INTEGRATIVE TASK 1

DAVIDE FLAMINI CAZARAN - A00381665 - SYSTEMS ENGINEERING

NICOLÁS CUÉLLAR MOLINA - A00394970 - SYSTEMS ENGINEERING

ANDRES CABEZAS GUERRERO - A00394772 - SYSTEMS ENGINEERING

ICESI UNIVERSITY

FACULTY OF ENGINEERING

COMPUTING AND DISCRETE STRUCTURES I

PROBLEM SPECIFICATION TABLE

CLIENT	Airline
USER	Designated flight crew member
FUNCTIONAL REQUIREMENTS	- R1: Passengers Load into the System - R2: Passengers Arrival Registration - R3: Aircraft Boarding Order - R4: Aircraft Disembarking Order - R5: Aircraft Disembarking Order
CONTEXT OF THE PROBLEM	The issue at hand is the inefficiency in the airline's boarding and disembarking process, which is caused by the absence of a system that enables passengers to board and disembark efficiently. This results in delays and wasted time for both the flight crew and passengers. The goal is to improve this process to achieve a more comfortable and satisfying travel experience for the airline's customers.
NON-FUNCTIONAL REQUIREMENTS	- RN1: Efficiency passenger information retrieval - RN2: Intuitive user interface - RN3: The project must be uploaded to the GitHub platform and must have changes that allow the evolution of the project to be tracked.

REQUIREMENT SPECIFICATIONS

Name or identifier	R1: Passengers Load into the System		
Summary	The system must allow loading the passenger information corresponding to a flight through a plain text file.		
Inputs	**input name**	**Datatype**	**Selection or repetition condition**
	data	txt (String)	
General activities necessary to obtain the results	1. Load the txt file through the loadData() method. 2. Read the lines of the txt file (each line corresponds to the information of a passenger). 3. Create the passengers in the program using the information read. 4. Add the passengers to the hashtable. 5. Print the information read.		
Result or postcondition	The loaded passenger information or an error message.		
Outputs	**output name**	**Datatype**	**Selection or repetition condition**
	confirmation	String	

Name or identifier	R2: Passengers Arrival Registration		
Summary	The system must allow registering the arrival of a passenger to the boarding gate.		
Inputs	**input name**	**Datatype**	**Selection or repetition condition**
	arriveOrderId	String	
General activities necessary to obtain the results	1. Enter a for loop that will end when the iterator is greater than the number of passengers. 2. Request the ID of the first arriving passenger. 3. Each time the ID is entered, increase the iterator by 1. 4. Once the for loop finishes, display a confirmation to the user.		
Result or postcondition	The registered arrival order or an error message.		
Outputs	**output name**	**Datatype**	**Selection or repetition condition**
	confirmation	String	

Name or identifier	R3: Aircraft Boarding Order		
Summary	The system must allow showing the crew member in charge the order in which passengers should board the plane, taking into account the arrival order and the priorities of first class.		
Inputs	**input name**	**Datatype**	**Selection or repetition condition**
General activities necessary to obtain the results	1. Add the passengers to a priority queue, where the priority is defined based on their row, arrival order, first-class status, and any applicable special conditions. 2. Remove each user from the priority queue. 3. Print the information of the passenger. 4. Display the order in which they are removed from the priority queue.		
Result or postcondition	Display how passengers should board or an error message.		
Outputs	**output name**	**Datatype**	**Selection or repetition condition**
	confirmation	String	

Name or identifier	R4: Aircraft Disembarking Order		
Summary	The system must allow setting the order of passenger departure, taking into account the configuration of the rows on the plane.		
Inputs	**input name**	**Datatype**	**Selection or repetition condition**
General activities necessary to obtain the results	1. Add the passengers to a reversed priority queue, where the priority is based on the passenger's distance from the aisle and the time of their arrival. 2. Remove the passengers from the priority queue and add them to a stack. 3. Remove the passengers from the stack and print their IDs. 4. Display the order of passenger departure on the screen.		
Result or postcondition	The order of passenger departure or an error message.		
Outputs	**output name**	**Datatype**	**Selection or repetition condition**
	confirmation	String	

Name or identifier	R5: Passenger search		
Summary	The system must allow efficient searching for a passenger's information once they arrive at the boarding area.		
Inputs	**input name**	**Datatype**	**Selection or repetition condition**
	id	String	
General activities necessary to obtain the results	1. Search for the passenger in the hashtable using their ID. 2. Check if the passenger exists in the hashtable. 3. If the passenger exists, retrieve their information using their ID.		
Result or postcondition	The passenger's information or an error message.		
Outputs	**output name**	**Datatype**	**Selection or repetition condition**
	confirmation	String	

ENGINEERING DESIGN PROCESS

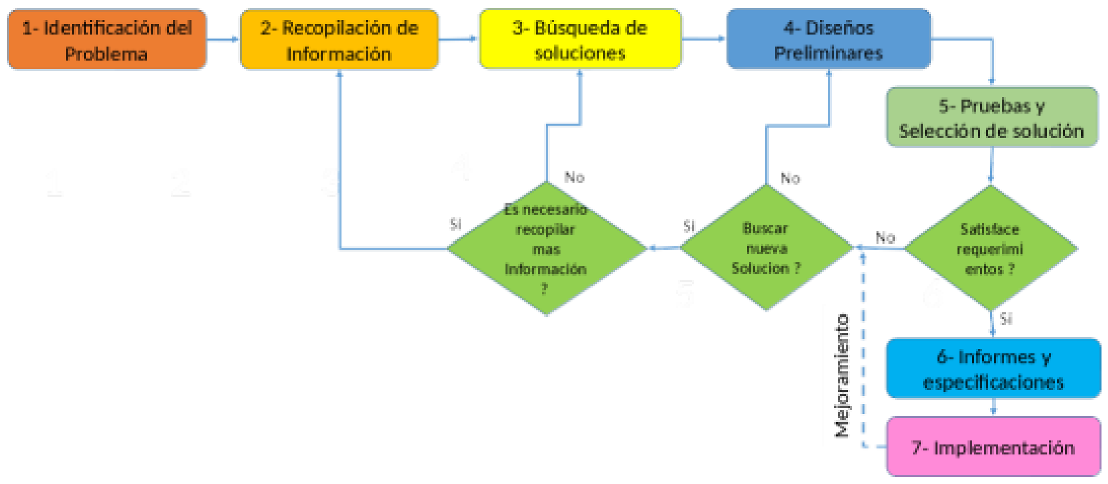
Context Problem

The issue at hand is the inefficiency in the airline's boarding and disembarking process, which is caused by the absence of a system that enables passengers to board and disembark efficiently. This results in delays and wasted time for both the flight crew and passengers. The goal is to improve this process to achieve a more comfortable and satisfying travel experience for the airline's customers.

