

|  |  |  |  |
| --- | --- | --- | --- |
| Course Code | : | SWE201 | |
| Course Name | : | Algorithm Analysis and Design | |
| Lecturer | : | Geetha Kanaparan | |
| Academic Session | : | 2021/09 | |
| Assessment Title | : | Quicksort: Theory and Experiments | |
| Submission Due Date | : | 8th December 2021 | |
| Prepared by |  |  | |
| : | Student ID | Student Name |
|  | SWE2009510 | LIU AOFAN |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Date Received | : | 8/12/2021 | |
|  |  |  | |
| Feedback from Lecturer:  Mark: | | | |

**Own Work Declaration**

I/We hereby understand my/our work would be checked for plagiarism or other misconduct, and the softcopy would be saved for future comparison(s).

I/We hereby confirm that all the references or sources of citations have been correctly listed or presented and I/we clearly understand the serious consequence caused by any intentional or unintentional misconduct.

This work is not made on any work of other students (past or present), and it has not been submitted to any other courses or institutions before.

Signature:



Date:

12 Nov., 21

**Table of Contents**

[A Greedy Algorithm for Solving Oranges Arrangement Problem 2](#_Toc89987611)

[1. Pseudocode 2](#_Toc89987612)

[2. Running Time 4](#_Toc89987613)

[2.1 Growth Rate 4](#_Toc89987614)

[2.2 Theoretical Analysis 6](#_Toc89987615)

[2.3 Why It Is Optimal 7](#_Toc89987616)

[2.4 Example of Input Output 9](#_Toc89987617)

[References 12](#_Toc89987618)

[Source Code 13](#_Toc89987619)

[Marking Rubrics 17](#_Toc89987620)

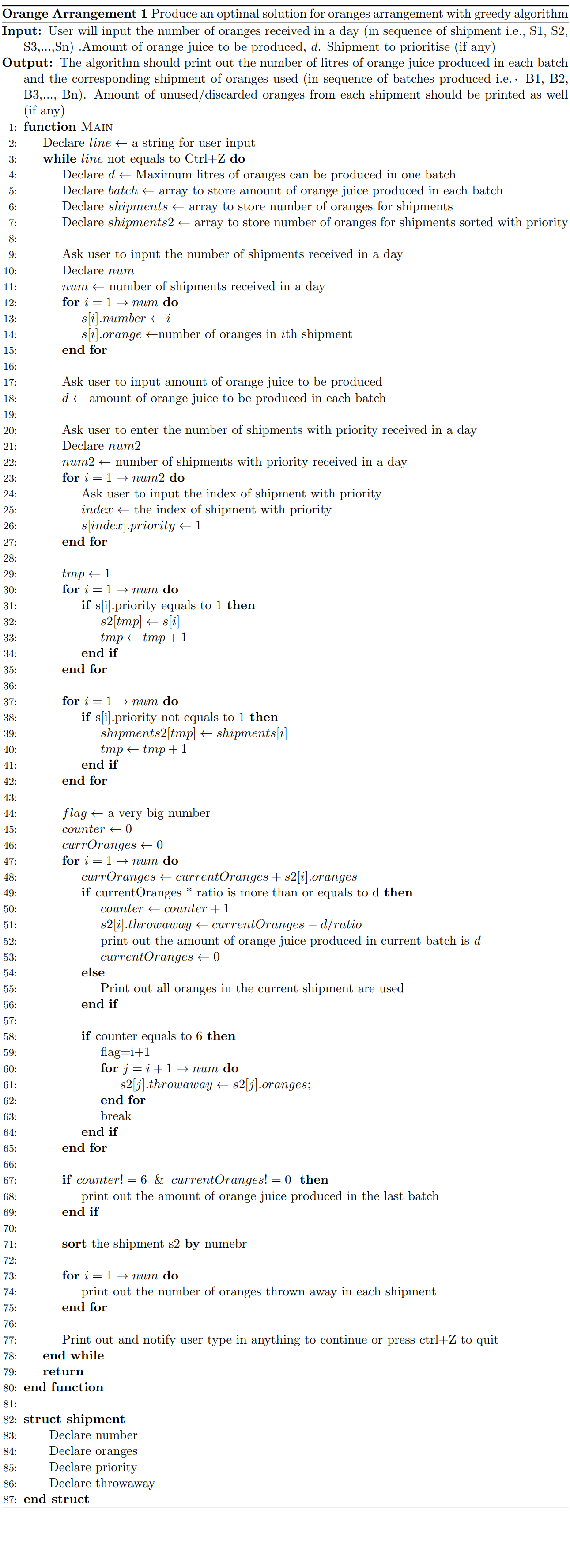
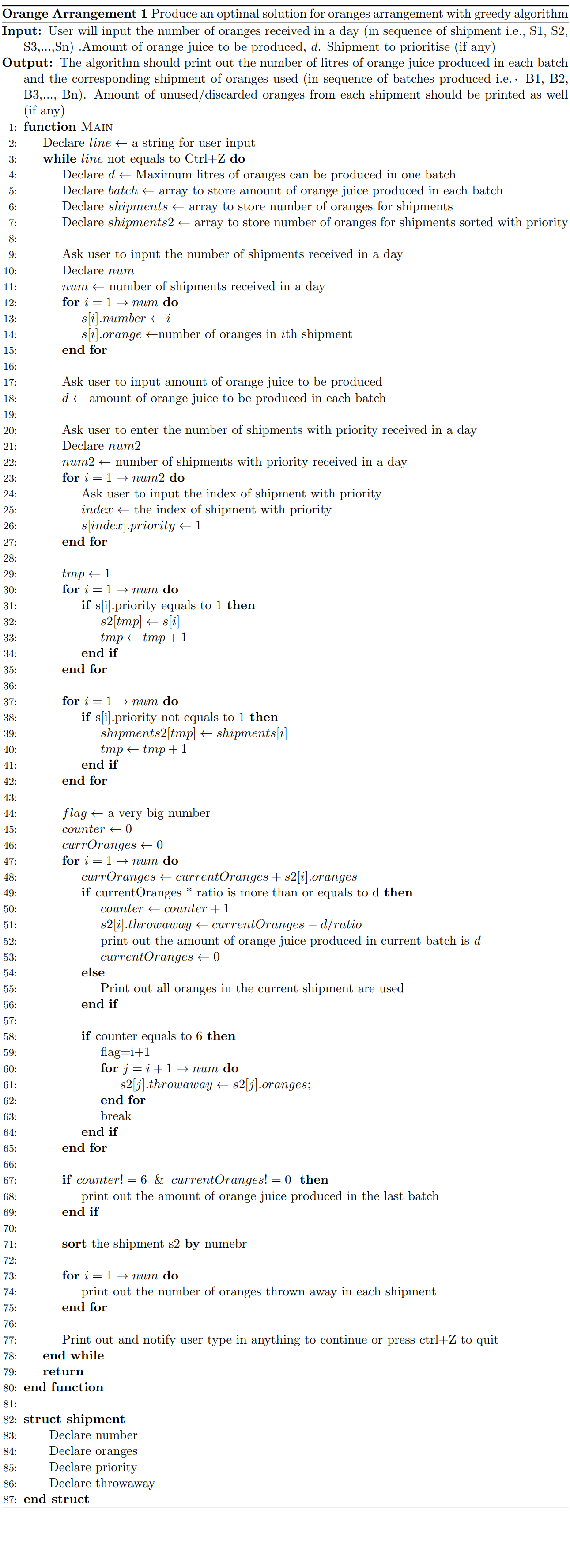
# A Greedy Algorithm for Solving Oranges Arrangement Problem

