The Algorithm Animation Project

THE DESIGN PRESENTATION

CONDUCTED BY NUR AISHAH B M SENIN

A brief description about the program

Show the different types of commonly used algorithms

From the 3 main algorithmic paradigms and sorting algorithms

Animations to show the essential pieces of the puzzle that makes the algorithm works as a whole

Enables user to “control” the animation

Program is run on a WPF application, which predominantly runs on Windows operating systems (from Windows 7 onwards)

The purpose of the *Algorithm Animation Program*

Provide a better understanding on how algorithms work in general

For the benefit of users by allowing them to learn algorithms from a different perspective

An extra educational content that can be used aside from lectures

Target audience: Computer Science students or individuals who are interested in finding ways of making a program more efficient

Aims and objectives of this project

To understand the difficulties that are faced by students when it comes to this topic, and to overcome them

Breaking down difficult algorithmic problems into simpler forms

Allow greater understanding from our users

Serve this program as a foundation for developers alike

Design of the software

The contents of the program

A main menu

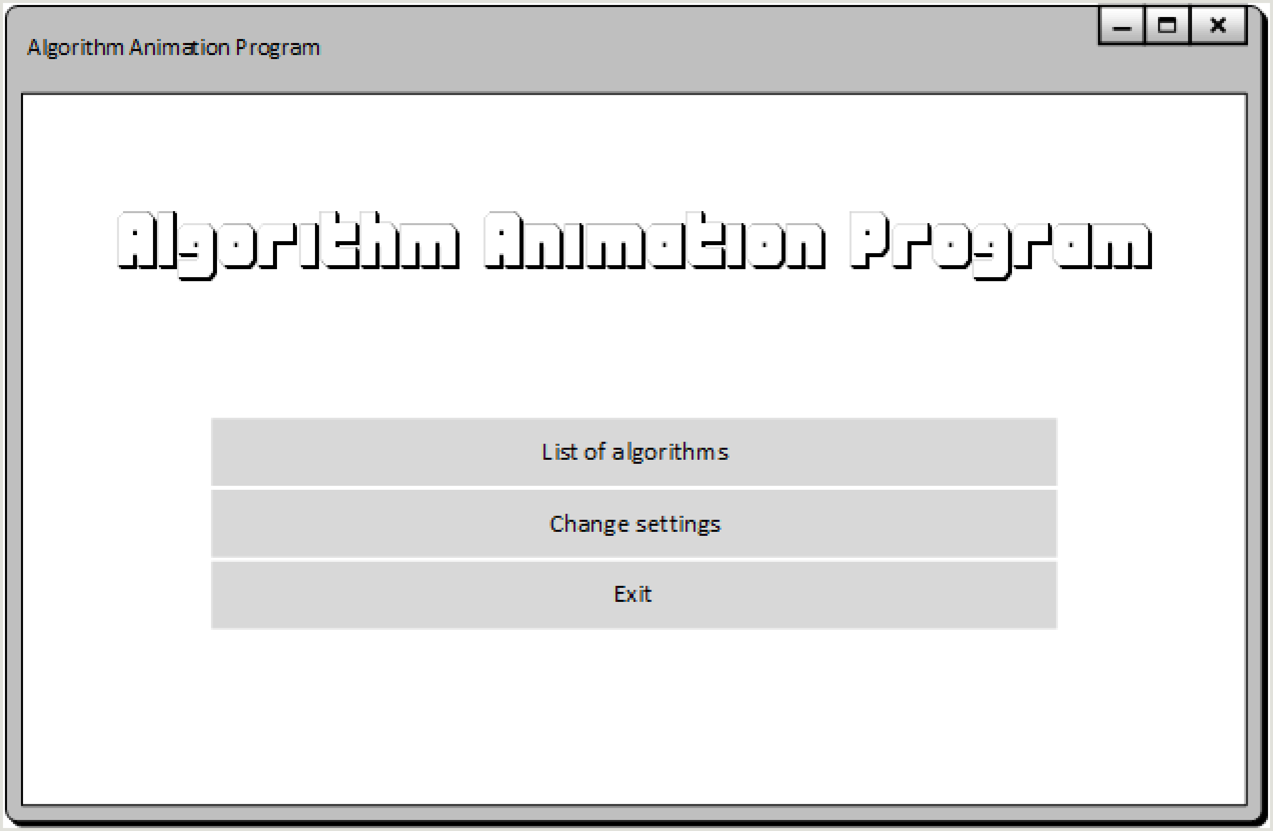
Settings feature

Animation feature

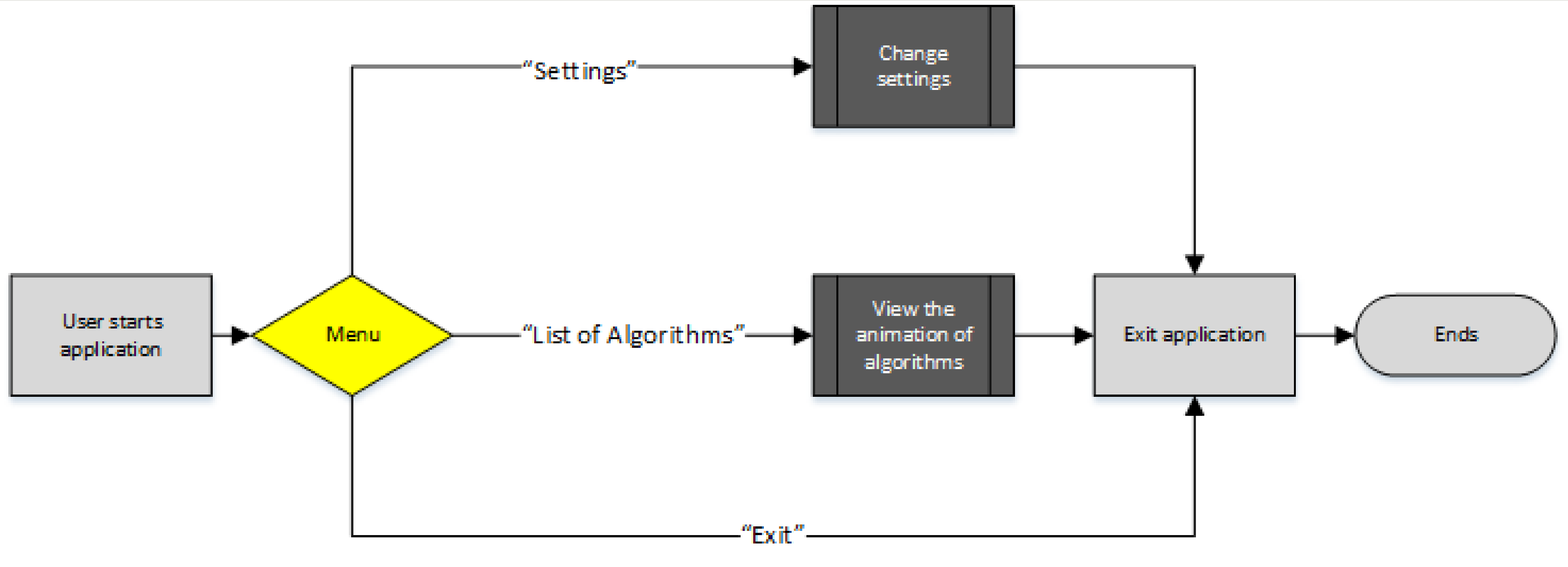
Main menu

To provide user navigation

Lists the selection of features for the user to select from



The simple UI design of the main menu



The flowchart that shows the overall view of the *Algorithm Animation Program* from the Main Menu

