Algorithm Complexity Analysis

• Summary:

"i": number of items;
"*m*": number of workstations for each operation;
"*k*": number of events processed;
Overall time complexity: O (*k* * *n* * *m*) (in the worst case).

• US01:

- "private boolean loadItems(String itemsFile)"

Line of Code	Operation	Complexity
1	Opening the file using a BufferedReader	O (1)
2	Skip the first line (header)	O (1)
3	Reading lines from the file	O (li) where "li" is the number of lines in the file
4	Splitting each line into na array of values	O (ci) where "ci" is the length of each line (number of fields)
5	Retrieving the number of values	O (1)
6	Extracting the first value (item ID)	O (1)
7	Converting the second value to a priority enum	O (1)
8	Initializing the list of operations	O (1)
9	Looping over remaining values to add operations to the list	O (o) where "o" is the number of operations
10	Creating new item	O (1)
11	Adding the item to the repository	O (1)
12	Incrementing the index	O (1)
13	Getting all itens and adding them	O (lw)
14	Printing the count of itens	O (1)
15	Returning true	O (1)
16	Catching I/O exceptions	O (1)
17	Printing error message	O (1)
18	Return false upon failure	O (1)

- "private boolean loadWorkStations(String machinesFile)"

Line of Code	Operation	Complexity
1	Method definition	O (1)
2	Opening a file for reading	O (1)
3	Variable initialization	O (1)
4	Reading the first line (header)	O (1)
5	Iterating over each line in the file	O (lw) where "lw" is the number of lines in the file
6	Splitting the line into na array	O (cw) where "cw" is the length of each line (number of fields)

7	Accessing the first element of the array	O (1)
8	Accessing the second elemento of the array	O (1)
9	Parsing the third elemento as na integer	O (1)
10	Creating a new "*WorkStation*" object	O (1)
11	Adding the "*WorkStation*" to the repository	O (1)
12	Getting all workstations and adding them	O (lw)
13	Printing the count of workstations	O (1)
14	Returning true	O (1)
15	Catching an exception	O (1)
16	Printing an error message	O (1)
17	Returning false	O (1)

- "public boolean loadData(String itemsFile, String workstationsFile)"

Line of Code	Operation	Complexity
1	Clear all events from the event repository	O (n)
2	Clear all items from the item repository	O (n)
3	Clear all workstations from the workstation repository	O (n)
4	Load items from the file (depends on "*loadItems*" method implementation)	O (li*ci)
5	Load workstations from the file (depends on "*loadWorkStations*" method implementation)	O (lw*cw)
6	Check if both items and workstations are loaded, print success, return true	O (1)
7	Print failure and return false	O (1)
8	Handle exception, print error message, and return false	O (1)

Therefore, the complexity of the data load (items and workstations) will be: O (li*(ci) + O (lw*cw)

- **US02 e US08:** **helper methods for both user stories**

- "private void updateItemAfterOperation(Item item)"

Line of Code	Operation	Complexity
1	Method definition	O (1)
2	Retrieving and incrementing the current operation index	O (1)
4	Comparing next operation index with operations size	O (1)
5	Setting the current operation index	O (1)
7	Else block initiation	O (1)
8	Setting all operations completed flag	O (1)
10	End of the method	O (1)

- "private WorkStation getFasterWorkStation(List<WorkStation> workStations)"

Line of Code	Operation	Complexity
1	Method definition	O (1)
2	Initializing "*fastestWorkStation*" variable	O (1)

3	Initializing "*fastestTime*" variable	O (1)
5	Loopping through workstations (n iterations)	O (n)
6	Comparing workstation time with "*fastestTime*"	O (1)
7	Updating "*fastestTime*"	O (1)
8	Updating "fastestWorkStation"	O (1)
10	End of the loop	O (1)
12	Returning "*fastestWorkStation*"	O (1)

- "private void addWorkStations(List<WorkStation> workStations)"

Line of Code	Operation	Complexity
1	Method definition	O (1)
3	Loopping through "*workStations*" (n iterations)	O (n)
4	Checking if operation exists in the map	O (1)
5	Adding a new key to the map if it doesn't exist	O (1)
6	Adding workstation to the list corresponding to the operation	O (1)
8	End of the loop	O (1)
10	End of the method	O (1)

- **US02:**

- "public void simulateWithoutPriority()"

Line of Code	Operation	Complexity
1	Initialization of "*currentTime*"	O (1)
2	"addWorkStations" method call	O (w)
3	Main event loop	O (k)
4 a 7	Update currentTime and totalTimeSimulator	O (1)
8	Loop to process events with the same EndTime	O (k*log(k))
9	Remove event from queue	O (log(k))
10 a 11	Get workstation from this event and make it available	O (1)
12 a 13	Get item from this event and make it available	O (1)
14	Loop over all items	O (i)
10	Check item status	O (1)
11	Gets current item operation	O (1)
12	Create list of available stations	O (1)
13	Loop over stations for operation	O (m)
14 a 15	if available add to list	O (1)
16	Check if there are available stations	O (1)
17	Select fastest station by calling "getFasterWorkStation " method	O (w)
18	Mark the workstation as unavailable	O (1)
19	Get workstation time	O (1)
20	Marks the item as in operation	O (1)
21	Create new event	O (1)
22	Add event to event queue	O (log(k))
23	Add event to event repository	O (1)
24	Calls the updateItemAfterOperation method	O (1)