Solution Development

Considering the context and nature of the problem at hand, we have opted to utilize the Engineering Method for the development of an effective and efficient solution. This systematic approach allows for a thorough analysis and understanding of the problematic situation, identification of necessary requirements, and establishment of clear and achievable objectives for the solution.

Based on the description of the Engineering Method from Paul Wright's book, "Introduction to Engineering," we have defined the following flowchart, which we will follow in the development of the solution.


1- Identificación del Problema
2- Recopilación de Información
3- Búsqueda de soluciones
4- Diseños Preliminares
5- Pruebas y Selección de solución
Satisface requerimientos ?
No
Buscar nueva Solucion ?
No
Si
Es necesario recopilar mas Información ?
No
Si
Si
6- Informes y especificaciones
7- Implementación
Mejoramiento

Step 1: Problem Identification:

<u>Symptoms and Needs:</u>

- The airline needs to improve the order in the process of boarding and disembarking the aircraft.
- The corresponding passenger information for a flight needs to be loaded.
- Passengers need to be located and their arrival to the boarding area needs to be registered.

- It is necessary to show the order in which passengers should board the plane to the crew.
- Special rules must be established for the boarding of first class passengers, prioritizing other data such as accumulated miles, special attention required, seniority, or other relevant data.
- For disembarking, an exit order must be established for each row taking into account proximity to the aisle or order of arrival.

Causes:

- Lack of a system that automatically manages passengers and their information.
- Lack of a database that allows for passenger management.
- Lack of a model process for registering passenger arrival and departure.
- Inefficiency in the boarding process.
- Inefficiency in the disembarkation process.

Problem definition:

The problem consists of inefficiency in the boarding and disembarkation of passengers on an airline's planes. Currently, this process can be inefficient and can generate confusion, delays, and discomfort for both passengers and airline personnel. This is due to the lack of an automated system that allows for the management of passenger boarding and disembarkation. Therefore, the main objective is to develop an automated system that allows for the registration of passenger arrival to the boarding area and establishes the order of passenger boarding and disembarkation, displaying it to the responsible crew member. This is done while keeping in mind that the implementation of this system seeks to maintain a high standard of efficiency and reliability.

Step 2: Information Gathering:

Once the problem has been identified and the needs appropriately defined, the engineer begins to gather the necessary information and data to solve it. Of course, the type of information required and the appropriate techniques for its collection depend on the nature of the problem to be solved. For example, systems engineers carry out a bidding of requirements.

In the case of the problem at hand, improving the order in the aircraft's boarding and deboarding process, information gathering is essential to understanding the current situation and finding effective and efficient solutions.

In this initial phase of the project, a comprehensive collection of information will be made about current procedures in the aviation industry, as well as best practices in the field of passenger management at airports and in boarding and deboarding systems of aircraft. Information will be sought from various sources.

In conclusion, the main objective of this phase is to better understand the problem and the different types of variables that affect it, thus establishing a solid knowledge base to propose appropriate and efficient solutions to the problem.

Firstly, for a correct abstraction of the problem, it is essential to understand the structure and functioning of airplanes, as this contributes to ensuring safety and efficiency in flights. The present document (Ministry of Education, 2018) aims to analyze the structure of airplanes and their impact on their performance and safety. According to the document, the basic structure of an airplane is:

- Fuselage: It is the main structure of the airplane and is responsible for holding all the other parts together. Additionally, it is the part of the airplane where passengers, crew, and cargo are housed.
- Wings: They are responsible for generating the necessary lift for the airplane to fly. They are composed of several sections, including the leading edge, trailing edge, and intrados.
- Empennage: It is the structure located at the rear of the airplane and is made up of the vertical stabilizer, horizontal stabilizer, and rudder and elevator. Its function is to maintain the stability of the airplane during flight.
- Landing gear: It is the system that allows the airplane to take off and land. It is composed of main wheels, nose wheels, and brake and steering systems.

Bibliographic reference:

Ministry of Education. (2018). Analysis of the structure of airplanes and their impact on performance and safety. Retrieved from https://www.curriculumnacional.cl/614/articles-215790_recurso_pdf.pdf

Second, it is important to understand how the boarding and disembarking process of a plane works to solve the problem because it allows identifying the different factors that can influence the flight's punctuality as well as the passenger experience. To understand the boarding process in the airline ANA (All Nippon Airways), which describes its boarding process, it has the following phases:

Check-in: Passengers check-in at the boarding gate and show their boarding pass and passport to be verified. Boarding announcement: The start of boarding is announced, and passengers are invited to board by groups and sections according to their service class, seat location, etc. Boarding: Passengers aboard the plane and head to their seats. Door closure: Once all passengers have boarded, the plane's door is closed, and it prepares for takeoff.

Bibliographic reference:

ANA. (2021). Boarding Procedures. Retrieved on April 21, 2023, from https://www.ana.co.jp/es/mx/travel-information/boarding-procedures/

Lastly, for the development of this work, it is essential to investigate the optimization of the boarding process on an airplane as it is a crucial task for airlines today. As the aviation industry continues to grow, airlines are looking for ways to improve efficiency and reduce passenger wait times. One of the main things that affect boarding time is the process itself, which can be complex and time-consuming if not handled properly. This importance is discussed in the document "Improvement Plan of the Organization in the Boarding Room of the Flight Operated by Lufthansa at Bogotá Station" (2016). The document describes an improvement plan for the organization in the boarding room of the flight operated by Lufthansa at the Bogotá station. The plan's objective is to optimize the boarding and disembarking process for passengers to improve the customer experience and reduce wait time.

The document mentions that thanks to the use of strategies to optimize the boarding process on an airplane, significant improvements were achieved in the airline's time and organization, which is one of the objectives we seek in the development of this writing.

Bibliographic reference:

Carrera, J. A. V. (2016). Improvement Plan of the Organization in the Boarding Room of the Flight Operated by Lufthansa at Bogotá Station (Bachelor's thesis). Fundación Universitaria Los Libertadores.

Step 3: Creative solutions search: In this phase, the creation of creative solution ideas for the problem is intended:

It is important to generate a preliminary list of solutions to have a more accurate approximation to the problem. This phase is fundamental to solve the problem effectively, and it is also important for steps 4 and 5 of the engineering method, where less feasible ideas are discarded, and the best solution is selected. The technique used for the project was brainstorming, which is based on the free and spontaneous generation of ideas without judging them, and in the subsequent selection and combination of those considered most relevant and useful. The team members were asked to propose solution ideas based on their research and knowledge of the problem, and the resulting solution ideas are as follows:

- Creation of a QR program: One possible solution is to create a program that allows passengers to enter their information and receive a QR code that allows them to board the plane in the appropriate order.

- Facial recognition system: Implementing a facial recognition system that can identify passengers and determine the order in which they arrive at the gate. The system can be connected to the airline's database and allow for quick passenger identification.