The settings feature

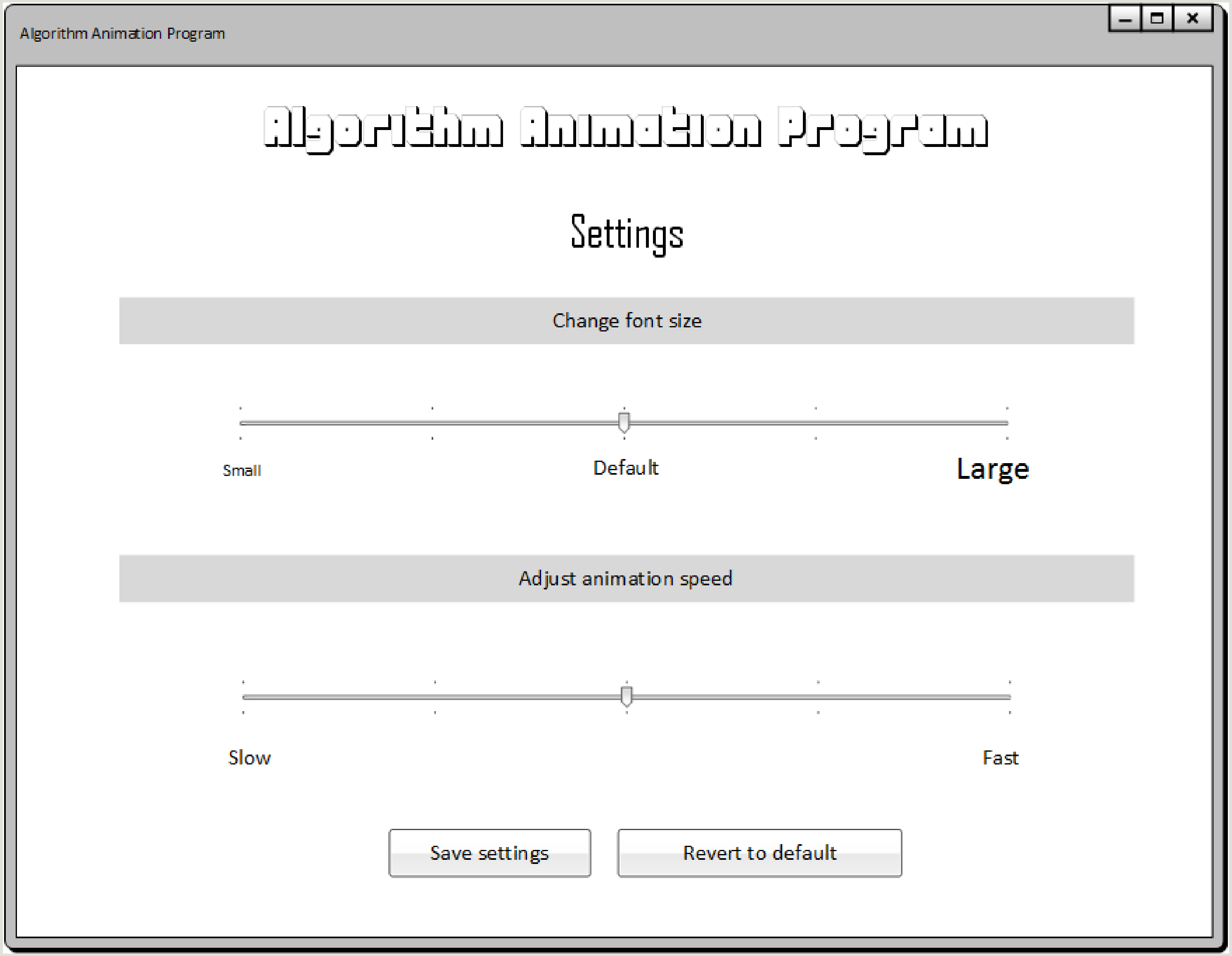
Allow users to change the settings

Changing the font size

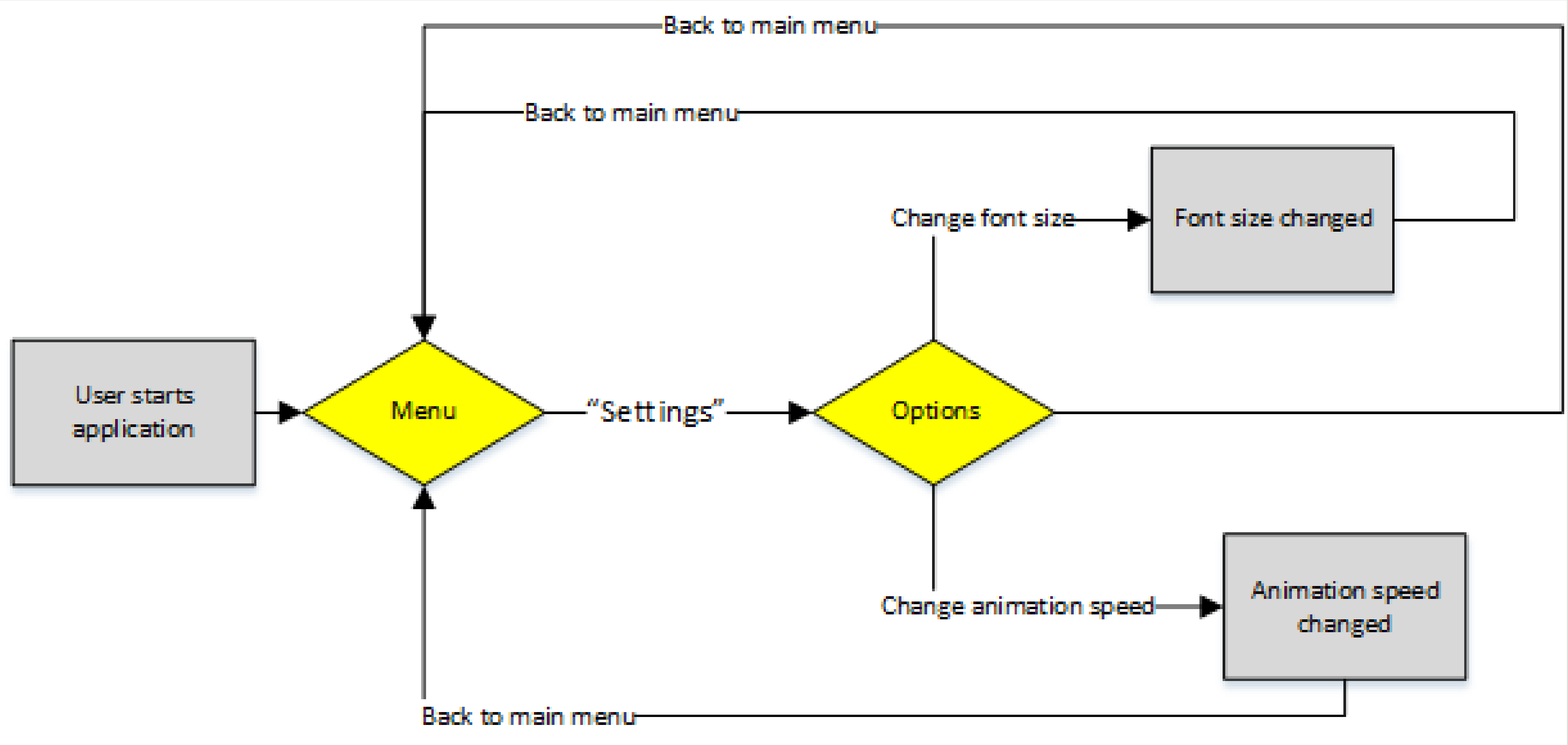
Changing of the speed of the animation

Allows the user to work with ease in an environment where they are most comfortable working in

Aim is to make the settings feature to be easily implementable for other developers