Name: Liu Aofan - Student ID: SWE2009510

Course: Algorithm Analysis and Design - Profession: Dr.Geetha Kanaparan

## Pseudocode

As a language that describes algorithms between natural language and programming language, pseudocode improves the readability of the code, acts as a bridge between programs and algorithms or flowcharts (Giryes, 2016), and can also be used as a rough document. Here is the pseudocode of our algorithm.



## Running Time

Generally speaking, we have two methods to analyze the growth rate of an algorithm, one is empirical analysis and the other is theoretical analysis. For this algorithm, it is not very suitable to use empirical analysis. So, we consider its growth rate by theoretical analysis (Bang-Jensen, Gutin and Yeo, 2004).

### Growth Rate

The growth rate of the analysis algorithm can help us predict its performance, compare with different algorithms, and tune parameters. Here, we define its running time through line-by-line analysis (Dilworth, Kalton, Kutzarova and Temlyakov, n.d.).

|  |  |  |
| --- | --- | --- |
| Algorithm | Cost | Time |
| int d; | t1 | 1 |
| int batch[7]; | t2 | 1 |
| shipment s[100]; | t3 | 1 |
| shipment s2[100]; | t4 | 1 |
| cout << "Please enter number of shipments received in a day: " << endl; | t5 | 1 |
| int num; | t6 | 1 |
| cin >> num; | t7 | 1 |
| cout << "Please enter the number of oranges received in each shipment: " << endl; | t8 | 1 |
| for (int i = 1; i <= num; i++){ | t9 | n+1 |
| s[i].number = i; | t10 | n |
| cin >> s[i].oranges; | t11 | n |
| } |  | - |
| cout << "Please enter the amount of orange juice to be produced, d: " << endl; | t12 | 1 |
| cin >> d; | t13 | 1 |
| cout << "Please enter the number of shipments with priority received in a day: " << endl; | t14 | 1 |
| int num2; | t15 | 1 |
| cin >> num2; | t16 | 1 |
| if (num2 != 0){ | t17 | 1 |
| cout << "Please enter the index of shipments with priority: " << endl; | t18 | 0-1 |
| for (int i = 1; i <= num2; i++){ | t19 | 0-n+1 |
| int index; | t20 | 0-n |
| cin >> index; | t21 | 0-n |
| s[index].priority = 1; | t22 | 0-n |
| } | - | - |
| } | - | - |
| int tmp = 1; | t23 | 1 |
| for (int i = 1; i <= num; i++){ | t24 | n+1 |
| if (s[i].priority == 1){ | t25 | n |
| s2[tmp] = s[i]; | t26 | 0-n |
| tmp++; | t27 | 0-n |
| } |  | - |
| } |  | - |
| for (int i = 1; i <= num; i++){ | t28 | n+1 |
| if (s[i].priority != 1){ | t29 | n |
| s2[tmp] = s[i]; | t30 | 0-n |
| tmp++; | t31 | 0-n |
| } | - | - |
| } | - | - |
| int flag = 1e9; | t32 | 1 |
| int counter = 0; | t33 | 1 |
| int currentOranges = 0; | t34 | 1 |
| for (int i = 1; i <= num; i++){ | t35 | n+1 |
| currentOranges = currentOranges + s2[i].oranges; | t36 | n |
| if (currentOranges \* ratio >= d){ | t37 | n |
| counter++; | t38 | 0-n |
| s2[i].throwaway = currentOranges - d / ratio; | t39 | 0-n |
| cout << "shipment " << s2[i].number << " uses " << s2[i].oranges - s2[i].throwaway << endl; | t40 | 0-n |
| cout << "batch " << counter << " contains " << d << " litres orange juice" << endl << endl; | t41 | 0-n |
| currentOranges = 0; | t42 | 0-n |
| } | - | - |
| else | t43 | 0-n |
| cout << "shipment " << s2[i].number << " uses " << s2[i].oranges << endl; | t44 | 0-n |
| if (counter == 6){ | t45 | 1 |
| flag = i + 1; | t46 | 1 |
| for (int j = i + 1; j <= num; j++) | t47 | 1 |
| s2[j].throwaway = s2[j].oranges; | t48 | 1 |
| break; | t49 | 1 |
| } | - | - |
| } | - | - |
| if (counter != 6 && currentOranges != 0) | t50 | 1 |
| cout << "batch " << counter + 1 << " contains " << currentOranges \* ratio << " litres orange juice" << endl << endl; | t51 | 1 |
| sort(s2, s2 + num + 1, [](shipment a, shipment b) { return a.number < b.number; }); | t52 | 1 |
| for (int i = 1; i <= num; i++){ | t53 | n+1 |
| cout << "shipment " << i << " throw away " << s2[i].throwaway << " oranges" << endl; | t54 | n |
| } | - | - |
| cout << endl << "Type in anything to continue or press ctrl+Z to quit" << endl; | t55 | 1 |