- Creation of a program: The proposed solution to improve the order in the boarding and deplaning process consists of using a combination of data structures to store and sort passenger information. The implementation of the program would be done in Java.

- Queue system: A queue system can be implemented at the gate to separate first-class passengers from regular passengers and provide them with special attention. The system can be designed to prioritize passengers with special needs and the elderly.

- Introduction of informative screens: Informative screens can be introduced in the boarding area that display information about passengers and their boarding procedures. These screens can be linked to the airline's database and allow information to be displayed quickly.

Step 4: Transition from idea formulation to preliminary designs:

In this phase, ideas that are not feasible will be discarded, and promising ideas will be shaped and modified to form feasible drafts and designs.

First, we will start with the ideas that will be discarded, which are as follows:

- Facial recognition system: The idea of using a facial recognition system can be discarded because it can pose privacy and security issues for passengers and is expensive and difficult to maintain.
- Queue system: The idea of a queue system can be discarded as it can create a sense of injustice and inequality among passengers, and it can be difficult to determine who should be served first.

- Introduction of informative screens: The introduction of informative screens can provide useful information to passengers, but it can also cause confusion if the information is not accurately updated and may require a significant investment in hardware and software.

It should be clarified that the main reason why the above-mentioned ideas will be discarded is due to the scope of the project. The team assigned to solve the project does not have the scope for the ideas presented. It is important to understand the scope of our project and the possibilities and magnitude of the team for an optimal and adequate solution. With this, it is guaranteed that the selected solutions can be executed and carried out.

Next, we will discuss the ideas that are not discarded, specifying them and making their respective analytical specification models:

Creation of a program

The proposed solution to improve the order in the process of boarding and disembarking the plane consists of using a combination of data structures to store and order passenger information.

Analytical model:

Objective:

To create a program that allows improving the process of boarding and disembarking passengers on a plane through the use of data structures and special rules for first-class passengers. The objective is to increase the efficiency of the process and reduce waiting times and congestion at boarding gates.

Functionalities:

- Reading data from a plain text file and storing it in a hashtable.
- Establishing priorities according to the passengers' location on the plane.
- Creating a priority queue to sort passengers according to their arrival order and location.
- Using another priority queue for the plane's disembarking, prioritizing each passenger's distance to the aisle.

Technologies and tools:

- Java as a programming language.
- IDEs such as Eclipse or NetBeans for program development.
- Data structures: stacks, priority queues, and hashtables.

Flowchart:

- Reading data from a plain text file.
- Storing data in hashtable.
- Establishing priorities.
- Creating priority queue.
- Show how passengers board the plane.
- Creating a priority queue for plane disembarking.
- Show how passengers disembark from the plane.
- Storing ordered information in a priority queue.

Expected Outcome:

- The program is expected to allow for a more efficient organization of the process of boarding and disembarking passengers on a plane, reducing waiting times and congestion at boarding gates.
- The use of data structures and special rules for first-class passengers should improve the passenger experience and increase customer satisfaction.

<u>Creation of a QR program</u>

The idea is to create a Java program that allows passengers to enter their information and receive a QR code that will allow them to board the plane in the appropriate order. The program would store passenger information and their location on the plane, and use an algorithm to determine the boarding order. Passengers could scan their QR code upon arrival at the boarding gate, allowing them to board the plane at the appropriate time.

<u>Analytical model:</u>

Objective:

To improve the boarding and exiting process of the plane by creating a Java program that allows passengers to enter their information and receive a QR code that will allow them to board the plane in the appropriate order.

Functionalities:

- The program will allow passengers to enter their personal information such as their name, seat number, and location on the plane.
- The program will generate a unique QR code for each passenger, which will be used to determine the boarding order.
- The program will order passengers according to their location on the plane and their arrival time, using an appropriate data structure (as seen in class).
- The program will allow passengers to scan their QR code at the boarding gate to board the plane at the appropriate time.
- The program will continuously update the data structure as passengers board the plane to control boarding.

Technologies and Tools:

- Java: programming language used to develop the program.
- QRGen: open-source library used to generate unique QR codes for each passenger.
- IDE such as Eclipse or NetBeans for program development.
- Data structures: an appropriate data structure will be used to order passengers according to their location on the plane and their arrival time.

Flowchart:

- Reading passenger data from a form in the program.
- Generating unique QR codes for each passenger using the QRGen library.
- Storing passenger information and their location on the plane in an appropriate data structure.
- Ordering passengers according to their location on the plane and their arrival time using the data structure.
- Continuously updating the data structure.
- Scanning QR codes at the boarding gate.

Expected results:

- A more organized and efficient boarding and exiting process for the plane.

- A reduction in waiting time and passenger confusion.
- An improvement in passenger experience and customer satisfaction.
- Increased efficiency for the airline by reducing boarding and exiting time for the plane.

Step 5: Evaluation and Selection of the Best Solution:

Finally, in this writing, the selection of the best solution will be carried out. For this purpose, a series of criteria were established that will be scored from 1 to 5 (where 5 is the highest score considered excellent and 1 is the lowest score considered regular) and will allow choosing the best solution proposal. The criteria are as follows:

- Efficiency: Which of the proposed solutions is more efficient in terms of time and resources?

- Ease of implementation: Which of the solutions is easier to implement and maintain? Is research necessary to solve the problem?

- Scalability: Can the solution handle large amounts of data and users? Can it adapt to future changes in the number of users and data?

- Usability: Is the solution easy to use for end-users? Is it easy to understand and use for airport crew and staff?

- Security: Does the solution guarantee the privacy and security of passenger data?

- Cost: What is the cost of implementing and maintaining the solution? Is it financially viable for the airline?

The methodology used to evaluate the criteria based on the solution proposals was to have the team meet and agree on a score (1-5) for each criterion. The results were as follows:

<u>Creation of a program:</u>

Criteria	***Score (1-5)***
Efficiency	5
Ease of implementation	4
Scalability	4
Usability	5
Security	5
Cost	5
Total	28

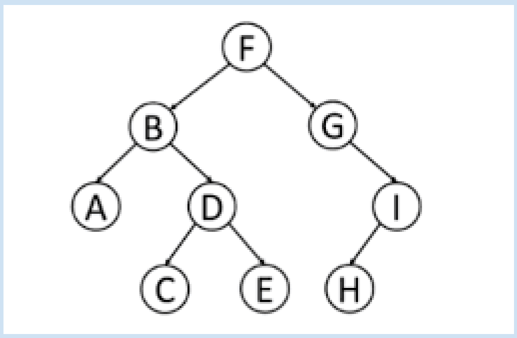
Creation of a QR program

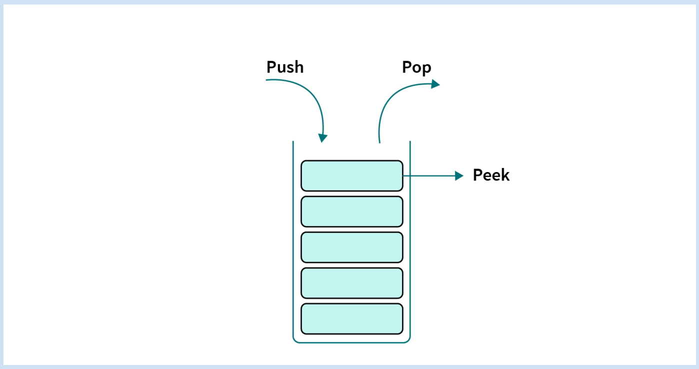
Criteria	***Score (1-5)***
Efficiency	3
Ease of Implementation	3
Scalability	4
Usability	5
Security	4
Cost	4
Total	23