The UI design of the settings feature



The flowchart of the settings feature

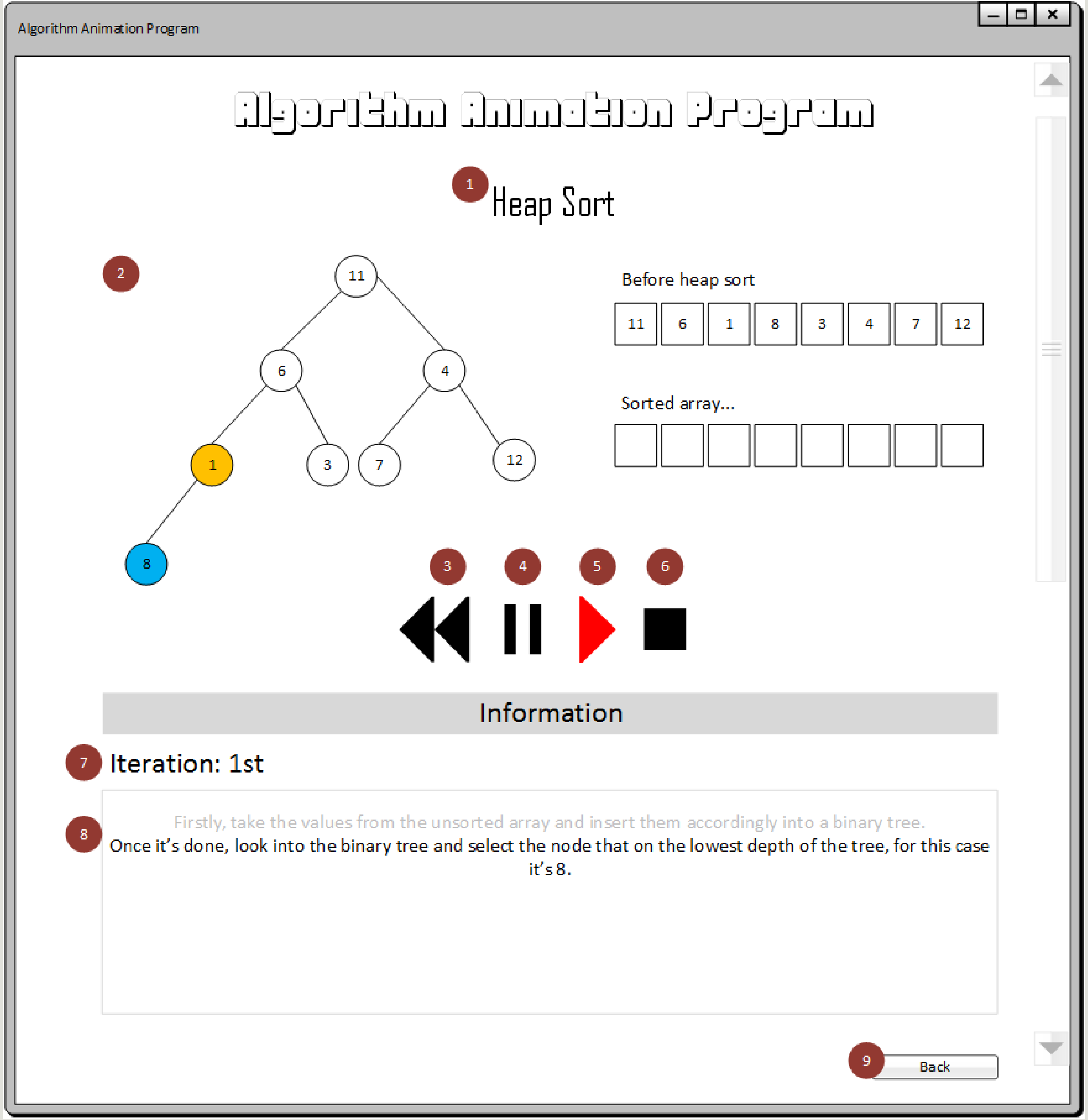
The animation feature

Main feature of this project

Shows the list of algorithms available for users to view

Input page

The animation page, and controls



Algorithm Animation Program

Heap Sort

Before heap sort

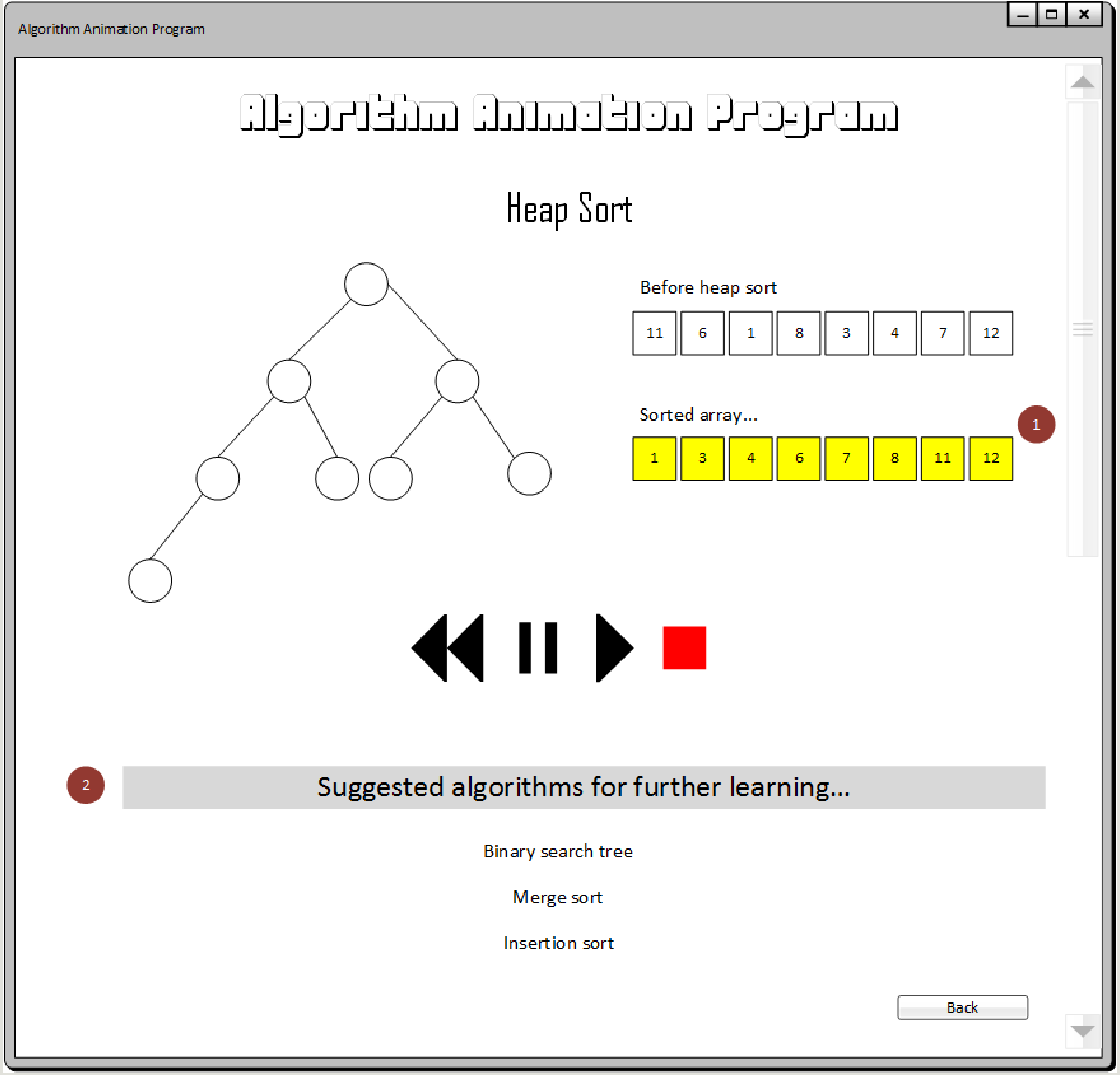
11	6	1	8	3	4	7	12

Sorted array...

Information

Iteration: 1st

Firstly, take the values from the unsorted array and insert them accordingly into a binary tree.
Once it's done, look into the binary tree and select the node that on the lowest depth of the tree, for this case it's 8.



Algorithm Animation Program

Heap Sort

Before heap sort

11	6	1	8	3	4	7	12

Sorted array...

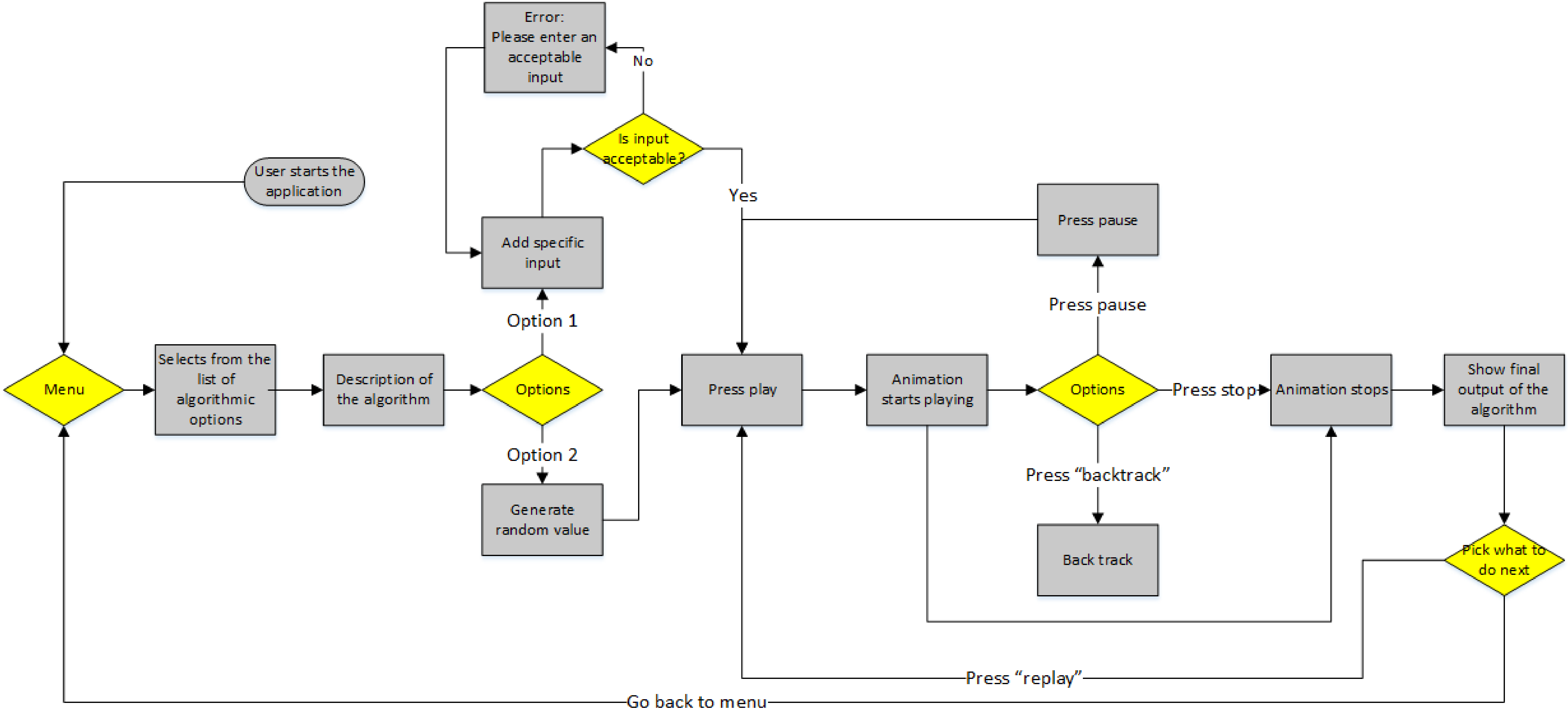
1	3	4	6	7	8	11	12

Suggested algorithms for further learning...