Minimum total time:

Maximum total time:

### Theoretical Analysis

As the exact performance value of the algorithm, big theta is a useful tool to analyze the algorithm performance. We use big theta to describe the running time of the algorithm.

It is defined as:

:

if and only if:

And we have the following minimum and maximum total time:

* Minimum total time:
* Maximum total time:

**Proof:**

Lower bound:

(i.e., take c = 34 and n0 = 1)

Upper bound:

(i.e., take c = 61 and n0 = 1)

According to the definition,

Thus, this algorithm (n).

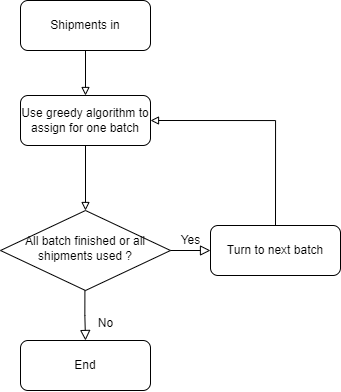
### Why It Is Optimal

We may have heard of greedy algorithms in many places, and in the actual programming process, many problems can be solved by greedy algorithms. This **Orange Arrangement Problem** is an example of greedy algorithms.

Our basic idea is to proceed step by step from a certain **initial batch** of the **Orange Arrangement Problem**. According to a certain optimization measure (i.e., optimize the number of times the production line has to operate in a day), each step must ensure that a **local optimal** solution can be obtained (NIE and JIANG, 2014). In other words, we do not consider the **global optimality of all batches**, and each step we taken for solving this problem is the **best allocation of current batch**.

So, here is the **process flow of our algorithm** for this problem:

* Build a model to describe the problem
* Divide the Orange Arrangement Problem into several Batch allocation problems
* Solve each batch allocation problem and get the **local optimal solution** of the problem
* Combine the local optimal solutions of all batch allocation problems



We are going to illustrate it with one set of input and output.

**Notation convention:**

d ← number of orange juice produced in each batch

ratio← how much liters of orange juice can we get from one orang

s[]← number of oranges carried in each shipment

sp[]← indexes of shipments with priority

**Input 1:**

**d**← 300

**ratio**← 0.1

**s[]**← {2000, 4000, 5000}

**sp[]**←{}

We need 300 liters of orange juice for each batch and we have 3 shipments of oranges, namely, shipment1 with 2000 oranges, shipment2 with 4000 oranges and shipment 3 with 5000 oranges.

In our algorithm, we shall first assign the shipment 1 to the first batch and it cannot satisfy the need of current batch. So, shipment 2 is used for batch 1 as well. Finally, we just need 2 batches for these 3 shipments.

However, we may assign one batch for both shipment 1 and shipment 2. In this way, we have to use three batches for these three shipments. Obviously, 2 is fewer than 3. So, it reflects that our algorithm may better than other algorithms.

But we should also realize that the greedy algorithm is very likely to get a solution that is not the global optimal solution (Jiang et al., 2019). Therefore, we need to **prove** that the **Orange Arrangement Problem satisfies the need of greedy algorithm**.

Here are two **properties** of greedy algorithm:

* **Optimal substructure**

The optimal solution with the optimal sub-structure property usually contains the optimal solution of the sub-problems (Ran, 2017).

* **Greedy choice nature**

When we make a choice, we always make the best choice based on the current situation without considering the solutions of the remaining sub-problems (Shaffer, 2012). If our choice has no aftereffect, then it is very likely that the solutions to our most sub-problems will eventually add up to the global optimal solution.

Let's talk about the **optimal substructures** first. In this algorithm, only one batch is considered at each step. Our algorithm revolves around the idea that the optimal solution of the entire problem must be derived from the optimal solution of the sub-problems existing in the greedy strategy (Vince, 2002).

The **state** is defined as a kind of combination used to indicate the usage of six batches. In this problem, the algorithm has **no aftereffect** (Weiss, 1995), that is, the current state does not affect the subsequent **state**. Therefore, we can follow the Triangle formula:

If the shipment arrangement for each batch does not reach local optima, it is likely that we may use more batches than expected. However, we cannot get better result in global with sacrificing of a part of the performance in current choice (i.e., we have worse choice in current batch) owing to the fact that this **Orange Arrangement Problem** has no aftereffect.