According to the previous vote, the solution chosen to improve the order in the process of boarding and leaving the plane was the creation of a Java program. This solution obtained the highest score in terms of the sum of all criteria; it fulfilled the criteria of efficiency, as it allows for a faster and more organized boarding and leaving process, and scalability, as it can handle large amounts of data. In addition, the solution is easily implementable in Java through data structures. Overall, this solution offers a significant improvement in the boarding and leaving process of the plane, improving the passenger experience and reducing stress on the crew.

In conclusion, after analyzing several solutions to improve the process of boarding and leaving the plane, the decision was made to create a Java program that allows for the organization of boarding and leaving planes. This solution meets the criteria established for evaluating the ideas, including efficiency in the process, ease of use for passengers, and technical feasibility. It is expected that this solution will contribute to a more organized and pleasant travel experience for passengers and crew.

TAD DEFINITIONS

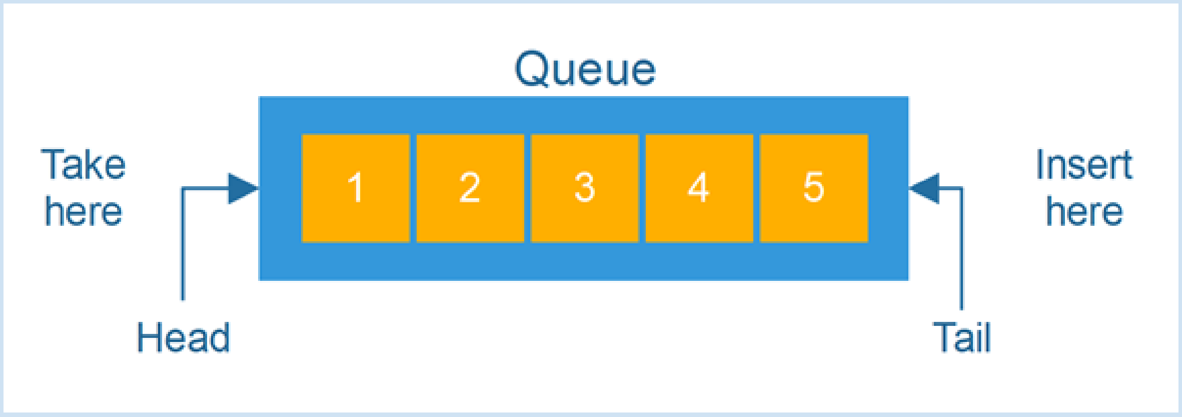
TAD <BST>
BST = 
Invariant: {left < elemento < right }
Main operations: CreateTree(): Element -> BST AddElement(): BST x Element -> BST RemoveElement(): BST x Element -> BST SearchElement(): BST x Element -> Element GetRoot(): BST -> Element PrintInOrder(): BST -> Text

TAD <Stack>
Stack = 
Invariant: Let S be a stack, such that: - S is a sequence of elements s_1, s_2, s_3, ..., s_n. - The size of S is equal to n. - The last element added to the stack is s_n. - The first element that will be removed from the stack is s_n. - A stack is empty if and only if its size is zero. - The last element added to the stack is the first one to be removed (LIFO).

Main operations:
CreateStack(): Element -> Stack
Push(): Stack x Element -> Stack
Pop(): Stack -> Element
Top(): Element -> Element
IsEmpty(): Stack -> Boolean

TAD <Queue>

Queue =



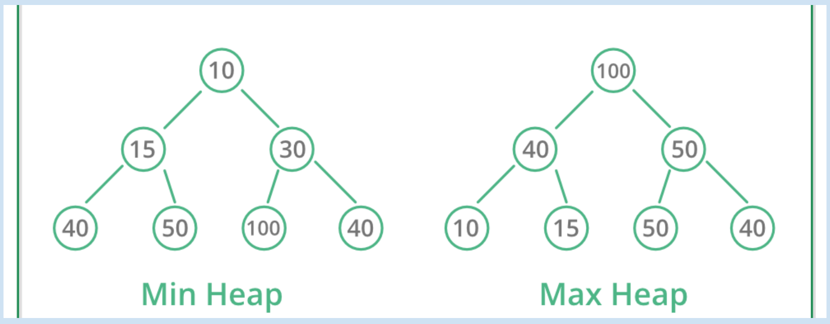
Invariant:
Let Q be a queue such that:

- Q is a sequence of elements q_1, q_2, q_3, ..., q_n.
- the size of Q is equal to n.
- The first element added to the queue is q_1.
- The last element added to the queue is q_n.
- The first element to be removed from the queue is q_1.

A queue is empty if and only if its size is zero.
The first element added to the queue is the first one to be removed (FIFO).

Main operations:
CreateQueue(): Element -> Queue
Offer(Enqueue): Queue x Element -> Queue
Peek(Front): Queue -> Element
Poll(Dequeue): Queue -> Element
IsEmpty(): Queue-> Boolean

TAD <Heap>

Heap = 

Invariant:

For a maximum heap: If n is a node in the heap, and lc(n) and rc(n) are the left and right child nodes, respectively, then it must be true that:

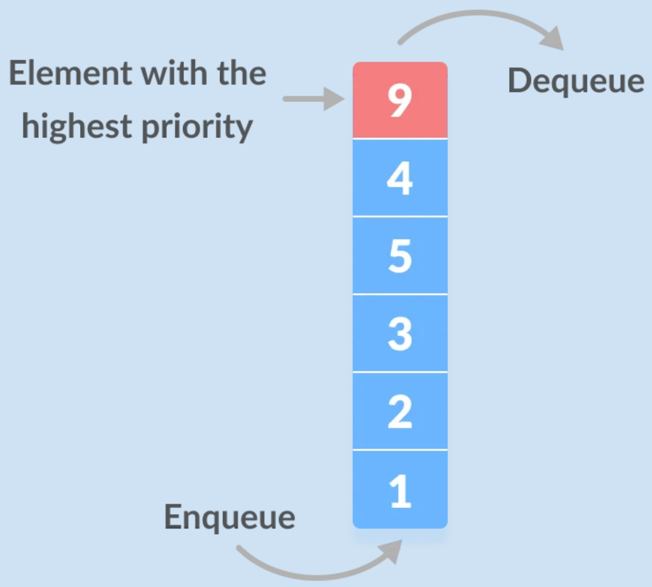
value(n) >= value(lc(n))
value(n) >= value(rc(n))

For a minimum heap: If n is a node in the heap, and lc(n) and rc(n) are the left and right child nodes, respectively, then it must be true that:
value(n) <= value(lc(n))
value(n) <= value(rc(n))

Here, value(n) refers to the value stored in node n.