Binary search tree

Merge sort

Insertion sort


Error:
Please enter an acceptable input
No
Is input acceptable?
Yes
User starts the application
Add specific input
Press pause
Option 1
Press pause
Menu
Selects from the list of algorithmic options
Description of the algorithm
Options
Press play
Animation starts playing
Options
Press stop
Animation stops
Show final output of the algorithm
Option 2
Press “backtrack”
Generate random value
Back track
Pick what to do next
Press “replay”
Go back to menu

Design of the animations

The algorithms that will be included in the program…

Fractional Knapsack problem

Knapsack problem

Activity Selection problem

Matrix multiplication

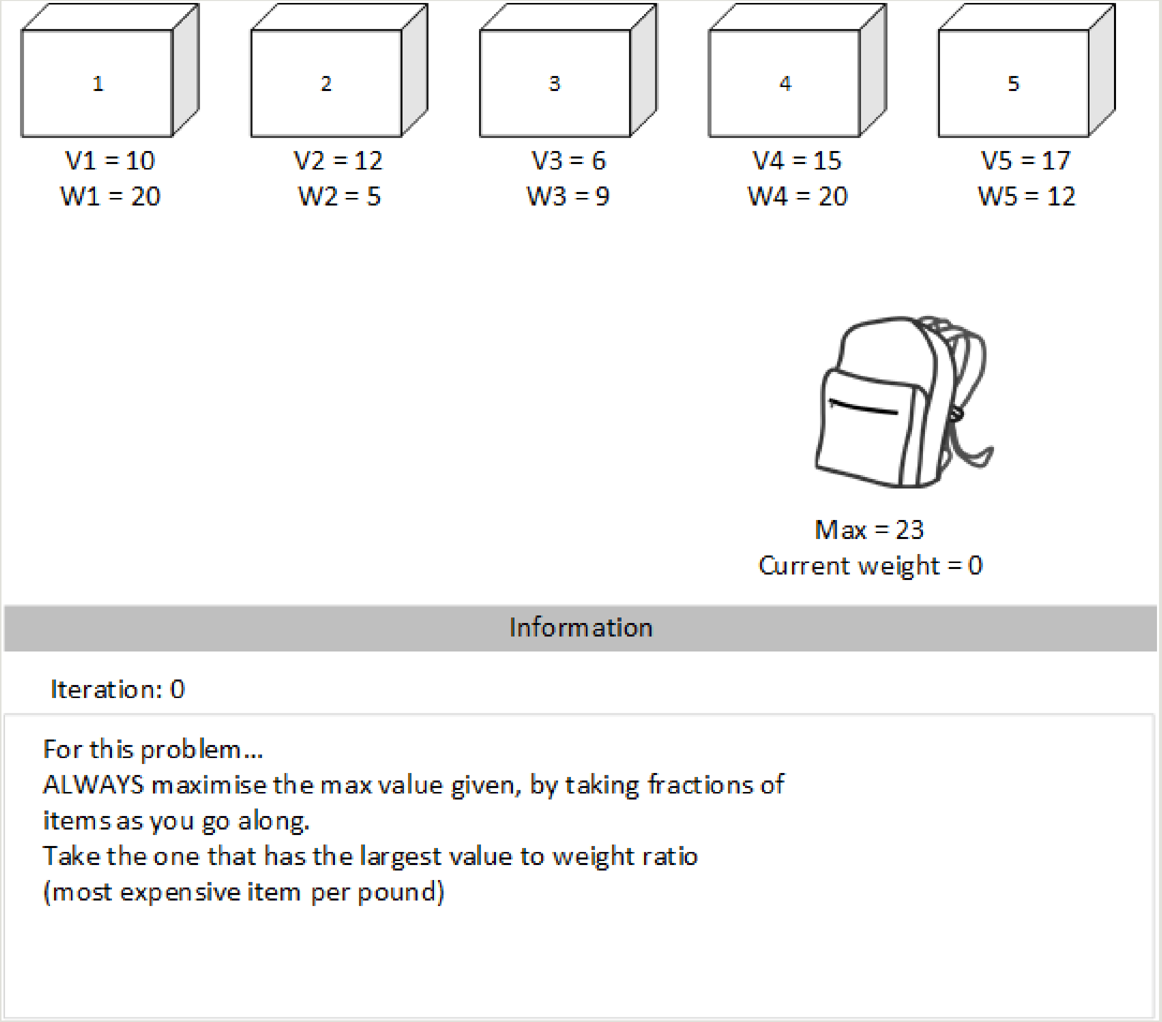
Rod cutting problem

Bubble sort

Merge sort

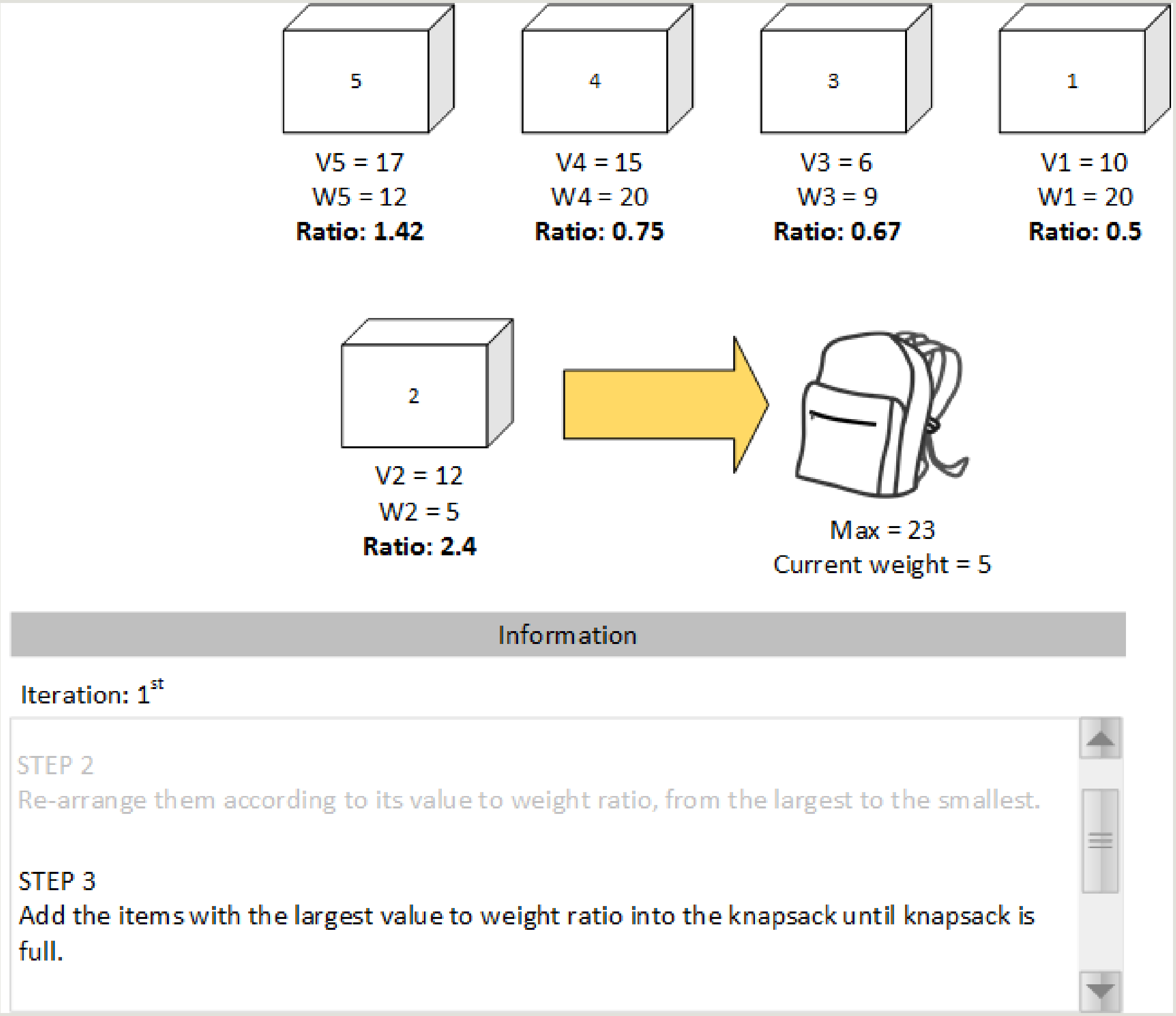