Hence, **the total minimum number of batches used must contain the minimum number of batches selected in each step**. Moreover, the shipment selection for each batch should satisfy the local optimization condition until **all shipments or batches are enumerated** and the algorithm stops.

Next, we are going to focus on the **Greedy choice nature**, to optimize the use of batches as much as possible, we need to **consume shipments as much as possible and maximize the utilization of each batch**. If we do not maximize the utilization of each current batch, the batch used will be greater than or equal to the optimal solution.

Assume the first shipment is marked with and we start finding the optimal solution with shipment until is not equals to 1. So, we can guarantee that all the **shipments previous to k** is satisfy the greedy algorithm. And for shipment , if it can be used in the last batch, we **move it to the last batch**. We replace oranges from the shipment in the current batch with oranges from the shipment .

Since each ship after may either remain unchanged or replace the place of the last shipment, the solution we get is always equal to or better than the current solution. Hence, it satisfies the greedy choice nature.

In case these two **properties** are met, the subsequent process of a certain **batch allocation state** of the current algorithm will not affect the previous state, and is only related to the current state, which means that our **Orange Arrangement Problem meets the principle of need of greedy algorithm.**

### Sets of Input Output

To verify the usability and optimality of our algorithm, we shall test it with various sets of input and output.

Let us consider the extreme situation first. We have only 1 shipment, and there are very few oranges or even no orange on this ship, and it is not enough for the first batch. Then we gradually increase the number of oranges until 6 batches Can be satisfied. Then, oranges were brought over by shipments and gradually increased until they far exceeded the demand, and many of them would be thrown away.

So, we can divide it into three categories:

* **Not enough**
* **Just enough**
* **Exceed**

In addition, we can also classify according to whether there is a priority:

* **No shipment with priority**
* **Few shipments have priority**
* **Almost all shipments have priority**

Finally, we got 3 plus 3 is equal to nine categories.

1. **Not enough | No shipment with priority**
2. **Not enough | Few shipments have priority**
3. **Not enough | Almost all shipments have priority**
4. **Just enough | No shipment with priority**
5. **Just enough | Few shipments have priority**
6. **Just enough | Almost all shipments have priority**
7. **Exceed | No shipment with priority**
8. **Exceed | Few shipments have priority**
9. **Exceed | Almost all shipments have priority**

For each of these three categories, we can have multiple sets of inputs and outputs. Another possible algorithm would be to assign each ship's oranges to a batch, with which we might compare the greedy algorithm when we are analyzing them (Geiger and Böcherer, 2016). Moreover, we define ratio to be 0.1 which means we can get 1 litre of orange juice from 10 oranges.

1. **Not enough, No shipment with priority**

**Scenario:** In this situation, we only have few shipments of ranges and we don’t have any shipment with priority. we even cannot satisfy the need of the first batch.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

qwe

Please enter number of shipments received in a day:

1

Please enter the number of oranges received in each shipment:

2000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

0

**Output 1:**

shipment 1 uses 2000

batch 1 contains 200 litres orange juice

shipment 1 throw away 0 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:**  In this set of input and output, we only have one set of data, so the performance of the greedy algorithm is the same as other algorithms.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

df

Please enter number of shipments received in a day:

1

Please enter the number of oranges received in each shipment:

3500

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

0

**Output 2:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 1 throw away 500 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** Similarly, we still only have one set of data, slightly different from the last set of data, we have more than one batch of data used. Our greedy algorithm will still end up in the same way with the other algorithms.

1. **Not enough | Few shipments have priority**

**Scenario:** We still don’t have enough oranges for the need of all batches, but some of the shipments are with priority

**Input1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

qw

Please enter number of shipments received in a day:

3

Please enter the number of oranges received in each shipment:

2000 4000 3000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

1

Please enter the index of shipments with priority:

3

**Output1:**

shipment 3 uses 3000

batch 1 contains 300 litres orange juice

shipment 1 uses 2000

shipment 2 uses 1000

batch 2 contains 300 litres orange juice

shipment 1 throw away 0 oranges

shipment 2 throw away 3000 oranges

shipment 3 throw away 0 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** 2000, 4000, 300 are the number of our oranges, but after sorting by priority, we get 3000, 2000, 4000. If we use the greedy algorithm 2000 and 4000 will be allocated to the same batch, so that only 2 batches are used, the least result of our sequential processing is to use two batches according to the conditions of the problem in case we can only process them in sequence. However, if we allocate 2000 and 4000 to two batches, we will have to use 3 batches, but this is not the optimal solution.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

er

Please enter number of shipments received in a day:

2

Please enter the number of oranges received in each shipment:

3400 2000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

1

Please enter the index of shipments with priority:

2

**Output 2:**

shipment 2 uses 2000

shipment 1 uses 1000

batch 1 contains 300 litres orange juice

shipment 1 throw away 2400 oranges

shipment 2 throw away 0 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** In this example, the order of processing after our shipment is prioritized is 2000, 3400. Obviously, if we deal with them, 1 batch is the least amount of usage. In other algorithms, we may assign each shipment to one batch, using more than or equal to one batch. In this case, the greedy algorithm is still optimal.