The complexity of this method is: O (w + k·log(k)+i·(w+log(k)))

Where:

- O (k * log(k)) represents the operations in the event queue
- O (i·(w+log(k))) covers item processing and selection of available workstations for the operation

• US08:

- "public void simulateWithPriority()"

Line of Code	Operation	Complexity
1	Initialization of "*currentTime*"	O (1)
2	"addWorkStations" method call	O (w)
3	Main event loop	O (k)
4 a 7	Update currentTime and totalTimeSimulator	O (1)
8	Loop to process events with the same EndTime	O (k*log(k))
9	Remove event from queue	O (log(k))
10 a 11	Get workstation from this event and make it available	O (1)
12 a 13	Get item from this event and make it available	O (1)
14	Loop over all items	O (i)
10	Check item status	O (1)
11	Add item to priority queue	O (log(i))
12	Loop that processes items in the priority queue	O (i * log(i))
13	Gets next item with highest priority	O (log(i))
11	Gets current item operation	O (1)
12	Create list of available stations	O (1)
13	Loop over stations for operation	O (m)
14 a 15	if available add to list	O (1)
16	Check if there are available stations	O (1)
17	Select fastest station by calling "getFasterWorkStation " method	O (w)
18	Mark the workstation as unavailable	O (1)
19	Get workstation time	O (1)
20	Marks the item as in operation	O (1)
21	Create new event	O (1)
22	Add event to event queue	O (log(k))
23	Add event to event repository	O (1)
24	Calls the updateItemAfterOperation method	O (1)

The complexity of this method is: O (w + k·log(k)+ i·(log(i) + w+log(k)))

Where:

- O (k * log(k)) represents the operations in the event queue
- O (i·(log(i) + w+log(k))) covers item processing and selection of available workstations for the operation

- **US03:**

- "public int getTotalTime()"

Line of Code	Operation	Complexity
1	Method definition	O (1)
2	Returning the value of "*totalTimeSimulator*"	O (1)

The method only has O (1) complexity, however, totalTimeSimulator is updated once for each event processed, that is, k times.

- **US04:**

- "public Map<String, Integer> executionTimeOperation()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*"	O (1)
2	Iterating over all events from the repository	O (k)
3	Retrieving the operation string from the workstation	O (1)
4	Checking if the operation is already in the map	O (1)
5	Adding the operation with an initial value if not present	O (1)
6	Retrieving the current total execution time for the operation	O (1)
7	Updating the total execution time for the operation	O (1)
8	Returning the map of execution times	O (1)

The method only has O(n) complexity, as it is determined by the iteration between repository events (k times).

- **US05:**

- "public Map<WorkStation, Integer> totalTimeWorkStation()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*"	O (1)
2	Iterating over all events from the repository	O (k)
3	Retrieving the workstation associated with the event	O (1)
4	Checking if the worksation is already in the map	O (1)
5	Adding the workstation with an initial value if not present	O (1)
6	Retrieving the current total time for the workstation	O (1)
7	Updating the total time for the workstation	O (1)
8	Returning the map of total times per workstation	O (1)

- "public int sumGlobalTime()"

Line of Code	Operation	Complexity
1	Initializing a sum variable	O (1)
2	Iterating over the entries of the map	O (m)
3	Retrieving the value for the current workstation	O (1)
4	Accumulating the total time	O (1)
5	Returning the total sum of times	O (1)

- "public Map<WorkStation, Double> percentageOperationTimeGlobalTime()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*"	O (1)
2	Calling "*sumGlobalTime*" method to get total global time	O (m)
3	Iterating over the entries of the map from "*totalTimeWorkStation*"	O (k)
4	Retrieving the key (workstation) from the current entry	O (1)
5	Retrieving the value (total time) for the current workstation	O (1)
6	Checking if the workstation is already in the map	O (1)
7	Adding the workstation with an initial value if not present	O (1)
8	Calculating the percentage of operation time relative to global time	O (1)
9	Updating the percentage for the workstation	O (1)
10	Creating a list from the entries of the percentage map	O (k)
11	Sorting the list based on percentage values	O (k log(k))
12	Initializing a new "*LinkedHashMap*" to preserve order	O (1)
13	Iterating over the sorted list	O (n)
14	Adding entries to the sorted map	O (1)
15	Returning the sorted map of percentages	O (1)

Thus, the total complexity of this us is: O(m)+O(k)+O(k*log(k)) = O(m+k*log(k))

• US06 e US07:

- "public Map<Item, List<Event>> getEventsByItem()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*"	O (1)
2	Iterating over all events int the event reporitory	O (k)
3	Retrieving the item from the current event	O (1)
4	Checking if the item is already in the map	O (1)
5	Adding the item with a new "*ArrayList*" if not present	O (1)
6	Adding the current event to the list associated with the item	O (1)
7	End of the loop	O (1)
8	Returning the map of events organizated by item	O (1)