Insertion sort

Fractional Knapsack Problem



Iteration: 0

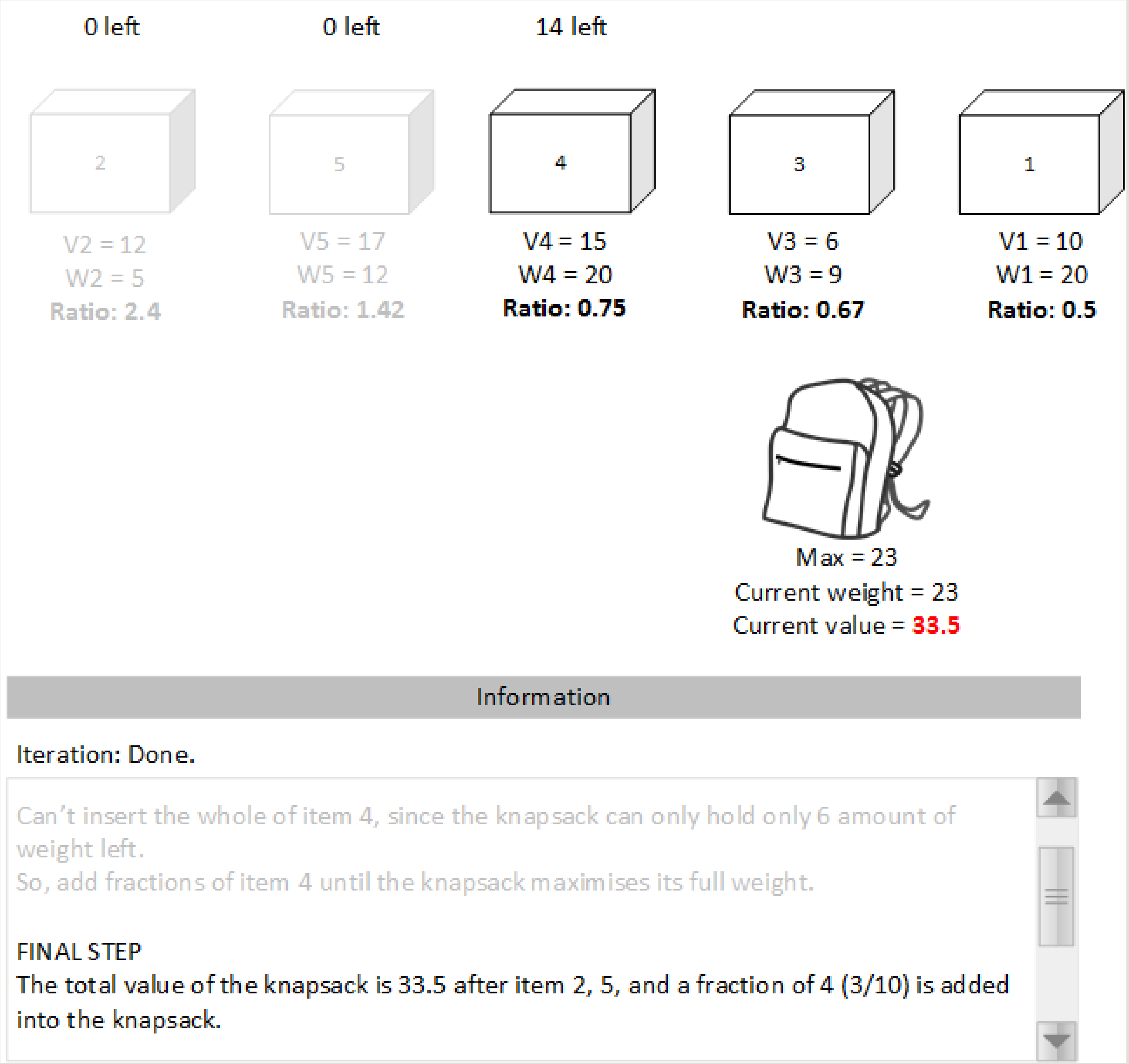
For this problem...
ALWAYS maximise the max value given, by taking fractions of items as you go along.
Take the one that has the largest value to weight ratio
(most expensive item per pound)



Iteration: 1st

STEP 2
Re-arrange them according to its value to weight ratio, from the largest to the smallest.

STEP 3
Add the items with the largest value to weight ratio into the knapsack until knapsack is full.



V2 = 12
W2 = 5
Ratio: 2.4

V5 = 17
W5 = 12
Ratio: 1.42

V4 = 15
W4 = 20
Ratio: 0.75

V3 = 6
W3 = 9
Ratio: 0.67

V1 = 10
W1 = 20
Ratio: 0.5

Max = 23
Current weight = 23
Current value = **33.5**

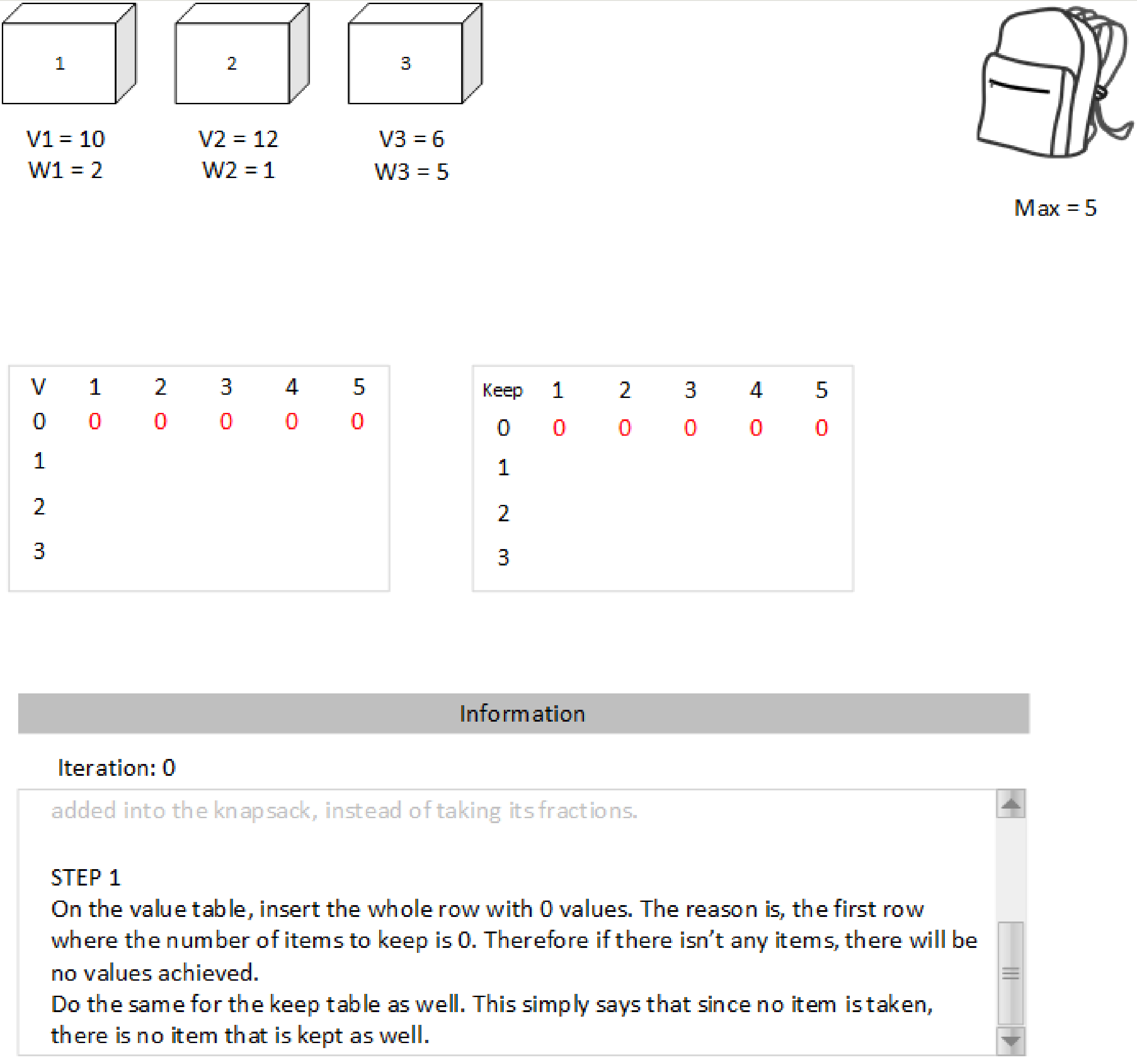
Information

Iteration: Done.

