Code analysis

Integrative Project – 2024/2025 - G102

Bernardo Cardoso
Francisco Lousada
Helena Mouro
Rui Santiago

Contents

Introduction

The focus of the current document is to analyse and calculate the time complexity of the simulator developed in USEI02 and its subsequent User Stories, as well as their auxiliary methods.

In this analysis, we are considering that all Java natively implemented classes and methods run according to their theoretical time complexity.

Simulator Complexity

Method getMinTime()

Code	**Complexity**
public static int getMinTime() {	O(1)
int minTime = Integer.*MAX_VALUE*;	O(1)
for (Machine machine : *machineTimeRemaining*.keySet()) {	**O(m)**
if (*machineTimeRemaining*.get(machine) < minTime) {	O(1)
minTime = *machineTimeRemaining*.get(machine);}}	O(1)
return minTime;}	O(1)

Table 1 – getMinTime method.

The time complexity of this method is O(m) where m is the number of machines.

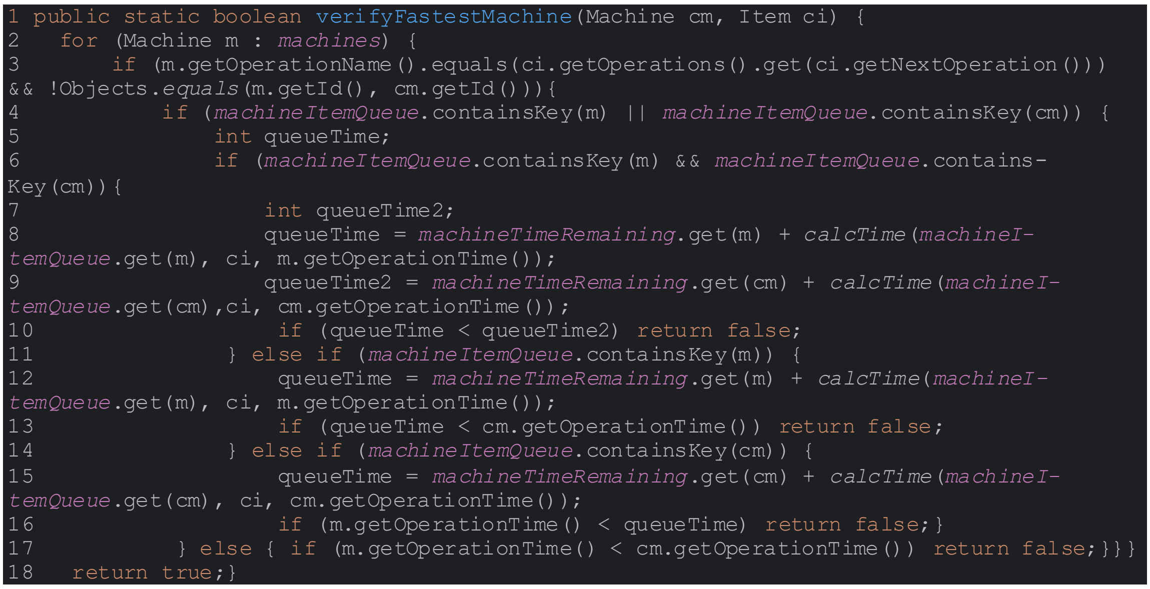
Method calcTime()

Code	**Complexity**
private static int calcTime(Queue<Item> items, Item item, int time) {	O(1)
int queueTime = 0;	O(1)
for (Item i : items) {	**O(i)**
if (pFlag) {	O(1)
if (!checkPriority(i, item)) {	O(1)
queueTime += time; }	O(1)
} else queueTime += time; }	O(1)
return queueTime;}	O(1)

Table 2 – calcTime method.

The time complexity of this method is O(i) where i is the number of items. The method checkPriority() called inside this method is a simple compareTo method with O(1) time complexity.