1. **Not enough | Almost all shipments have priority**

**Scenario:** We still don’t have enough shipments of oranges but almost all shipments of oranges have priority in this case.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

q

Please enter number of shipments received in a day:

5

Please enter the number of oranges received in each shipment:

2000 4000 3000 5000 7000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

4

Please enter the index of shipments with priority:

2 3 4 5

**Output 1:**

shipment 2 uses 3000

batch 1 contains 300 litres orange juice

shipment 3 uses 3000

batch 2 contains 300 litres orange juice

shipment 4 uses 3000

batch 3 contains 300 litres orange juice

shipment 5 uses 3000

batch 4 contains 300 litres orange juice

shipment 1 uses 2000

batch 5 contains 200 litres orange juice

shipment 1 throw away 0 oranges

shipment 2 throw away 1000 oranges

shipment 3 throw away 0 oranges

shipment 4 throw away 2000 oranges

shipment 5 throw away 4000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** After sorting, we get 4000, 3000, 5000, 7000, 2000. In this case, only each shipment can be allocated to each batch. The greedy algorithm is similar to other algorithms.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

k

Please enter number of shipments received in a day:

2

Please enter the number of oranges received in each shipment:

3400 2000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

2

Please enter the index of shipments with priority:

1 2

**Output 2:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 2 uses 2000

batch 2 contains 200 litres orange juice

shipment 1 throw away 400 oranges

shipment 2 throw away 0 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** This input is a bit special. All ships we get have priority, so we need to sort them according to the normal order. At the same time, we need at least two batches to process these oranges, because the first batch is already full. The solution obtained by greedy algorithm is the same as other algorithms.

1. **Just enough | No shipment with priority**

**Scenario:** In this case, the number of oranges carried on the ship is just enough for our 6 batches. At the same time, none of the ships has priority.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

a

Please enter number of shipments received in a day:

6

Please enter the number of oranges received in each shipment:

4000 3000 5000 6000 3000 4000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

0

**Output 1:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 2 uses 3000

batch 2 contains 300 litres orange juice

shipment 3 uses 3000

batch 3 contains 300 litres orange juice

shipment 4 uses 3000

batch 4 contains 300 litres orange juice

shipment 5 uses 3000

batch 5 contains 300 litres orange juice

shipment 6 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 1000 oranges

shipment 2 throw away 0 oranges

shipment 3 throw away 2000 oranges

shipment 4 throw away 3000 oranges

shipment 5 throw away 0 oranges

shipment 6 throw away 1000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** From this perspective, all of our oranges exceed the batch processing capacity, so the situation obtained by the greedy algorithm will be similar to other algorithms, but they are all optimal solutions.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

asd

Please enter number of shipments received in a day:

7

Please enter the number of oranges received in each shipment:

2400 1500 3400 3500 4000 5000 7000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

0

**Output 2:**

shipment 1 uses 2400

shipment 2 uses 600

batch 1 contains 300 litres orange juice

shipment 3 uses 3000

batch 2 contains 300 litres orange juice

shipment 4 uses 3000

batch 3 contains 300 litres orange juice

shipment 5 uses 3000

batch 4 contains 300 litres orange juice

shipment 6 uses 3000

batch 5 contains 300 litres orange juice

shipment 7 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 0 oranges

shipment 2 throw away 900 oranges

shipment 3 throw away 400 oranges

shipment 4 throw away 500 oranges

shipment 5 throw away 1000 oranges

shipment 6 throw away 2000 oranges

shipment 7 throw away 4000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** The number of oranges carried by our first and second ships is not enough for batch usage, which is the key point of this input. Therefore, the number of oranges carried by other ships will far exceed Batch's requirement. If other algorithms are used, we may allocate these two ships into two batches at most and one batch at least. When we used the greedy algorithm, we used just one batch. So greedy works in this case.

1. **Just enough | Few shipments have priority**

**Scenario:** In the fifth case, we have just enough oranges, and some ships have priority. The number of oranges on these priority ships will have an important effect on our batch usage.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

sPlease enter number of shipments received in a day:

6

Please enter the number of oranges received in each shipment:

4000 3000 5000 6000 3000 4000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

2

Please enter the index of shipments with priority:

2 4

**Output 1:**

shipment 2 uses 3000

batch 1 contains 300 litres orange juice

shipment 4 uses 3000

batch 2 contains 300 litres orange juice

shipment 1 uses 3000

batch 3 contains 300 litres orange juice

shipment 3 uses 3000

batch 4 contains 300 litres orange juice

shipment 5 uses 3000

batch 5 contains 300 litres orange juice

shipment 6 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 1000 oranges

shipment 2 throw away 0 oranges

shipment 3 throw away 2000 oranges

shipment 4 throw away 3000 oranges

shipment 5 throw away 0 oranges

shipment 6 throw away 1000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** In this case, since all the ships carried more oranges than the batch required, our adjustment of the order did not have much impact on the final batch, so the greedy was optimal in a similar way to the other algorithms.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

wqePlease enter number of shipments received in a day:

7

Please enter the number of oranges received in each shipment:

2400 1500 3400 3500 4000 5000 7000

Please enter the amount of orange juice to be produced, d: 300

Please enter the number of shipments with priority received in a day:

2

Please enter the index of shipments with priority:

1 6

**Output 2:**

shipment 1 uses 2400

shipment 6 uses 600

batch 1 contains 300 litres orange juice

shipment 2 uses 1500

shipment 3 uses 1500

batch 2 contains 300 litres orange juice

shipment 4 uses 3000

batch 3 contains 300 litres orange juice

shipment 5 uses 3000

batch 4 contains 300 litres orange juice

shipment 7 uses 3000

batch 5 contains 300 litres orange juice

shipment 1 throw away 0 oranges

shipment 2 throw away 0 oranges

shipment 3 throw away 1900 oranges

shipment 4 throw away 500 oranges

shipment 5 throw away 1000 oranges

shipment 6 throw away 4400 oranges

shipment 7 throw away 4000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** At this time, the value of the greedy algorithm is reflected, because after sorting by priority, the 2400 and 1500 data that are less than the batch demand need to be supplemented by the oranges of the next ship, so we only need 5 batches. When using greedy algorithm, other algorithms may have 7 batches.