Main operations:
CreateHeap(): Integer -> Heap
Insert(): Heap x Integer x Element -> Heap
GetMax(): Heap -> Integer x Element
ExtractMax(): Heap -> Integer x Element
Remove(): Heap x Integer -> Heap
ChangeKey() -> Integer x Integer -> Heap

TAD <Priority Queue>



Priority Queue =

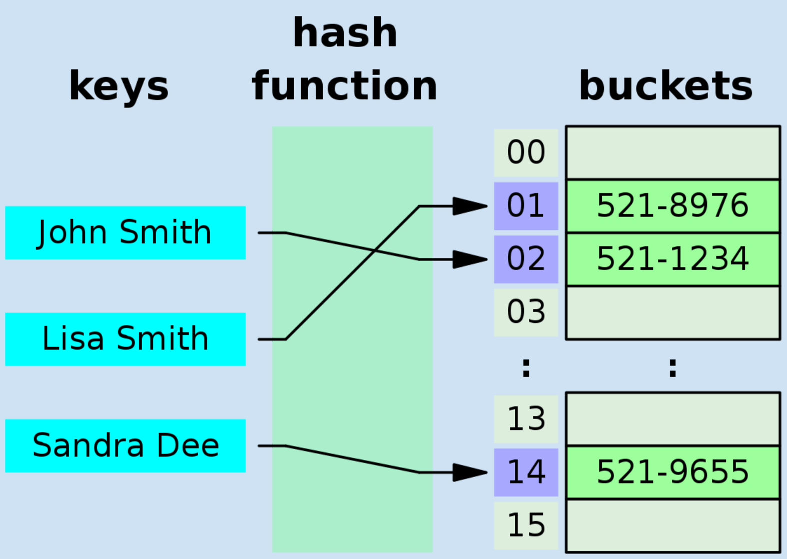
Invariant:
For a priority queue P, if x is the element with the highest priority in P, then for any other element y in P, it is satisfied that:

priority(y) ≤ priority(x)

where "priority(y)" is the priority of element "y" and "priority(x)" is the priority of element "x".

Main operations:
CreatePriorityQueue(): Element -> PriorityQueue
Offer(Enqueue)(): Queue x Element x Integer -> Queue
Peek(Front)(): Queue -> Element
Poll(Dequeue)(): Queue -> Element
IsEmpty(): Priority Queue -> Boolean
Size(): Priority Queue -> Integer

TAD <HashTable>



HashTable =

Invariant:

Let T be a hash table with n entries and h be a hash function that maps each key k to an integer between 0 and n-1.

For every pair of keys k1 and k2, if k1 ≠ k2, then h(k1) ≠ h(k2).

Main operations:
CreateHashTable(): Integer -> HashTable
Insert(): HashTable x Integer x Element -> HashTable
Remove(): HashTable x Integer x Element -> HashTable
SearchElement(): HashTable x Integer -> Element
Size(): HashTable -> integer
Clear(): HashTable -> HashTable

TESTS:

Stack Test

Name	Class	Scenery
setUp	StackTest	The Stack is empty.

Test Objective: Verify that the Stack ends up empty after some operations.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Stack	isEmpty()	setUp	N/A	The Stack is empty
Stack	isEmpty()	setUp	Node: Key-1;Value-"one"	The Stack isn't empty
Stack	isEmpty()	setUp	Node1: Key-1;Value-"one"' Delete Node1	The Stack is empty

Test Objective: Verify that the push operation correctly inserts.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Stack	push()	setUp	Node: Key-1;Value-"one"	The Stack add to Node
Stack	push()	setUp	Node1: Key-1;Value-"one"' Node2: Key-2;Value-"two"'	The Stack add to Node1 and Node2
Stack	push()	setUp	Node1: Key-1;Value-"one"' Node2: Key-2;Value-"two"'	The push the Node in the position correct

Test Objective: Verify that the top operation returns the correct value.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**

Stack	top()	setUp	N/A	null
Stack	top()	setUp	Node1: Key-1;Value-”one”’	The top is Node1
Stack	top()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’	The top is Node2

Test Objective: Verify that the pop operation correctly removes an element.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Stack	pop()	setUp	N/A	null
Stack	pop()	setUp	Node1: Key-1;Value-”one”’ Delete Node1	The Stack is empty
Stack	pop()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’ Delete Node1	The top is Node1

HashTable Tests

Name	Class	Scenery
setUp	HashTableTest	The HashTable is empty.

Test Objective: Verify that the add operation correctly inserts into the HashTable.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Hash	add()	setUp	Key-”one”;Value-1 Key-”two”;Value-2	The Hash add with the key correct
Hash	add()	setUp	Key-”one”;Value-1 Key-”two”;Value-2 Key-"neo";Value-3	The Hashadd with the key correct
Hash	add()	setUp	Key-”one”;Value-1 Key-”two”;Value-2 Key-"three";Value-3 Key-"four";Value-4 Key- "five";Value-5	The size of the Hash is correct The value is correct of the key Null when the key isn’t exist

Test Objective: Verify that the getValue operation correctly searches for a value.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Hash	getValue()	setUp	Key-”one”;Value-1 Key-”two”;Value-2	The Hash find the value correct of the key
Hash	getValue()	setUp	Key-”one”;Value-1 Key-”two”;Value-2	Null when the key isn’t exist
Hash	getValue()	setUp	Key-”one”;Value-1 Key-”two”;Value-2 Key-"three";Value-3 Key-"four";Value-4 Key- "five";Value-5	The Hash find the value correct of the key Null when the key isn’t exist

Test Objective: Verify that the remove operation works correctly.

Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Hash	remove()	setUp	Key-”one”;Value-1 Key-”two”;Value-2 remove “one”	The Hash remove to “one”
Hash	remove()	setUp	Key-”one”;Value-1 Key-”two”;Value-2 remove “three”	The Hash don’t remove to “three” because it isn’t exist
Hash	remove()	setUp	Key-”one”;Value-1 Key-”two”;Value-2 Key-"three";Value-3 Key-"four";Value-4 Key- "five";Value-5 remove “two” remove “four”	The Hash remove to “one” and “four”

Queue Test

Name	Class	Scenery
setUp	QueueTest	The Queue is empty.