Method verifyFastestMachine()



```
1 public static boolean verifyFastestMachine(Machine cm, Item ci) {
2   for (Machine m : machines) {
3       if (m.getOperationName().equals(ci.getOperations().get(ci.getNextOperation())) && !Objects.equals(m.getId(), cm.getId())){
4           if (machineItemQueue.containsKey(m) || machineItemQueue.containsKey(cm)) {
5               int queueTime;
6               if (machineItemQueue.containsKey(m) && machineItemQueue.containsKey(cm)){
7                   int queueTime2;
8                   queueTime = machineTimeRemaining.get(m) + calcTime(machineItemQueue.get(m), ci, m.getOperationTime());
9                   queueTime2 = machineTimeRemaining.get(cm) + calcTime(machineItemQueue.get(cm),ci, cm.getOperationTime());
10                   if (queueTime < queueTime2) return false;
11               } else if (machineItemQueue.containsKey(m)) {
12                   queueTime = machineTimeRemaining.get(m) + calcTime(machineItemQueue.get(m), ci, m.getOperationTime());
13                   if (queueTime < cm.getOperationTime()) return false;
14               } else if (machineItemQueue.containsKey(cm)) {
15                   queueTime = machineTimeRemaining.get(cm) + calcTime(machineItemQueue.get(cm), ci, cm.getOperationTime());
16                   if (m.getOperationTime() < queueTime) return false;}
17           } else { if (m.getOperationTime() < cm.getOperationTime()) return false;}}}
18   return true;}
```

Line number	Complexity
1	O(1)
2	O(m)
3	O(1)
4	O(1)
5	O(1)
6	O(1)
7	O(1)
8	O(i)
9	O(i)
10	O(1)
11	O(1)
12	O(i)
13	O(1)
14	O(1)
15	O(i)
16	O(1)
17	O(1)
18	O(1)

Table 3 – verifyFastestMachine method.

The time complexity of this method is O(m * 2i) or simply O(m * i), where is the number of machines and i is the number of items.

Method organizeQueue()

```
1 public static void organizeQueue() {
2    boolean changesInQueue = false;
3    List<Item> unassignedItems = new ArrayList<>();
4    for (Machine machine : machines) {
5        if (machineItemQueue.containsKey(machine) && machineItemQueue.get(machine).size() > 1) {
6            Queue<Item> itemQueue = machineItemQueue.get(machine);
7            List<Item> itemList = new ArrayList<>(itemQueue);
8            int iIndex = (machineTimeRemaining.get(machine) < machine.getOperationTime()) ? 1 : 0;
9            for (int i = iIndex; i < itemList.size(); i++) {
10                for (int j = i; j < itemList.size(); j++) {
11                    if (checkPriority(itemList.get(i), itemList.get(j))) {
12                        ConsoleWriter.displayLog("ITEM " + itemList.get(i).getId() + ANSIColors.paint(" SWAPPED ", ANSIColors.ANSI_RED) + "WITH " + itemList.get(j).getId());
13                        changesInQueue = true;
14                        unassignedItems.addAll(itemList.subList(i, itemList.size()));
15                        Item temp = itemList.get(j);
16                        itemList = itemList.subList(0, i);
17                        itemList.addLast(temp);
18                        unassignedItems.remove(temp);
19                        break;}}
20                if (changesInQueue) break; }
21            itemQueue.clear();
22            itemQueue.addAll(itemList); }}
23    if (changesInQueue) addItemToQueue(unassignedItems); }
```

Line number	Complexity
1	O(1)
2	O(1)
3	O(1)
4	O(m)
5	O(1)
6	O(1)
7	O(1)
8	O(1)
9	O(i)
10	O(i)
11	O(1)
12	O(1)
13	O(1)
14	O(i)
15	O(1)
16	O(1)
17	O(1)
18	O(1)
19	O(1)
20	O(1)
21	O(1)
22	O(i)
23	O(1)

Table 4 – organizeQueue method.

The time complexity of this method is O(m * i^3), where m is the number of machines and i is the number of items.