• US06:

- "private Map<String, Integer> getNumberEventsPerOperation()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*"	O (1)
2	Iterating over all events int the event reporitory	O (k)
3	Retrieving the operation from the current event	O (1)
4	Checking if the operation is already in the map	O (1)
5	Adding the operation with an initial count if not present	O (1)
6	Start of the else block	O (1)
7	Retrieving the current for the operation	O (1)
8	Updating the count for the operation	O (1)
9	End of the loop and if-else statement	O (1)
10	Returning the map of event counts per operation	O (1)

- "public Map<String, Double> averageTimePerOperation()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*"	O (1)
2	Calling "*getNumberEventsPerOperation*" to get events counts	O (k)
3	Iterating over the entries of the map from "*executionTimeOperation*" method	O (k)
4	Retrieving the key (operation) from the current entry	O (1)
5	Retrieving the total execution time for the current operation	O (1)
6	Checking if the operation is already in the map	O (1)
7	Adding the operation with an initial value if not present	O (1)
8	Retrieving the count of events for the current operation	O (1)
9	Checking if the count is valid	O (1)
10	Calculating the average execution time for the operation	O (1)
11	Updating the average time for the operation	O (1)
12	Returning the map of average times per operation	O (1)

- "public Map<String, Integer> getWaitTimePerOperation()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*"	O (1)
2	getEventsByItem() method call	O (k)
3	Iterating over items in the "*eventsByItem*" map	O (i)
4	Retrieving the listo f events for the current item	O (1)
5	Iterating over events for the current item	O (n)
6	Retrieving the current event	O (1)
7	Retrieving the previous event	O (1)
8	Calculating the waiting time	O (1)
9	Retrieving the operation from the current event's workstation	O (1)
10	Checking if the operation is already in the map	O (1)
11	Adding the operation with is waiting time if not present	O (1)
12	Start of the else block	O (1)
13	Retrieving the current total waiting time for the operation	O (1)

14	Updating the waiting time for the operation	O (1)
15	End of the inner loop	O (1)
16	Retrieving the first event for the current item	O (1)
17	Getting the start time of the first event	O (1)
18	Retrieving the operation from the first event's workstation	O (1)
19	Checking if the operation is already in the map	O (1)
20	Adding the operation with its initial waiting time if not present	O (1)
21	Start of the else block	O (1)
22	Retrieving the current total waiting time for the operation	O (1)
23	Updating the waiting time for the operation	O (1)
24	End of the if-else statement	O (1)
25	End of the outer loop	O (1)
26	Returning the map of wait times organizated by operation	O (1)

Being "*n*" interactions over the events of an item, then the total complexity is: O (k) + O (i·n)

Since we have O (k) due to the call of the "getEventsByItem()" method, the iteration over items (O (n)) and the interaction over the events for each item (O (i*n)).

• US07:

- "public Map<WorkStation, Map<WorkStation, Integer>> generateDependencies()"

Line of Code	Operation	Complexity
1	Initializing a new "*HashMap*" for workstation dependencies	O (1)
2	"*getEventsByItem()*" method call	O (k)
3	Iterating over items in the "*eventsByItem*" map	O (i)
4	Retrieving the list of events for the current item	O (1)
5	Iterating over events for the current item	O (n)
6	Retrieving the current workstation	O (1)
7	Retrieving the next workstation	O (1)
8	Adding entry for current workstation if it doesn't exist	O (1)
9	Retrieving the dependency map for the current workstation	O (1)
10	Updating the dependency count for the next workstation	O (1)
11	Returning workstation dependencies map	O (1)

- "public Map<WorkStation, Map<WorkStation, Integer>> sortWorkStationFlow()"

Line of Code	Operation	Complexity
5	Initializing a new "*HashMap*" for flow sums	O (1)
19	Iterating over workstation dependencies to calculate flow sums	O (1)
23	Updating the flow sum for the current workstation	O (w)
25	Creating a list from the flow sums for sorting	O (w)
	Sorting the entries based on flow sums	O (w * log(w))
28	Initializing a new "*LinkedHashMap*"	O (1)
30	Iterating over sorted entries	O (w)
31	Retrieving the workstation from the entry	O (1)

32	Adding workstation dependencies to the sorted map	O (1)
36	Returning thge sorted map of workstation dependencies	O (1)

- “public Map<WorkStation, Map<WorkStation, Integer>> generateWorkstationFlow()”

Line of Code	Operation	Complexity
1	Calls the "*generateDependencies*" method and assigns you a new map	O (k + i*n)
2	Calls the "*sortWorkStationFlow*" method for the map and return	O (w * log(w))

This us has complexity O (k+i*n + w*log(w)), since the method that generates the dependencies has complexity (O (k+i*n), with k being the number of events and i*n the iteration of each event over the iteration of each item and the method that orders the map has a complexity of O (w * log(w)) with w being the number of workstations.