Test Objective: Verify the function isEmpty in the Queue.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Queue	isEmpty()	setUp	N/A	The Queue is Empty
Queue	isEmpty()	setUp	Node1: Key-1;Value-”one”’	The Queue isn´t Empty
Queue	isEmpty()	setUp	Node1: Key-1;Value-”one”’ poll()	The Queue is Empty

Test Objective: Verify that the poll operation correctly deletes an element.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Queue	poll()	setUp	N/A	Null because the Queue don’t have nodes
Queue	poll()	setUp	Node1: Key-1;Value-”one”’	The Queue is Empty
Queue	poll()	setUp	Node1: Key-1;Value-”one”’ poll() Node2: Key-2;Value-”two”’	The peek isn’t Node1

Test Objective: Verify that the peek operation correctly gets the top element.				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Queue	peek()	setUp	N/A	Null because the Queue don’t have nodes

Queue	peek()	setUp	Node1: Key-1;Value-”one”’ isEmpty()	The function return the Node1 and then, the Queue is Empty
Queue	peek()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’	The Queue isn´t Empty

Test Objective: Verify that the offer operation correctly inserts an element.

Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
Queue	offer()	setUp	Node: Key-1;Value-”one” peek()	The Queue is Empty
Queue	offer()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’ peek()	The Queue isn’t Empty
Queue	offer()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’ Node3: Key-3;Value-”three” Node4: Key-4;Value-”four” poll() poll()	The peek return the Node3 and then, the Queue is Empty The Queue isn’t Empty

PriorityQueue Tests

Name	Class	Scenery
setUp	PriorityQueueTest	The Queue is empty and with size 100

Name	Class	Scenery
setUpZeroSize	PriorityQueueTest	The Queue size is 0

Test Objective: Verify that the priority Queue insert the node correctly

Class	Method	Scenery	Entry Values	Expected result
IPriority Queue	maxHeapInsert()	setUp	Node1: Key-1;Value-”cat”’ Node2: Key-2;Value-”dog”’	The Priority Queue insert in the position correct
IPriority Queue	maxHeapInsert()	setUp	Node1: Key-1;Value-”cat”’	The Priority Queue insert in the position correct
IPriority Queue	maxHeapInsert()	setUp	Node1: Key-1;Value-”cat”’ Node2: Key-2;Value-”dog”’ Node3: Key-2;Value-”mouse” Node4: Key-2;Value-”elephant”	The Priority Queue insert in the position correct

Test Objective: Verify that the heap Maximum get the maximum after organizing

Class	Method	Scenery	Entry Values	Expected result
IPriority Queue	heapMaximun()	setUpZeroSize	N/A	Null because the size is 0
IPriority Queue	heapMaximun()	setUp	Node1: Key-1;Value-”one”’	The function return el Node1

IPriority Queue	heapMaximun()	setUp	Node2: Key-2;Value-”two” Node4: Key-4;Value-”four” Node1: Key-1;Value-”one”’ Node3: Key-3;Value-”three”	The function return el Node4

Test Objective: Verify that the priority Queue extracts the node with the higher key

Class	Method	Scenery	Entry Values	Expected result
IPriority Queue	heapExtractMax()	setUpZeroSize	N/A	HeapUnderflow ´cause size 0
IPriority Queue	heapExtractMax()	setUp	Node1: Key-1;Value-”one”’	Function returns Node1 key(1)
IPriority Queue	heapExtractMax()	setUp	Node1: Key-2;Value-”two”’ Node2: Key-4;Value-”four”’ Node3: Key-1;Value-”one”’ Node4: Key-3;Value-”three”’	Function return Node4 Key(4)

Test Objective: Verify that the priority Queue increase the key of node correctly

Class	Method	Scenery	Entry Values	Expected result
IPriority Queue	HeapIncreaseKey()	setUpZeroSize	N/A	Throws KeyIsSmaller
IPriority Queue	HeapIncreaseKey()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’	Function returns Node2 key(2)
IPriority Queue	HeapIncreaseKey()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’	Function returns the Node correct

			Node3: Key-3;Value-”three” Node4: Key-4;Value-”four” Node6: Key-6;Value-”six” Node7: Key-7;Value-”seven” Node5: Key-5;Value-”five”	

Test Objective: Verify that the priority Queue insert the node correctly

Class	Method	Scenery	Entry Values	Expected result
IPriority Queue	minHeapIns ert()	setUp	Node1: Key-1;Value-”cat”’ Node2: Key-2;Value-”dog”’	The Priority Queue insert in the position correct
IPriority Queue	minHeapIns ert()	setUp	Node1: Key-1;Value-”cat”	The Priority Queue insert in the position correct
IPriority Queue	minHeapIns ert()	setUp	Node1: Key-1;Value-”cat”’ Node2: Key-2;Value-”dog”’ Node3: Key-2;Value-”mouse” Node4: Key-2;Value-”elephan t”	The Priority Queue insert in the position correct

Test Objective: Verify that the heap Minimum get the minimum after organizing

Class	Method	Scenery	Entry Values	Expected result
IPriority Queue	heapMinim un()	setUp	Node1: Key-2;Value-”hello”’ Node2: Key-3;Value-”world”’	Function returns Node1 key(2)
IPriority	heapMinim	setUpZeroS	N/A	Null Because the size is 0

Queue	un()	ize		
IPriority Queue	heapMinimun()	setUp	Node1: Key-2;Value-”hello”’ Node2: Key-2;Value-”world”’	Function returns Node 1 or 2 because they have the same key

Test Objective: Verify that the priority Queue extracts the node with the smallest key				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
IPriority Queue	heapExtract Minimun()	setUpZeroSize	N/A	Null Because the size is 0
IPriority Queue	heapExtract Minimun()	setUp	Node1: Key-2;Value-”hello”’ Node2: Key-3;Value-”world”’	Function return the Node1 and delete it of the Priority Queue
IPriority Queue	heapExtract Minimun()	setUp	Node1: Key-2;Value-”two”’ Node2: Key-4;Value-”four”’ Node3: Key-1;Value-”one”’ Node4: Key-3;Value-”three”’	Function return Node 3 and delete it, call the function again and return Node 1 because Node 3 does not exist no more

Test Objective: Verify that the priority Queue decrease a node key correctly				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
IPriority Queue	decreaseKey ()	setUp	Node: Key-1;Value-”one”	Throws KeyIsBigger ‘cause the new key is 2 and it isn’t correct
IPriority Queue	decreaseKey ()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’	The Function return the Node 2
IPriority Queue	decreaseKey ()	setUp	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’	The Function return Node 1, the insert Node 5 and return it, and finally insert Node 6 and return it because have the lowest key (-5)

			Node3: Key-3;Value-”three” Node4: Key-4;Value-”four” Node5: Key-(-1);Value-”-one” Node6: Key-(-5);Value-”-five”	