Method addItemToQueue()

```
1 public static void addItemToQueue(List<Item> items) {
2    boolean assigned = false;
3    for (Item item : items) {
4        for (Machine machine : machines) {
5            if (item.getOperations().get(item.getNextOperation()).equals(machine.getOperationName()) && !assigned) {
6                if (verifyFastestMachine(machine, item)) {
7                    if (machineItemQueue.containsKey(machine)) {
8                        machineItemQueue.get(machine).add(item);
9                        ConsoleWriter.displayLog("ITEM " + item.getId() + ANSIColors.paint(" ASSIGNED ", ANSIColors.ANSI_LIGHT_GREEN) + "TO MACHINE " + machine.getId());
10                        if (pFlag) organizeQueue();
11                    } else {
12                        Queue<Item> itemQueue = new LinkedList<>();
13                        itemQueue.add(item);
14                        machineItemQueue.put(machine, itemQueue);
15                        machineTimeRemaining.put(machine, machine.getOperationTime());
16                        machine.setTimeActive(new Time(simulationTime), new Time(0));
17                        ConsoleWriter.displayLog("ITEM " + item.getId() + ANSIColors.paint(" ASSIGNED ", ANSIColors.ANSI_LIGHT_GREEN) + "TO MACHINE " + machine.getId());}
18                    assigned = true;}}}
19        assigned = false;}}
```

Line number	**Complexity**
1	O(1)
2	O(1)
3	O(i)
4	O(m)
5	O(1)
6	O(m * i)
7	O(1)
8	O(m)
9	O(1)
10	O(m * i^3)
11	O(1)
12	O(1)
13	O(1)
14	O(1)
15	O(1)
16	O(1)
17	O(1)
18	O(1)
19	O(1)

Table 5 – addItemToQueue method.

This method has a time complexity of O(m^4 * i^5), where m is the number of machines and i is the number of items.

Method simulateProduction()

```java
public static void simulateProduction() {
    simulationTime = 0;
    int minTime;
    System.out.println("==================");
    while (!machineItemQueue.isEmpty()) {
        minTime = getMinTime();
        simulationTime += minTime;
        for (Machine m : machineTimeRemaining.keySet()) {
            if (machineItemQueue.containsKey(m)) {
                if (machineTimeRemaining.get(m).toSeconds() - minTime == 0) {
                    machineTimeRemaining.replace(m, new Time(0));
                } else {
                    machineTimeRemaining.replace(m, new Time(machineTimeRemaining.get(m).toSeconds() - minTime));
                }
            }
        }
        Map<Machine, Time> machineTimeCopy = new LinkedHashMap<>(machineTimeRemaining);
        for (Machine machine : machineTimeCopy.keySet()) {
            if (machineTimeRemaining.containsKey(machine) && machineTimeRemaining.get(machine).toSeconds() == 0) {
                if (machineItemQueue.containsKey(machine) && !machineItemQueue.get(machine).isEmpty()) {
                    Queue<Item> itemQueue = machineItemQueue.get(machine);
                    Item item = itemQueue.poll();
                    if (item != null) {
                        item.setProductionTime(item.getProdTime() + machine.getOperationTime().toSeconds());
                        item.addMachineUsed(machine);
                        machine.addItemProcessedTime(new Time (simulationTime-machine.getOperationTime().toSeconds()), item);
                        if (item.getNextOperation() == item.getOperations().size() - 1) {
                            System.out.println(ANSIColors.paint("ITEM " + item.getId() + " FINISHED PRODUCTION", ANSIColors.ANSI_CYAN_BACKGROUND));
                            System.out.println("PRODUCTION TIME: " + new Time(item.getProdTime()) + "s");
                            finishedProducts++;
                        } else {
                            item.setNextOperation(item.getNextOperation() + 1);
                            ConsoleWriter.displayLog("ITEM " + item.getId() + ANSIColors.paint(" MOVED ", ANSIColors.ANSI_LIGHT_YELLOW) + item.getOperations().get(item.getNextOperation()));
                            addItemToQueue(Collections.singletonList(item));
                        }
                        if (machineItemQueue.get(machine).isEmpty()) {
                            machineItemQueue.remove(machine);
                            machineTimeRemaining.remove(machine);
                            machine.setEndTime(new Time(simulationTime));}}}}}
        for (Machine machine : machineTimeRemaining.keySet()) {
            if (machineTimeRemaining.get(machine).toSeconds() == 0) {
                machineTimeRemaining.put(machine, machine.getOperationTime());}}
        for (Machine m : machines) {
            if (machineItemQueue.containsKey(m)) {
                printMachines(m);}}
        System.out.println("==================");
        ConsoleWriter.displayLog("FINISHED " + finishedProducts + " PRODUCTS");}}
```