Can’t insert the whole of item 4, since the knapsack can only hold only 6 amount of weight left.
So, add fractions of item 4 until the knapsack maximises its full weight.

FINAL STEP
The total value of the knapsack is 33.5 after item 2, 5, and a fraction of 4 (3/10) is added into the knapsack.

Knapsack Problem



V	1	2	3	4	5
0	0	0	0	0	0
1					
2					
3					

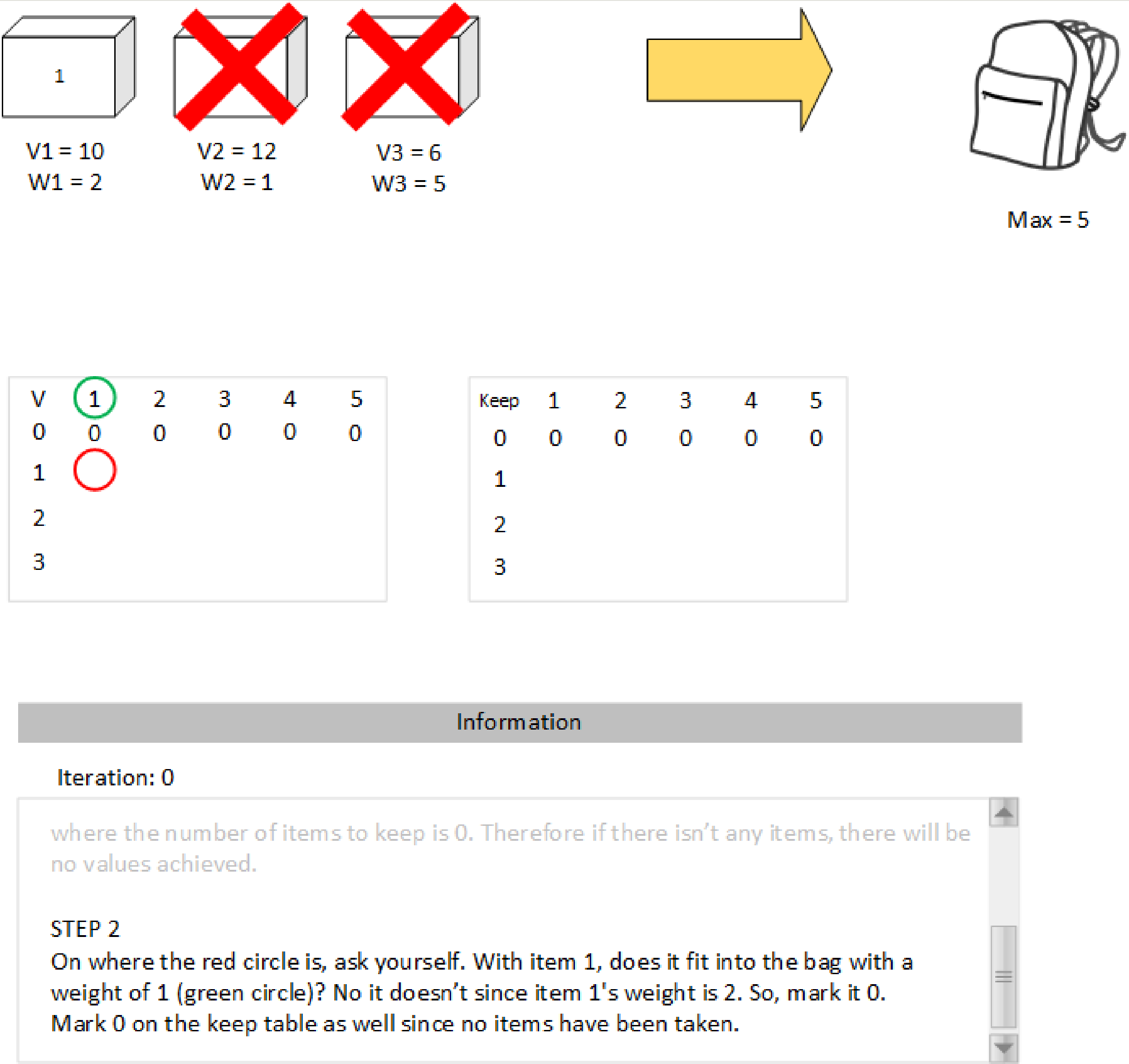
Keep	1	2	3	4	5
0	0	0	0	0	0
1					
2					
3					

Information

Iteration: 0

added into the knapsack, instead of taking its fractions.

STEP 1
On the value table, insert the whole row with 0 values. The reason is, the first row where the number of items to keep is 0. Therefore if there isn't any items, there will be no values achieved.
Do the same for the keep table as well. This simply says that since no item is taken, there is no item that is kept as well.



V	1	2	3	4	5
0	0	0	0	0	0
1					
2					
3					

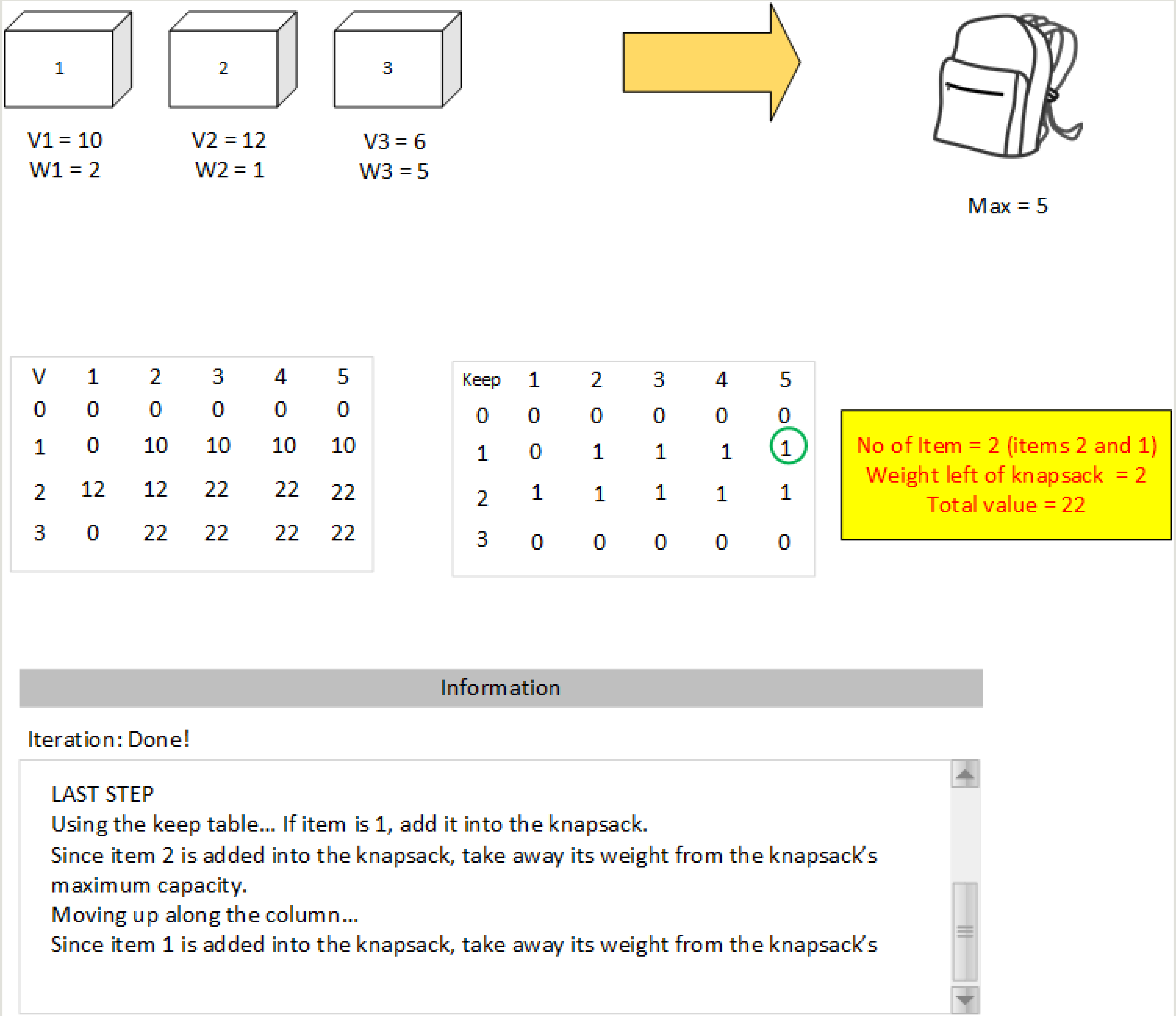
Keep	1	2	3	4	5
0	0	0	0	0	0
1					
2					
3					

Information

Iteration: 0

where the number of items to keep is 0. Therefore if there isn't any items, there will be no values achieved.

