------------------------------------------------------------------------------------- 1

----------------------------------------------------------------------------- 2

------------------------------------------------------------------- 1

-------------- 1

-------------------------------------------------------------------------------------------------- 1

-------------------------------------------------------------------------------------------------- n+1

--------------------------------------------------------------------------------------- n

--------------------------------------------------------------------------------------- n

--------------------------------------------------------------------------------------- n

--------------------------------------------------------------------------------------- n

----------------------------------- n

----------------------------------- n

----------------------------------- n

------------------------- n

----------------------------------- n

----------------------------------- n

-------------------------------------------------------------------------------------------------- 1

-------------------------------------------------------------------------------------------------- 2

-------------------------------------------------------------------------------------------------- 1

**Time complexity:** O(n)

File file = new File(path);

if (!file.exists()) throw new FileNotFoundException();

HashTable<String, Passenger> hashTable = new HashTable<>();

BufferedReader reader = new BufferedReader(new InputStreamReader(new FileInputStream(file)));

String line = reader.readLine();

while ((line = reader.readLine()) != null) {

Passenger passenger = new Passenger();

passenger.setName(line.split(";")[0]);

passenger.setId(line.split(";")[1]);

passenger.setNationality(line.split(";")[2]);

passenger.setBirthday(LocalDate.of(Integer.parseInt(line.split(";")[3].split("/")[2]),

Integer.parseInt(line.split(";")[3].split("/")[1]),

Integer.parseInt(line.split(";")[3].split("/")[0])));

passenger.setMiles(Integer.parseInt(line.split(";")[4]));

passenger.setSeat(new Seat(

SeatType.values()[Integer.parseInt(line.split(";")[5])],

new Location(Integer.parseInt(line.split(";")[7]), line.split(";")[6])));

passenger.setPreference(PassengerPreference.values()[Integer.parseInt(line.split(";")[8])]);

hashTable.put(passenger.getId(), passenger);

tableAction.load(passenger);

}

reader.close();

if (hashTable.size() == 0) return null;

return hashTable;

}

*Space complexity analysis:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable** | **Size of 1 A.V** | **Quantity of A.V** |
| **Input** | path  action | 16 bits  ¿? | n  1 |
| **Auxiliary** | file  line  reader | ¿?  ¿?  ¿? | 1  1  1 |
| **Output** | hashtable | ¿? | 1 |
| **Total** | | | n+5 |
| **Total space complexity** | | | Θ(n) |