Line number	**Complexity**
1	O(1)
2	O(1)
3	O(1)
4	O(1)
5	O(i)
6	O(m)
7	O(1)
8	O(m)
9	O(1)
10	O(1)
11	O(1)
12	O(1)
13	O(1)
14	O(1)
15	O(1)
16	O(1)
17	O(1)
18	O(m)
19	O(1)
20	O(1)
21	O(1)
22	O(1)
23	O(1)
24	O(1)
25	O(1)
26	O(1)
27	O(1)
28	O(1)
29	O(1)
30	O(1)
31	O(1)
32	O(1)
33	O(1)
34	O(m^4 * i^5)
35	O(1)
36	O(1)
37	O(1)
38	O(1)
39	O(1)

40	O(m)
41	O(1)
42	O(1)
43	O(m)
44	O(1)
45	O(m)
46	O(1)
47	O(1)

Table 6 – simulateProduction method.

The final calculated time complexity of the simulator is O(m^6 * i^5).

USEI03 Complexity

Code	**Complexity**
public static void printTotalProdTime() {	O(1)
System.out.print("TOTAL PRODUCTION TIME: " + new Time(simulationTime) + "s ("+new Time(simulationTime).toSeconds()+"s).¥n");}	O(1)

Table 7 – USEI03 method.

The USEI03’s method was implemented with a simple print function, so it’s time complexity is O(1).

USEI04 Complexity

Code	**Complexity**
public static void displayExecutionTimesByOperation() {	O(1)
for (String operation : operationTimes.keySet()) {	O(op)
System.out.println(operation + " total execution time: " + operationTimes.get(operation) + "s");}}	O(1)

Table 8 – USEI04 method.

The USEI04’s method time complexity is O(op), where m is the number of operations.

USEI05 Complexity

```
1 public static void printMachineOperatingPerc() {
2    int allProductions = 0;
3    for (Item item : items) {
4        allProductions += item.getProdTime();}
5    machines.sort(Comparator.comparing(Machine::getTimeActiveInInt));
6    for (Machine machine : machines) {
7        float timeInPerc = (float) (machine.getTimeActiveInInt() * 100) / allProductions;
8        float timeOperationInPerc = (float) (machine.getTimeActiveInInt() * 100) / operationTimes.get(machine.getOperationName()).toSeconds();
9        System.out.printf("Machine %s| OPERATING TIME: %ss - %.2f%% of total production time - %.2f%% of total %s time.\n", machine.getId(), new Time(machine.getTimeActiveInInt()), timeInPerc, timeOperationInPerc, machine.getOperationName());}
10    System.out.println("\n================== \n");}
```

Code	Complexity
1	O(1)
2	O(1)
3	O(i)
4	O(1)
5	O(1)
6	O(m)
7	O(1)
8	O(1)
9	O(1)
10	O(1)

Table 9 – USEI05 method.

The USEI05's method time complexity is O(m + i), where m is the number of machines and i is the number of items.

USEI06 Complexity

```
1 public static void calculateAverageTimePerOperation() {
2    HashSet<String> operations = new HashSet<>();
3    for (Machine machine : machines) {
4        operations.add(machine.getOperationName());}
5    Map<String, Float> operationAverages = new LinkedHashMap<>();
6    Map<String, Float> waitingAverages = new LinkedHashMap<>();
7    for (String op : operations) {
8        int i = 0;
9        int j = 0;
10       int sum = 0;
11       int sum2 = 0;
12       for (Machine machine : machines) {
13            if (machine.getOperationName().equals(op) && !machine.getTimeActive().isEmpty()) {
14                i += machine.getActiveTimesLengths().size();
15                j += machine.getWaitingTimeLengths().size();
16                for (Integer t : machine.getActiveTimesLengths()) {
17                    sum += t;}
18                for (Integer t : machine.getWaitingTimeLengths()) {
19                    sum2 += t;}}}
20        if (sum != 0)
21            operationAverages.put(op, (float) sum / i);
22        else
23            operationAverages.put(op, 0.0f);
24        if (sum2 != 0)
25            waitingAverages.put(op, (float) sum2 / j);
26        else
27            waitingAverages.put(op, 0.0f);}
28     for (String s : operationAverages.keySet()) {
29        System.out.printf("Operation " + s + " average time: %.2fs\n", operationAverages.get(s));
30        System.out.printf("Operation " + s + " average waiting time: %.2fs\n", waitingAverages.get(s));}}
```