STEP 2
On where the red circle is, ask yourself. With item 1, does it fit into the bag with a weight of 1 (green circle)? No it doesn't since item 1's weight is 2. So, mark it 0.
Mark 0 on the keep table as well since no items have been taken.



V1 = 10
W1 = 2

V2 = 12
W2 = 1

V3 = 6
W3 = 5

Max = 5

V	1	2	3	4	5
0	0	0	0	0	0
1	0	10	10	10	10
2	12	12	22	22	22
3	0	22	22	22	22

Keep	1	2	3	4	5
0	0	0	0	0	0
1	0	1	1	1	1
2	1	1	1	1	1
3	0	0	0	0	0

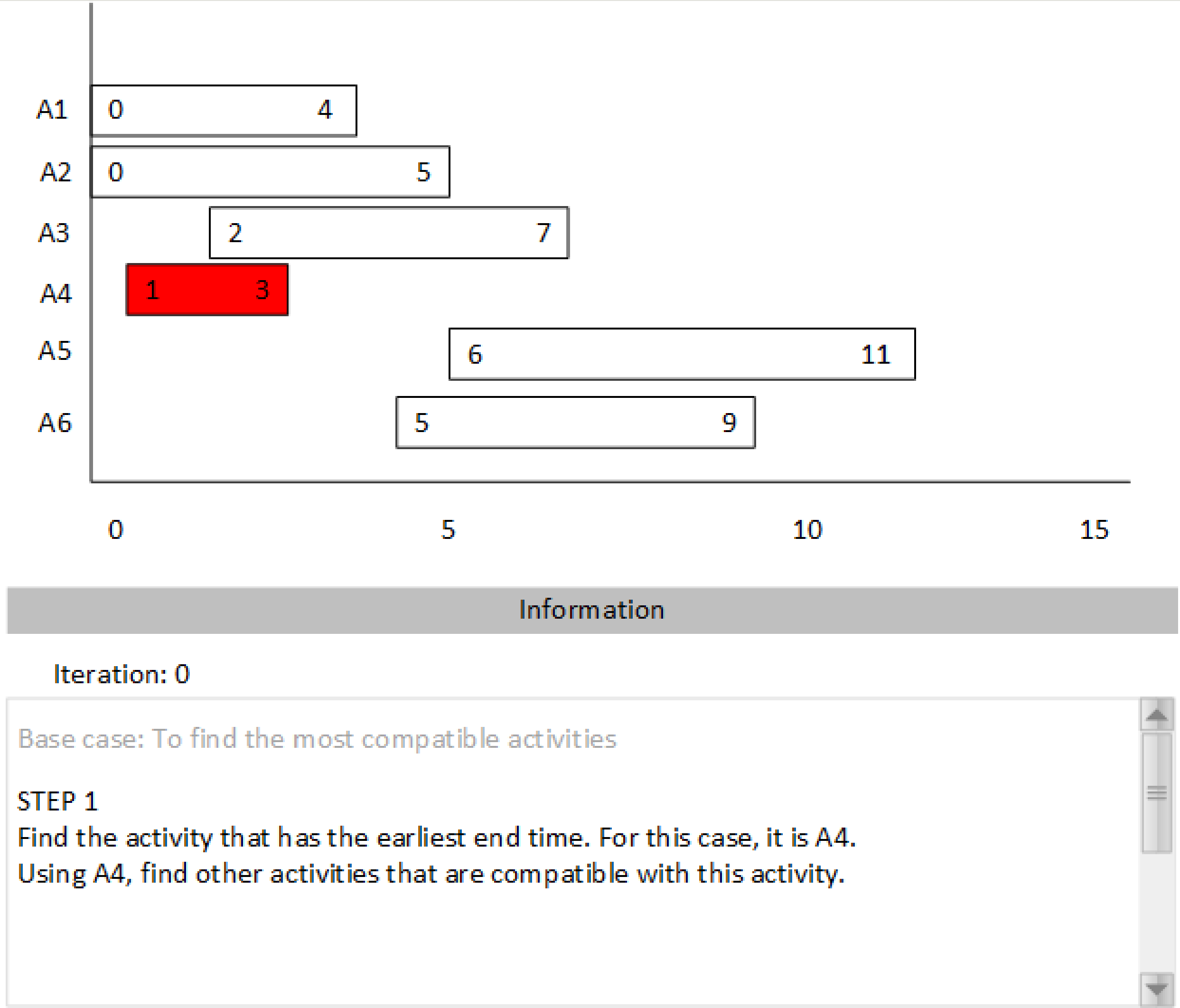
No of Item = 2 (items 2 and 1)
Weight left of knapsack = 2
Total value = 22

Information

Iteration: Done!

LAST STEP
Using the keep table... If item is 1, add it into the knapsack.
Since item 2 is added into the knapsack, take away its weight from the knapsack's maximum capacity.
Moving up along the column...
Since item 1 is added into the knapsack, take away its weight from the knapsack's

Activity Selection Problem

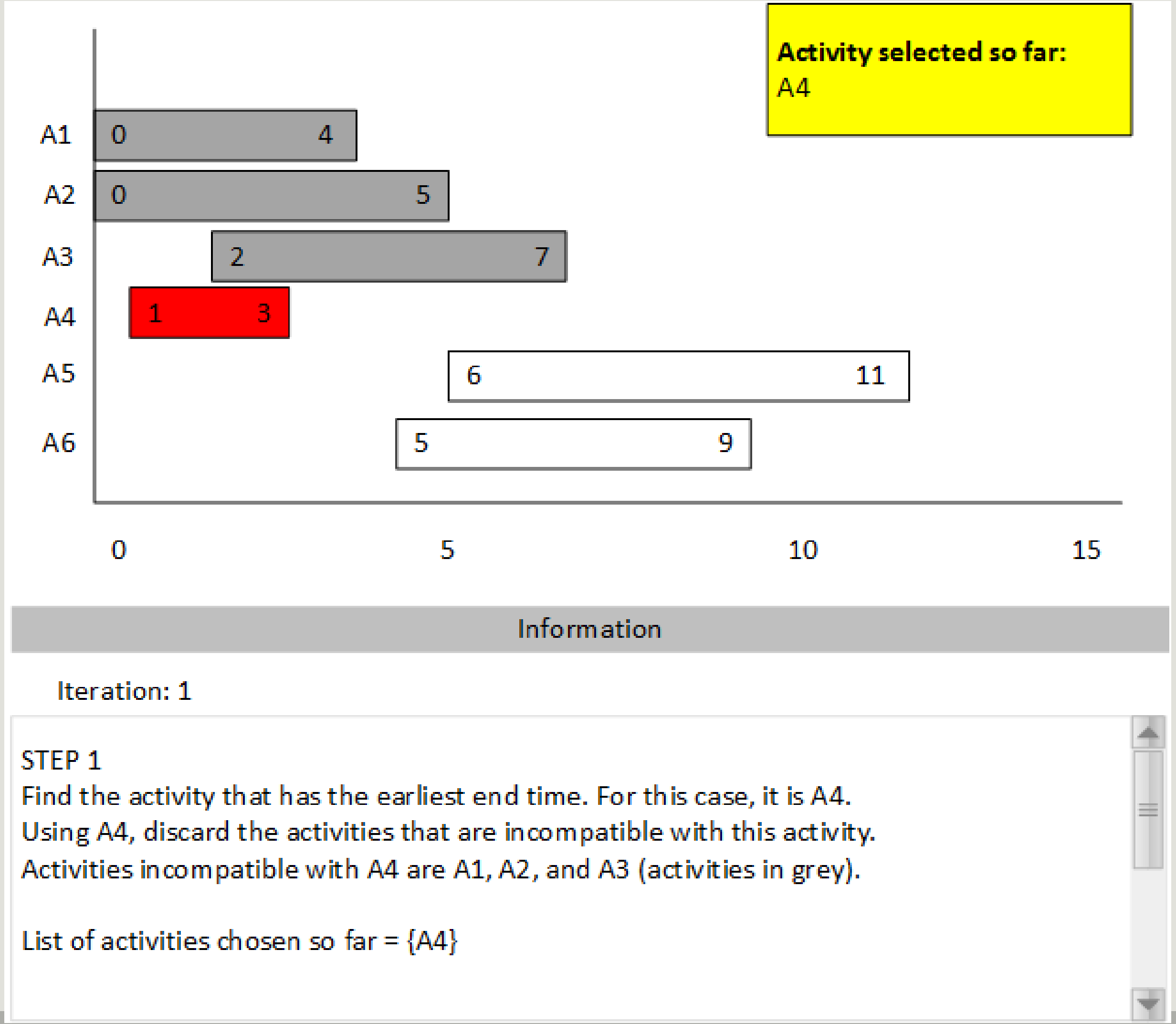


Information

Iteration: 0

Base case: To find the most compatible activities

STEP 1
Find the activity that has the earliest end time. For this case, it is A4.
Using A4, find other activities that are compatible with this activity.