1. **Just enough | Almost all shipments have priority**

**Scenario:** In this case, almost all ships have priority except a few of them. The number of ships supplied is just enough for the batch demand.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

sPlease enter number of shipments received in a day:

6

Please enter the number of oranges received in each shipment:

4000 3000 5000 6000 3000 4000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

5

Please enter the index of shipments with priority:

1 2 3 5 6

**Output 1:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 2 uses 3000

batch 2 contains 300 litres orange juice

shipment 3 uses 3000

batch 3 contains 300 litres orange juice

shipment 5 uses 3000

batch 4 contains 300 litres orange juice

shipment 6 uses 3000

batch 5 contains 300 litres orange juice

shipment 4 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 1000 oranges

shipment 2 throw away 0 oranges

shipment 3 throw away 2000 oranges

shipment 4 throw away 3000 oranges

shipment 5 throw away 0 oranges

shipment 6 throw away 1000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** In this case, since all the ships carried more oranges than the batch required, our adjustment of the order did not have much impact on the final batch, so the greedy was optimal in a similar way to the other algorithms.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

we

Please enter number of shipments received in a day:

7

Please enter the number of oranges received in each shipment:

2400 1500 3400 3500 4000 5000 7000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

6

Please enter the index of shipments with priority:

1 2 3 4 5 7

**Output 2:**

                tmp++;

shipment 1 uses 2400

shipment 2 uses 600

batch 1 contains 300 litres orange juice

shipment 3 uses 3000

batch 2 contains 300 litres orange juice

shipment 4 uses 3000

batch 3 contains 300 litres orange juice

shipment 5 uses 3000

batch 4 contains 300 litres orange juice

shipment 7 uses 3000

batch 5 contains 300 litres orange juice

shipment 6 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 0 oranges

shipment 2 throw away 900 oranges

shipment 3 throw away 400 oranges

shipment 4 throw away 500 oranges

shipment 5 throw away 1000 oranges

shipment 6 throw away 2000 oranges

shipment 7 throw away 4000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** In this case, as 1, 2, 3, 4, 5 and 7 are all prematurely assigned, only six batches are put to the end. However, we notice that both 7 and 6 are over the batch number of batches, yet we still need six batches. As analysis in the above, other algorithms may require seven batches

1. **Exceed | No shipment with priority**

**Scenario:** Here, we consider that the number of oranges far exceeds the demand, none of them have priority, we only need to process them in order.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

g

lease enter number of shipments received in a day:

8

Please enter the number of oranges received in each shipment:

4000 3000 2000 4000 3000 5000 6000 8000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

0

**Output 1:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 2 uses 3000

batch 2 contains 300 litres orange juice

shipment 3 uses 2000

shipment 4 uses 1000

batch 3 contains 300 litres orange juice

shipment 5 uses 3000

batch 4 contains 300 litres orange juice

shipment 6 uses 3000

batch 5 contains 300 litres orange juice

shipment 7 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 1000 oranges

shipment 2 throw away 0 oranges

shipment 3 throw away 0 oranges

shipment 4 throw away 3000 oranges

shipment 5 throw away 0 oranges

shipment 6 throw away 2000 oranges

shipment 7 throw away 3000 oranges

shipment 8 throw away 8000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** In this input, shipments far exceed the demand of batches, so the greedy algorithm behaves the same as the other algorithms. Note, however, that the implementation details are different: the greedy algorithm merges 2000 and 4000, while other algorithms may assign two batches to them.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

eqw

Please enter number of shipments received in a day:

7

Please enter the number of oranges received in each shipment:

3500 8000 4000 5000 6000 3400 5000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

0

**Output 2:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 2 uses 3000

batch 2 contains 300 litres orange juice

shipment 3 uses 3000

batch 3 contains 300 litres orange juice

shipment 4 uses 3000

batch 4 contains 300 litres orange juice

shipment 5 uses 3000

batch 5 contains 300 litres orange juice

shipment 6 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 500 oranges

shipment 2 throw away 5000 oranges

shipment 3 throw away 1000 oranges

shipment 4 throw away 2000 oranges

shipment 5 throw away 3000 oranges

shipment 6 throw away 400 oranges

shipment 7 throw away 5000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** The situation is a bit similar. The greedy algorithm behaves similarly to other algorithms when shipments exceed batch's demand, especially when we are dealing with input that similar to this.

1. **Exceed | Few shipments have priority**

**Scenario:** Now it comes to the case when few shipments have priority although the number of oranges is bigger than need.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

s

Please enter number of shipments received in a day:

8

Please enter the number of oranges received in each shipment:

4000 3000 2000 4000 3000 5000 6000 8000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

2

Please enter the index of shipments with priority:

1 7

**Output 1:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 7 uses 3000

batch 2 contains 300 litres orange juice

shipment 2 uses 3000

batch 3 contains 300 litres orange juice

shipment 3 uses 2000

shipment 4 uses 1000

batch 4 contains 300 litres orange juice

shipment 5 uses 3000

batch 5 contains 300 litres orange juice

shipment 6 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 1000 oranges

shipment 2 throw away 0 oranges

shipment 3 throw away 0 oranges

shipment 4 throw away 3000 oranges

shipment 5 throw away 0 oranges

shipment 6 throw away 2000 oranges

shipment 7 throw away 3000 oranges

shipment 8 throw away 8000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** In this example, most of the shipments exceed the batch demand, except that the third shipment is put together with the fourth shipment in the greedy algorithm. Therefore, the greedy algorithm will still be the optimal solution.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

qwe

Please enter number of shipments received in a day:

7

Please enter the number of oranges received in each shipment:

3500 8000 4000 5000 6000 3400 5000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

2

Please enter the index of shipments with priority:

2 4

**Output 2:**

shipment 2 uses 3000

batch 1 contains 300 litres orange juice

shipment 4 uses 3000

batch 2 contains 300 litres orange juice

shipment 1 uses 3000

batch 3 contains 300 litres orange juice

shipment 3 uses 3000

batch 4 contains 300 litres orange juice

shipment 5 uses 3000

batch 5 contains 300 litres orange juice

shipment 6 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 500 oranges

shipment 2 throw away 5000 oranges

shipment 3 throw away 1000 oranges

shipment 4 throw away 2000 oranges

shipment 5 throw away 3000 oranges

shipment 6 throw away 400 oranges

shipment 7 throw away 5000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** The greedy algorithm still behaves similarly to the other algorithms in this case, as each shipment exceeds the number of oranges batch can handle.

1. **Exceed | Almost all shipments have priority**

**Scenario:** Things are the same. In this case, our number of oranges far exceeds what is needed. In addition, almost most ships have priority, and only a small number of ships do not have priority.

**Input 1:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

k

Please enter number of shipments received in a day:

8

Please enter the number of oranges received in each shipment:

4000 3000 2000 4000 3000 5000 6000 8000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

7

Please enter the index of shipments with priority:

1 2 3 4 5 6 8

**Output 1:**

shipment 1 uses 3000

batch 1 contains 300 litres orange juice

shipment 2 uses 3000

batch 2 contains 300 litres orange juice

shipment 3 uses 2000

shipment 4 uses 1000

batch 3 contains 300 litres orange juice

shipment 5 uses 3000

batch 4 contains 300 litres orange juice

shipment 6 uses 3000

batch 5 contains 300 litres orange juice

shipment 8 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 1000 oranges

shipment 2 throw away 0 oranges

shipment 3 throw away 0 oranges

shipment 4 throw away 3000 oranges

shipment 5 throw away 0 oranges

shipment 6 throw away 2000 oranges

shipment 7 throw away 6000 oranges

shipment 8 throw away 5000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** Back to the data, only the seventh batch is placed behind, and the 2000 and 4000 are combined. The greedy algorithm uses 7 shipments for 6 batches. The other algorithms used 6 shipments, although the results were similar.

**Input 2:**

This program will always loop for new n values

Type in anything to continue or press ctrl+Z to quit:

qwe

Please enter number of shipments received in a day:

7

Please enter the number of oranges received in each shipment:

3500 8000 4000 5000 6000 3400 5000

Please enter the amount of orange juice to be produced, d:

300

Please enter the number of shipments with priority received in a day:

5

Please enter the index of shipments with priority:

3 4 5 6 7

**Output 2:**

shipment 3 uses 3000

batch 1 contains 300 litres orange juice

shipment 4 uses 3000

batch 2 contains 300 litres orange juice

shipment 5 uses 3000

batch 3 contains 300 litres orange juice

shipment 6 uses 3000

batch 4 contains 300 litres orange juice

shipment 7 uses 3000

batch 5 contains 300 litres orange juice

shipment 1 uses 3000

batch 6 contains 300 litres orange juice

shipment 1 throw away 500 oranges

shipment 2 throw away 8000 oranges

shipment 3 throw away 1000 oranges

shipment 4 throw away 2000 oranges

shipment 5 throw away 3000 oranges

shipment 6 throw away 400 oranges

shipment 7 throw away 2000 oranges

Type in anything to continue or press ctrl+Z to quit

^Z

**Analysis:** Let us observe the data itself. The seven ships carried 3500 8000 4000 5000 6000 3400 5000 oranges. After sorting, they were 4000 5000 6000 3400 5000 3500 8000. Our ratio is defined as 0.1, and d=300, each of the above data exceeds 300, so the performance of the greedy algorithm and other algorithms are similar, but they are both optimal solutions.

**References**

Bang-Jensen, J., Gutin, G. and Yeo, A., 2004. When the greedy algorithm fails. *Discrete Optimization*, 1(2), pp.121-127.

Dilworth, S., Kalton, N., Kutzarova, D. and Temlyakov, V., n.d. *The Thresholding Greedy Algorithm, Greedy Bases, and Duality*.

Geiger, B. and Böcherer, G., 2016. Greedy Algorithms for Optimal Distribution Approximation. *Entropy*, 18(7), p.262.

Giryes, R., 2016. A greedy algorithm for the analysis transform domain. *Neurocomputing*, 173, pp.278-289.

*International Journal of Recent Trends in Engineering and Research*, 2018. GREEDY-KNAPSACK ALGORITHM FOR OPTIMAL DOWNLINK RESOURCE ALLOCATION IN LTE NETWORKS. pp.142-150.

Jiang, Z., Sahasrabudhe, V., Mohamed, A., Grebel, H. and Rojas-Cessa, R., 2019. Greedy Algorithm for Minimizing the Cost of Routing Power on a Digital Microgrid. *Energies*, 12(16), p.3076.

Jiang, Z., Sahasrabudhe, V., Mohamed, A., Grebel, H. and Rojas-Cessa, R., 2019. Greedy Algorithm for Minimizing the Cost of Routing Power on a Digital Microgrid. *Energies*, 12(16), p.3076.

Jiang, Z., Sahasrabudhe, V., Mohamed, A., Grebel, H. and Rojas-Cessa, R., 2019. Greedy Algorithm for Minimizing the Cost of Routing Power on a Digital Microgrid. *Energies*, 12(16), p.3076.