Code	Complexity
1	O(1)
2	O(1)
3	O(m)
4	O(1)
5	O(1)
6	O(1)
7	O(op)
8	O(1)
9	O(1)
10	O(1)
11	O(1)
12	O(m)
13	O(1)
14	O(1)
15	O(1)
16	O(t)
17	O(1)
18	O(t)

19	O(1)
20	O(1)
21	O(1)
22	O(1)
23	O(1)
24	O(1)
25	O(1)
26	O(1)
27	O(1)
28	O(av)
29	O(1)
30	O(1)

Table 9 – USEI06 method.

The USEI06's method time complexity is O(op * m * t), where op is the number of operations, m is the number of machines, and t is the number of relevant time events.

USEI07 Complexity

```
1 public static void machineDependency() {
2    if (!sFlag){
3        for (Item item : items) {
4            if (!item.getMachinesUsed().isEmpty()) {
5                for (Machine machine : machines) {
6                    if (item.getMachinesUsed().contains(machine)) {
7                        boolean flag = false;
8                        for (int i = 0; i < item.getMachinesUsed().size(); i++) {
9                            if (item.getMachinesUsed().get(i).equals(machine)) {
10                               flag = true;}
11                           if (flag && item.getMachinesUsed().get(i) != machine) {
12                               if (machine.getMachineDependency().get(item.getMachinesUsed().get(i)) != null) {
13                                   machine.setMachineDependency(item.getMachinesUsed().get(i),machine.getMachineDependency().get(item.getMachinesUsed().get(i))+1);} else {
14                                  machine.getMachineDependency().put(item.getMachinesUsed().get(i), 1);}
15                               flag = false;}}}}}}}
16    sFlag = true;
17    Map<Machine, Integer> map = new LinkedHashMap<>();
18    for (Machine machine : machines) {
19        if (!machine.getMachineDependency().isEmpty()) {
20            map.put(machine, sumDependencies(machine));}}
21    StringBuilder sb = new StringBuilder();
22    List<Map.Entry<Machine, Integer>> list = new ArrayList<>(map.entrySet());
23    list.sort(reverseOrder(Map.Entry.comparingByValue()));
24    for (Map.Entry<Machine, Integer> entry : list) {
25        sb.append(entry.getKey().getId()).append(" : [");
26        for (Machine m : entry.getKey().getMachineDependency().keySet()) {
27            sb.append("(").append(m.getId()).append(",").append(entry.getKey().getMachineDependency().get(m)).append(")").append(" , ");}
28        sb = new StringBuilder(sb.substring(0, sb.length() - 3));
29        sb.append("]  (").append(entry.getValue()).append(")\n");}
30    System.out.println(sb);}
```

Code	Complexity
1	O(1)
2	O(1)
3	O(i)
4	O(1)
5	O(m)
6	O(1)
7	O(1)
8	O(m)
9	O(1)
10	O(1)
11	O(1)
12	O(1)
13	O(1)
14	O(1)
15	O(1)
16	O(1)
17	O(1)
18	O(m)
19	O(1)

20	O(m)
21	O(1)
22	O(1)
23	O(m * log(m))
24	O(t)
25	O(1)
26	O(m)
27	O(1)
28	O(1)
29	O(1)
30	O(1)

Table 10 – USEI07 method.

The USEI07's method time complexity is O(i * m^2), where i is the number of items and m is the number of machines.