Advanced Algorithms

# Warsaw University of Technology

Final Documentation

## January 11th, 2019

### List of Modifications

The following modifications have been noted and changed or added from conducting the tests described later in this documentation:

1. Added the case for where is the number of boxes.
2. Added validation for the case where .
3. Added validation for the case where .

### User’s Manual

This document is archived along with another archive file called “Applications.rar”, in a file named “Boxes.rar”.

Inside the “Applications.rar” archive, you will find two folders:

* CLI: this folder contains the version of the boxes application that is based on the command line interface called “boxes.py” and a file that contains all the input text files: empty.txt, one.txt, no\_size.txt, and incorrect\_l.txt.
* GUI: this folder contains the version of the boxes application that has a graphical use interface called “boxes.exe”, an image file for the application to work called “box.ico”, and the uncompiled version holding the code for the executable application called “boxes.py”.

The whole directory map is as follows:

* Boxes
  + Advanced Algorithms - Final Documentation.pdf
  + Applications
    - CLI
      * Boxes.py
      * Data
        + Empty.txt
        + One.txt
        + No\_size.txt
        + Incorrect\_l.txt
        + Wrong\_index.txt
    - GUI
      * Boxes.exe
      * Box.ico
      * Boxes.py

#### CLI Manual

To run boxes.py do the following steps:

1. Open the CLI folder
2. Hold shift + right click inside the folder window and select “Open Command Prompt here” on windows. Otherwise, open terminal and use cd to change directory to the CLI folder.
3. In the CMD/Terminal window, you will have the following options to type after typing in “python boxes.py”, which you can mix and match between them however you desire:
   * “-s **or** --solution [file\_path]”: this argument prompts the python script to generate a solution file inside the CLI directory called “output.txt” based on the input text file from the given path.
   * “-i **or** --input”: this argument generates an input text file inside the CLI folder called “input.txt” filled with randomly generated values.
   * “-b **or** --boxes [number\_of\_boxes]”: combine this argument with the “-i” or “--input" argument to set the number of boxes that you want to generate. Note that if not specified, the default value would be used which is 10.
   * “-m **or** --maximum [maximum\_box\_size]”: combine this argument with the “-i” or “--input" argument to set the maximum box size of the boxes that you want to generate. Note that if not specified, the default value would be used which is 50.
   * “-h **or** --help": this is the conventional help argument that would display all of the descriptions of the aforementioned arguments.

Do note that running the command “python boxes.py” by itself will run the script and prompt it to generate a solution to “output.txt” from the file “input.txt” (which you can add yourself or generate from adding the input argument). Otherwise if the file “input.txt” doesn’t exist, and you run the command “python boxes.py”, the script will return a missing file error.

#### GUI Manual

To run the application, simply double click the file “boxes.exe” and wait for it to initialize. It might take a few moments for it to initialize on some systems so patience would be advised.

Once run, the following options are available:

1. The first text field which has the default value of 10, is the field that contains the number of boxes to be generated.
2. The second text field which has the default value of 50, is the field that contains the maximum size of the boxes to be generated.
3. The first button labeled “Generate Boxes”, generates a new file (if the file doesn’t already exist) or replaces the one called “input.txt” (so you do not have to remove the file after every generation) if the file has been generated before, filled with randomly generated values for the boxes based on the valued from the first and second text fields.
4. The second button labeled “Generate Solution”, generates a new file (if the file doesn’t already exist) or replaces the one called “output.txt” (so you do not have to remove the file after every generation) holding the solution of the inputs from the file “input.txt”. Note that, logically, you have to generate an input file before pressing this button to get a solution.
5. The third and final button labeled “Plot Time Complexity”, plots the runtime of thousands of generated boxes in real time as a fun way to prove the Time Complexity of proven in the functional documentation. Do note that pressing this button will freeze the application while your computer’s CPU calculates and plots the time complexity on a graph to display it for you.

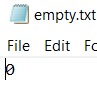
Side note, the Time Complexity figure used in this Final Documentation was generated from that last GUI functionality.

### Test Description, Conclusions, and Time Complexity Validation

#### Test Description

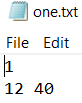
The following tests have been conducted to verify the correctness of the output of the program:

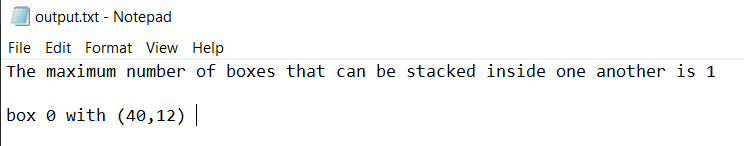
1. The empty set test (with the file “empty.txt”) holding the following values:



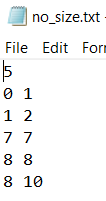
* + It’s a trivial task on paper to show that there is nothing to return.
  + We get the following result from running the script from the CLI:

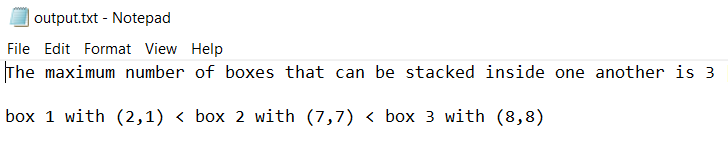
1. The one element in the set test (with the file “one.txt”) holding the following values:



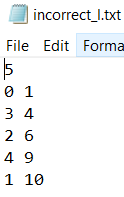
* + It’s a trivial task on paper to show that the largest increasing subset of a unit set is the set itself, and in this case out input file contains one element which is the tuple “12 40”, therefore the largest increasing subset is the tuple itself.
  + We get the following result from running the script from the CLI: 

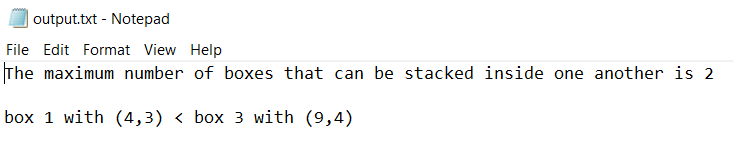
1. The test where we have boxes with either their length or width equal to 0 (with the file “no\_size.txt”) holding the following values:



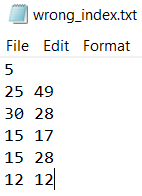
* On paper, we can solve this and we will get the following largest increasing subsequence: (1,2) < (7,7) < (8,8) or (8,10). Noting that (0,1) is not an actual box since a box cannot have a size of 0.
* We get the following result from running the script from the CLI:

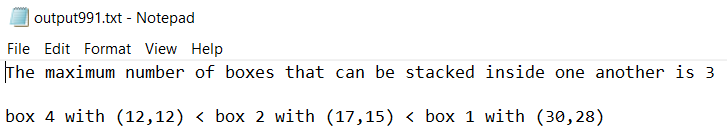
1. The test where we have a longest increasing subset of widths that fit into each other but have lengths that would not (with the file “incorrect\_l.txt”) holding the following values:



* On paper, we can solve this and we will get the following largest increasing subsequence: (3,4) or (2,6) < (4,9). Also note that (0,1), similar to the previous test, is not an actual box since a box cannot have a size of 0.
* We get the following result from running the script from the CLI:

1. The test where the lis function returns elements that exist more than once in the vector (with the file “wrong\_index.txt”) holding the following values:



* On paper, we can solve this and we will get the following largest increasing subsequence: (12,12) < (15,17) or (15,28) < (28,30) or (25,49).
* We get the following result from running the script from the CLI: 

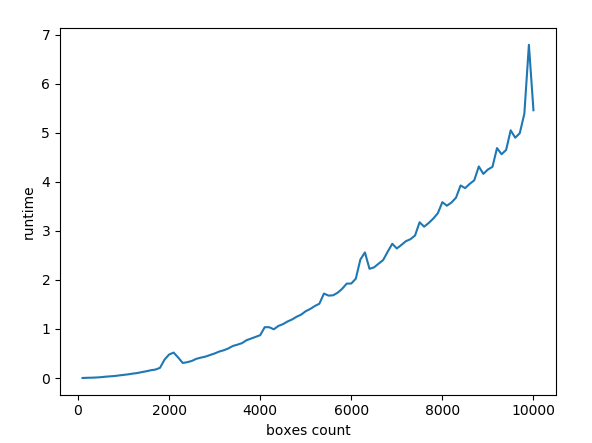
#### Conclusions

The following conclusions can be noted from the testes that were conducted:

1. This test was conducted to make sure that the algorithm can handle the base case scenario where we have no boxes to stack.
2. This test was conducted to make sure that the algorithm can handle the case where we only have one box in our set. To make sure that the algorithm returns the whole set as the longest increasing subset.
3. This test was conducted to make sure that the program can handle cases where boxes have no size to them to be able to be used to stack, in cases where we have either or .
4. This test was conducted to make sure that the program can handle cases where the longest increasing subsequence generated by the lis function has elements with non-increasing indexes. In particular suppose in the case above, the lis function tells us that the box with goes into the box with . But, the lengths of these boxes disagree, with . Thus, such cases should be skipped.
5. This test was conducted to make sure that the program can handle the cases where we have multiple instances of the element in our vector , but those instances do not all have the same values. So, the index of the recurrent elements in would be the same index (code wise). Thus, we ignore the first element that the index returns and select the last one to have the index.

#### Time Complexity Validation

The following figure validates the Time Complexity of proven in the Functional Documentation, the axis represents the number of boxes, and the axis represents the runtime of each iteration of the whole solution algorithm:



Do note that the spikes are due to the mechanical CPU defects, which are a normal occurrence as all life’s imperfections. But the whole graph has the shape of validating the assumption of .

### Job Partition

The job partitioning process was decided, and jobs were conducted based on the team’s strengths as follows:

* Example Instance and Solution Algorithm: Elie Saad, Alaa Abboushi
* Example Instance and Solution Document: Elie Saad,
* Functional Documentation: Elie Saad,
* Beta Version of the Application: Elie Saad,
* Application CLI: Elie Saad,
* Application GUI: Elie Saad,
* Final Documentation: Elie Saad.