Heap Tests

Name	Class	Scenery
setUpBig	HeapTest	The Heap has size 10

Name	Class	Scenery
setUpSmall	HeapTest	The Heap has size 5

Test Objective: Verify that the Heap get the index of the son Left				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
IHeap	getLeft()	setUpBig	i:3 expected: 6	The function return 6
IHeap	getLeft()	setUpBig	i:0 expected: 1	The function return 1
IHeap	getLeft()	setUpBig	i:2 expected: 4	The function return 4

Test Objective: Verify that the Heap get the index of the son Right				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
IHeap	getRigth()	setUpBig	i:3 expected: 7	The function return 7
IHeap	getRigth()	setUpBig	i:0 expected: 2	The function return 2
IHeap	getRigth()	setUpBig	i:2 expected: 5	The function return 5

Test Objective: Verify that the Heap get the index of the parent				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**

IHeap	getParent()	setUpBig	i:3 expected: 1	The function return 1
IHeap	getParent()	setUpBig	i:0 expected: 0	The function return 0
IHeap	getParent()	setUpBig	i:7 expected: 3	The function return 3

Test Objective: Verify that the method accommodates the nodes in the generic array so that the max heap property is satisfied

Class	Method	Scenery	Entry Values	Expected result
IHeap	maxHeapify( )	setUpSmall	Node1: Key-1;Value-”A”’ Node2: Key-2;Value-”B”’ Node3: Key-3;Value-”C”’ Node4: Key-4;Value-”D”	The Heap organize the array of Nodes
IHeap	maxHeapify( )	setUpSmall	Node1: Key-1;Value-”A”’ Node2: Key-2;Value-”B”’ Node3: Key-3;Value-”C”’	The Heap organize the array of Nodes
IHeap	maxHeapify( )	setUpSmall	Node1: Key-1;Value-”A”’ Node2: Key-2;Value-”B”’ Node3: Key-3;Value-”C”’	The Heap organize the array of Nodes

Test Objective: Verify that the method leaves the minimum value in the root

Class	Method	Scenery	Entry Values	Expected result
IHeap	buildMaxHe ap()	setUpBig	Node1: Key-4;Value-”four”’	The Heap organize and build correctly

			Node2: Key-2;Value-”two”’ Node3: Key-8;Value-”eight”’ Node4: Key-5;Value-”five” Node5: Key-1;Value-”one” Node6: Key-6;Value-”six”	
IHeap	buildMaxHeap()	setUpBig	Node1: Key-5;Value-”five”’	The Heap organize and build correctly
IHeap	buildMaxHeap()	setUpBig	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’ Node3: Key-3;Value-”three” Node4: Key-4;Value-”four”	The Heap organize and build correctly

Test Objective: Verify that the Heap array is organized from smallest to largest

Class	Method	Scenery	Entry Values	Expected result
IHeap	heapSortMInToMax()	setUpBig	Node1: Key-1;Value-”A”’ Node2: Key-2;Value-”B”’ Node3: Key-3;Value-”C”’ Node4: Key-4;Value-”D” Node5: Key-5;Value-”E” Node6: Key-6;Value-”F” Node7: Key-7;Value-”D”	The Heap array from smallest to largest

IHeap	heapSortMInToMax()	setUpSmall	Node1: Key-5;Value-”E”’ Node2: Key-2;Value-”B”’ Node3: Key-8;Value-”H”’ Node4: Key-3;Value-”C” Node5: Key-1;Value-”A”	The Heap array organized without the node5 cause the extractMax
IHeap	heapSortMInToMax()	setUpSmall	Node1: Key-5;Value-”E”’ Node2: Key-2;Value-”B”’ Node3: Key-2;Value-”H”’ Node4: Key-1;Value-”C” Node5: Key-1;Value-”A”	The Heap array from smallest to largest

Test Objective: Verify that the method organize the nodes in the generic array so that the min heap property is satisfied				
Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
IHeap	minHeapify()	setUpSmall	Node1: Key-4;Value-”A”’ Node2: Key-3;Value-”B”’ Node3: Key-2;Value-”C”’ Node4: Key-1;Value-”D”	The Heap organize the array of Nodes
IHeap	minHeapify()	setUpSmall	Node1: Key-3;Value-”A”’ Node2: Key-2;Value-”B”’ Node3: Key-1;Value-”C”’	The Heap organize the array of Nodes

IHeap	minHeapify( )	setUpSmall	Node1: Key-4;Value-”A”’ Node2: Key-3;Value-”B”’ Node3: Key-2;Value-”C”’ Node4: Key-1;Value-”D”	The Heap organize the array of Nodes

Test Objective: Verify that the method leaves the minimum value in the root

Class	**Method**	**Scenery**	**Entry Values**	**Expected result**
IHeap	buildMinHe ap()	setUpBig	Node1: Key-4;Value-”four”’ Node2: Key-2;Value-”two”’ Node3: Key-8;Value-”eight”’ Node4: Key-5;Value-”five” Node5: Key-1;Value-”one” Node6: Key-6;Value-”six”	The Heap organize and build correctly
IHeap	buildMinHe ap()	setUpBig	Node1: Key-5;Value-”five”’	The Heap organize and build correctly
IHeap	buildMinHe ap()	setUpBig	Node1: Key-1;Value-”one”’ Node2: Key-2;Value-”two”’ Node3: Key-3;Value-”three” Node4: Key-4;Value-”four”	The Heap organize and build correctly

Test Objective: Verify that the Heap array is organized from largest to smallest

Class	Method	Scenery	Entry Values	Expected result
IHeap	heapSortMaxToMin()	setUpBig	Node1: Key-5;Value-”E”’ Node2: Key-2;Value-”B”’ Node3: Key-8;Value-”H”’ Node4: Key-3;Value-”C” Node5: Key-1;Value-”A” Node6: Key-4;Value-”D” Node7: Key-6;Value-”F” Node8: Key-7;Value-”G”	The Heap array from largest to smallest
IHeap	heapSortMaxToMin()	setUpSmall	Node1: Key-5;Value-”E”’ Node2: Key-2;Value-”B”’ Node3: Key-8;Value-”H”’ Node4: Key-3;Value-”C” Node5: Key-1;Value-”A”	The Heap array organized without the node5 cause the extractMin
IHeap	heapSortMaxToMin()	setUpSmall	Node1: Key-5;Value-”E”’ Node2: Key-2;Value-”B”’ Node3: Key-8;Value-”H”’ Node4: Key-3;Value-”C” Node5: Key-1;Value-”A”	The Heap array from largest to smallest

COMPLEXITY ANALYSIS

The following will be an analysis of the complexity of two of the longest non-recursive algorithms used in our proposed solution through Java code.