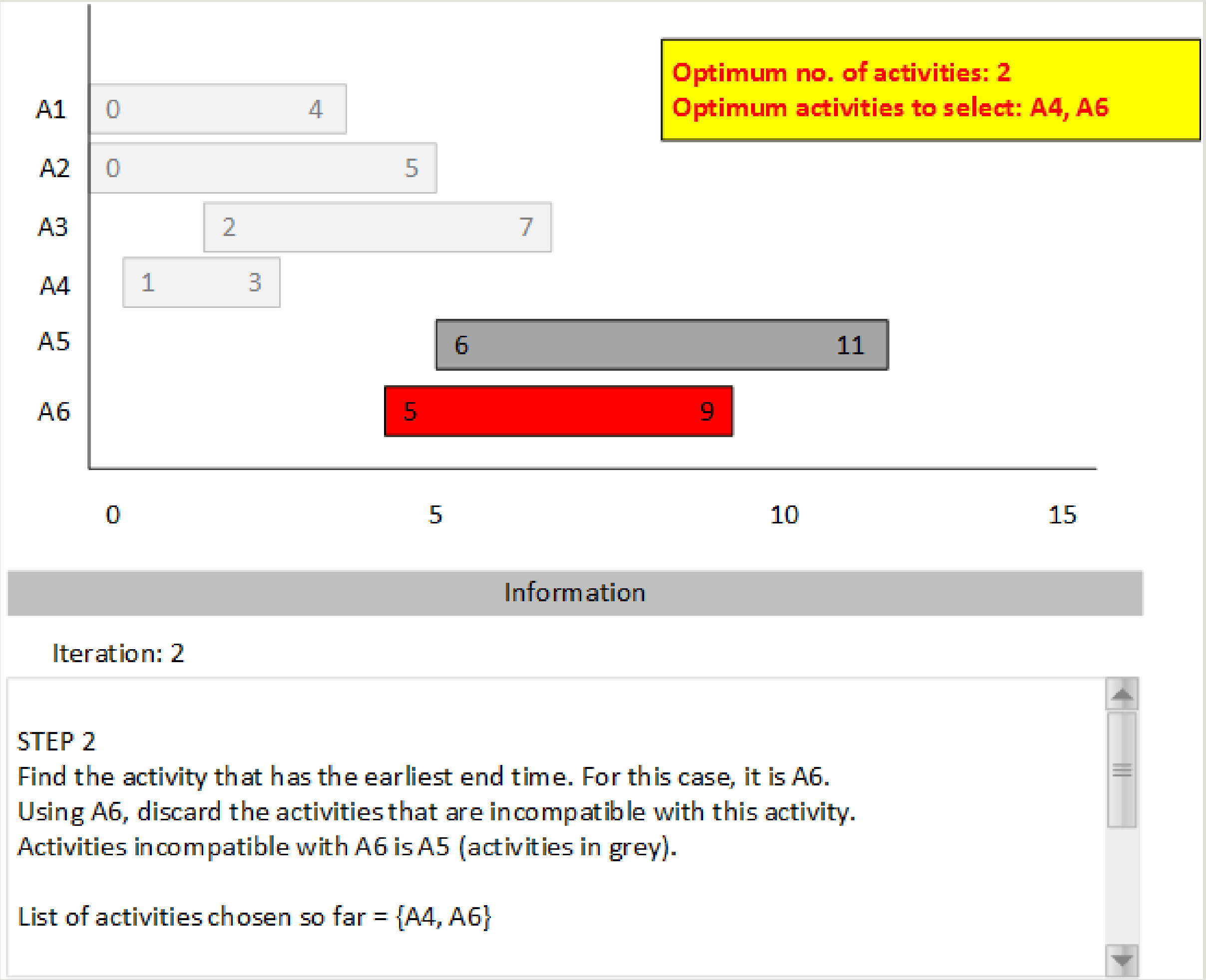
Activity selected so far:
A4

Information

Iteration: 1

STEP 1
Find the activity that has the earliest end time. For this case, it is A4.
Using A4, discard the activities that are incompatible with this activity.
Activities incompatible with A4 are A1, A2, and A3 (activities in grey).

List of activities chosen so far = {A4}



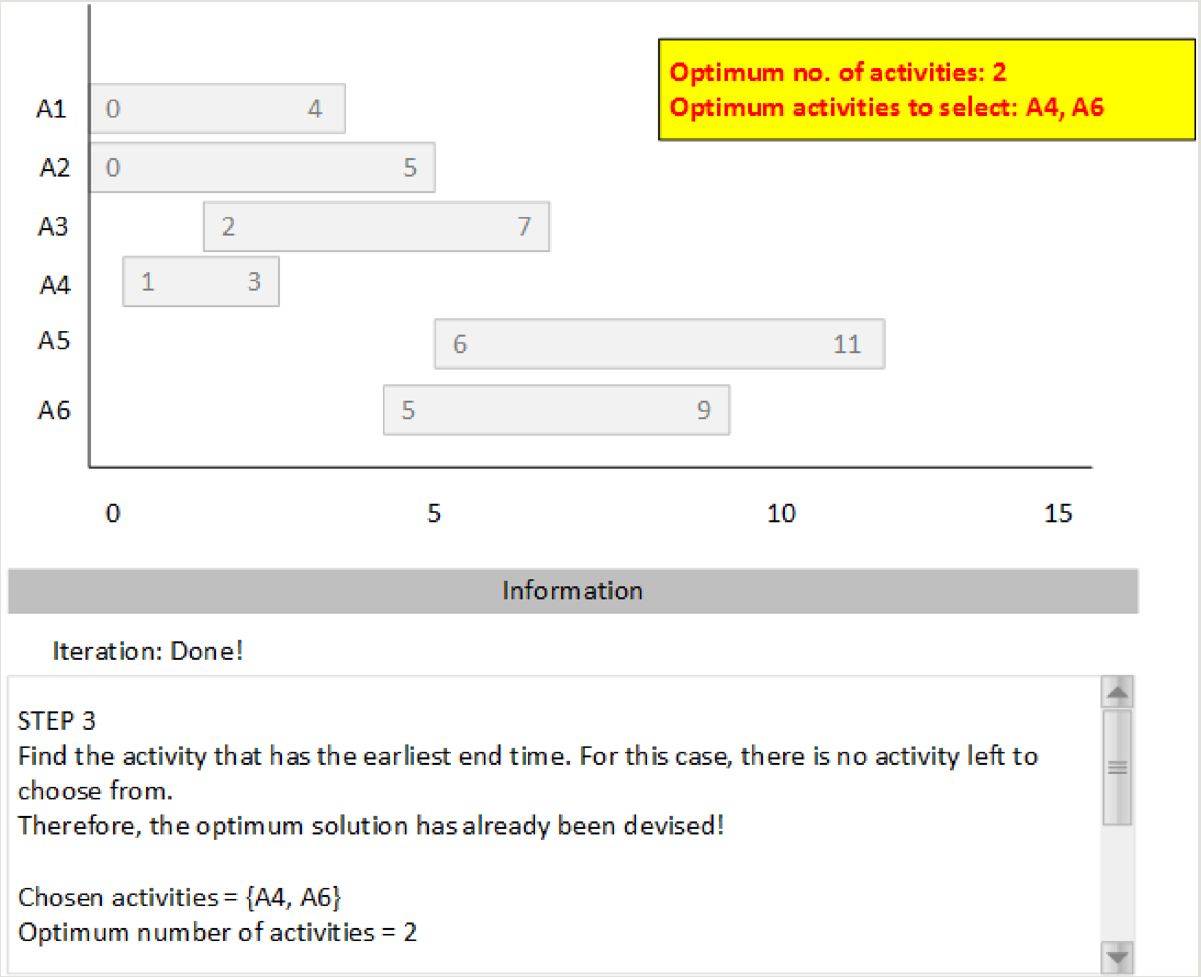
Optimum no. of activities: 2
Optimum activities to select: A4, A6

Information

Iteration: 2

STEP 2
Find the activity that has the earliest end time. For this case, it is A6.
Using A6, discard the activities that are incompatible with this activity.
Activities incompatible with A6 is A5 (activities in grey).

List of activities chosen so far = {A4, A6}



Optimum no. of activities: 2
Optimum activities to select: A4, A6

Information

Iteration: Done!

STEP 3
Find the activity that has the earliest end time. For this case, there is no activity left to choose from.
Therefore, the optimum solution has already been devised!

Chosen activities = {A4, A6}
Optimum number of activities = 2

Evaluation design

The main criteria to assess during evaluation

1. Usability
2. Comprehensiveness
3. Correctness

The evaluation process

Testers are required to use the program

Assess the program based on criteria mentioned before

Filling up the feedback form

Changes that was made that deviates from the specification documentation

Removed several algorithms in order make time

Using of testers to conduct the evaluation of the program

Addition of the settings feature