NIE, C. and JIANG, J., 2014. Optimization of Configurable Greedy Algorithm for Covering Arrays Generation. *Journal of Software*, 24(7), pp.1469-1483.

Ran, J., 2017. Maximization Influence Flow Using Modified Greedy Algorithm. *International Journal Of Engineering And Computer Science*,.

Ran, J., 2017. Maximization Influence Flow Using Modified Greedy Algorithm. *International Journal Of Engineering And Computer Science*,.

Shaffer, C., 2012. *Data Structures and Algorithm Analysis in C++*. Dover Publications.

Shaffer, C., n.d. *Data structures and algorithm analysis in C++*.

Vince, A., 2002. A framework for the greedy algorithm. *Discrete Applied Mathematics*, 121(1-3), pp.247-260.

Weiss, M., 1995. *Data structures and algorithm analysis*. Redwood City (Ca.) etc.!: The Benjamin/Cummings.

**Appendix1**

**Source Code**

#include <bits/stdc++.h>

#define ratio 0.1

using *namespace* std;

// here i assume the ratio 10 oranges to 1 liter of juice

*struct* shipment

{

    // all information with regard to the shipment of oranges

*int* number = 0;

*int* oranges = 0;

*int* priority = 0;

*int* throwaway = 0;

};

*int* main()

{

    string line;

    cout << "This program will always loop for new n values" << endl

         << "Type in anything to continue or press ctrl+Z to quit: " << endl;

    while (cin >> line)

    {

        //initialize variables

        //struct s for input all shipments

        //struct s2 for shipments sorted by priority

*int* d;

*int* batch[7];     //start from 1, so batch[0] is not used

        shipment s[100];  //array of structs

        shipment s2[100]; //array of structs with priority

        //input amount of oranges in each shipment

        cout << "Please enter number of shipments received in a day: " << endl;

*int* num;

        cin >> num;

        cout << "Please enter the number of oranges received in each shipment: " << endl;

        for (*int* i = 1; i <= num; i++)

        {

            //variable number to remember the shipment number

            s[i].number = i;

            cin >> s[i].oranges;

        }

        //input amoung of orange juice in each batch

        cout << "Please enter the amount of orange juice to be produced, d: " << endl;

        cin >> d;

        //input each shipment with priority

        cout << "Please enter the number of shipments with priority received in a day: " << endl;

*int* num2;

        cin >> num2;

        if (num2 != 0)

        {

            cout << "Please enter the index of shipments with priority: " << endl;

            for (*int* i = 1; i <= num2; i++)

            {

*int* index;

                cin >> index;

                s[index].priority = 1;

            }

        }

        //sort the shipments by priority

        //if the shipment has priority, it will be added to the array s2 first

*int* tmp = 1;

        for (*int* i = 1; i <= num; i++)

        {

            if (s[i].priority == 1)

            {

                s2[tmp] = s[i];

                tmp++;

            }

        }

        //if the shipment has no priority, it will be added to the array s2 later

        for (*int* i = 1; i <= num; i++)

        {

            if (s[i].priority != 1)

            {

                s2[tmp] = s[i];

                tmp++;

            }

        }

*int* flag = 1e9;  //flag to mark the last shipment used, got the idea of flag usage from https://www.geeksforgeeks.org/use-of-flag-in-programming/

*int* counter = 0; //counter for counting the number of batches used

*int* currentOranges = 0;

        for (*int* i = 1; i <= num; i++)

        {

            //add the oranges of the shipment to the current oranges in the batch (total oranges for now)

            currentOranges = currentOranges + s2[i].oranges;

            //until, we can satisfy the need of current batch

            //then we change to next batch

            if (currentOranges \* ratio >= d)

            { //go to the new batch

                counter++;

                s2[i].throwaway = currentOranges - d / ratio;

                //if there are orange left, we need to throw away some oranges

                //we only need to throw away the oranges in the last shipment for this batch

                //all of the oranges in the privious shipments are already used

                cout << "shipment " << s2[i].number << " uses " << s2[i].oranges - s2[i].throwaway << endl;

                cout << "batch " << counter << " contains " << d << " litres orange juice" << endl

                     << endl;

                currentOranges = 0;

            }

            else

                cout << "shipment " << s2[i].number << " uses " << s2[i].oranges << endl;

            //break when comes to the end of the batch

            if (counter == 6)

            {

                flag = i + 1;

                for (*int* j = i + 1; j <= num; j++)

                    s2[j].throwaway = s2[j].oranges;

                break;

            }

        }

        //print the leftover oranges

        if (counter != 6 && currentOranges != 0)

            cout << "batch " << counter + 1 << " contains " << currentOranges \* ratio << " litres orange juice" << endl

                 << endl;

        //because in s2 shipments with priority are added first

        //struct s2 is not sorted by number

        //so, we sort the shipments by number again

        //thanks to (Giryes, R., 2016. A greedy algorithm for the analysis transform domain) for let me know [](shipment a, shipment b){ return a.number < b.number; }

        sort(s2, s2 + num + 1, [](shipment *a*, shipment *b*)

             { return a.number < b.number; });

        //print the number of oranges thrown away

        for (*int* i = 1; i <= num; i++)

        {

            cout << "shipment " << i << " throw away " << s2[i].throwaway << " oranges" << endl;

        }

        //loop for new value

        //I got the idea of cin >> line and while statement

        //from Shaffer, C., n.d. Data structures and algorithm analysis in C++.

        cout << endl

             << "Type in anything to continue or press ctrl+Z to quit" << endl;

    }

    return 0;

}

**Appendix2**

**Marking Rubrics**