The first algorithm to be analyzed is:

```java
public String printListDisembarkationOrder() {
       String msj = "\n<< DISEMBARKATION LIST >> \n";
       Heap<Integer, String> dd = disembarkationOrder;

       Node<Integer, String> ps = null;
       Node<Integer, String> ps2 = null;
       for (int index = 0; index < disembarkationOrder.getArray().length; index +=
2) {


           if (index == 29) {
               break;
           }
           try {
               ps = dd.heapExtracMax(dd.getArray());
               ps2 = dd.heapExtracMax(dd.getArray());

               /////////////

               if (dd.getArray()[index] != null) {
                   if (ps.getKey().equals(ps2.getKey())) {
                       Passenger passenger1 =
passengersInfo.getValue(ps.getValue());
                       Passenger passenger2 =
passengersInfo.getValue(ps2.getValue());
                       if (compareArrival(passenger1, passenger2) == 1) {
                           msj += "" + (index + 1) + ") " + ps.getValue() + "\t"
                                   + ps.getKey() + "\n";
                           msj += "" + (index + 2) + ") " + ps2.getValue() + "\t"
                                   + ps2.getKey() + "\n";
                       } else {
                           msj += "" + (index + 1) + ") " + ps2.getValue() + "\t"
                                   + ps2.getKey() + "\n";
                           msj += "" + (index + 2) + ") " + ps.getValue() + "\t"
                                   + ps.getKey() + "\n";
                       }
                   } else {
                       msj += "" + (index + 1) + ") " + ps.getValue() + "\t"
                               + ps.getKey() + "\n";
                       msj += "" + (index + 2) + ") " + ps2.getValue() + "\t"
```

```
                                + ps2.getKey() + "\n";
                    }
                }

                //////// 7
            } catch (HeapUnderflow e) {
                System.out.println("Heap");
            }
        }

        setDesembarkationList(msj);

        return msj;
    }
```

Time complexity:

Line of code	Operation	Time complexity	Number of repetitions
4	Variable assignment	O(1)	1
5	Variable assignment	O(1)	1
6-22	For loop	O(n log n)	n/2
8-9	Index verification	O(1)	n/2
10	Exception handling	O(1)	n/2
12-13	Heap extraction (x2)	O(log n)	n/2

15-27	Comparisons and assignments	O(1)	n/2
29	Variable assignment	O(1)	1
31	Variable return	O(1)	1

Overall, the time complexity of this algorithm is O(n log n), where n is the length of the "disembarkationOrder" list. The most expensive operation within the for loop is the extraction of two elements from the heap, which is performed n/2 times, resulting in a total time complexity of O(n/2 log n), which simplifies to O(n log n).

Spatial complexity:

The analysis of spatial complexity refers to the amount of space (memory) that the algorithm requires for its execution. In this case, the spatial complexity of the algorithm can be divided into two parts:

1. Local variables: three local variables are created: msj, dd, ps, and ps2 in the function. The msj, ps, and ps2 variables are of constant size and do not depend on the input size, while dd is a reference to the existing disembarkationOrder object in the program. Therefore, the amount of space required for these variables is constant and can be considered O(1).

2. Data structures: an object of the Heap class and two objects of the Node class are used. The amount of space required by these data structures depends on the input size. The Heap object is initialized with the disembarkationOrder reference, and its size depends on the number of passengers who have not yet disembarked. The Node objects are created temporarily in each iteration of the loop and are removed in the same iteration. Therefore, the total space required by these data structures is O(N), where N is the number of passengers who have not yet disembarked.

In general, the spatial complexity of the algorithm can be considered O(N), where N is the number of passengers who have not yet disembarked.

The second algorithm to analyze is:

```
public void addPassengersToDataStructures(String passengersInfo) {
        String[] lines = passengersInfo.split("\n");
        plane.setTotalPassengers(lines.length - 1);
        plane.createBoardingAndDisembarkationOrder();
        for (int i = 1; i < lines.length; i++) {
            String[] infoPassenger = lines[i].split("::");
            Passenger passenger = new Passenger(infoPassenger[0], infoPassenger[1],
Integer.parseInt(infoPassenger[2]),
                    infoPassenger[3], Boolean.parseBoolean(infoPassenger[4]),
Boolean.parseBoolean(infoPassenger[5]),
                    Boolean.parseBoolean(infoPassenger[6]),
Integer.parseInt(infoPassenger[7]));
            /// Calcula la prioridad de abordaje de cada pasajero///
            passenger.setPriorityBoarding(calculateBoardingPriorityNew(passenger));
            /////////////////////////////////////////////////////////
            plane.getPassengersInfo().add(passenger.getId(), passenger);

            /// Calcula la prioridad de desbordaje de cada pasajero///

passenger.setPriorityDisembarking(calculateDisembarkationPriority(passenger));
            /////////////////////////////////////////////////////////

            try {

plane.getBoradingOrder().maxHeapInsert(plane.getBoradingOrder().getArray(),
                        new Node<>(passenger.getPriorityBoarding(),
passenger.getId()));
            } catch (KeyIsSmaller e) {
                System.out.println("No Sirvio");
            }

            try {

plane.getDisembarkationOrder().maxHeapInsert(plane.getDisembarkationOrder().getArra
y(),
                        new Node<>(passenger.getPriorityDisembarking(),
passenger.getId()));
            } catch (KeyIsSmaller e) {
                System.out.println("No Sirvio");
            }

        }
        //
```

```
    }
```

Time complexity:

Line of Code	Operation	Time Complexity	Number of Repetitions
2	split()	O(n)	1
3	setTotalPassengers()	O(1)	1
4	createBoardingAndDisembarkationOrder()	O(1)	1
5-25	for loop	O(n)	n-1
6	split()	O(n)	n-1
7	new Passenger()	O(1)	n-1
8	setPriorityBoarding()	O(1)	n-1
10	add()	O(1)	n-1
11	setPriorityDisembarking()	O(1)	n-1

13-16	maxHeapInsert()	O(log n)	2*(n-1)

In general, the time complexity of the addPassengersToDataStructures method is O(n log n), where n is the length of the input string passengersInfo. This is because the method contains a for loop that runs n-1 times (where n is the length of the string array lines) and within each iteration, operations of O(1) and O(log n) complexity are performed. The string split() operation also contributes to the time complexity, but its complexity is dominated by the for loop and is considered insignificant.

Spatial complexity:

The algorithm has a spatial complexity of O(n), where n is the number of passengers being added to the data structure.

This is because the main data structure, plane.getPassengersInfo(), is storing a Passenger object for each line in passengersInfo. Additionally, two additional data structures (plane.getBoradingOrder() and plane.getDisembarkationOrder()) are created, which also store information about the passengers.

All temporary variables (lines, infoPassenger, passenger) also have a size proportional to the number of passengers.

In summary, the spatial complexity of the algorithm is linear with respect to the number of passengers being added to the data structure.