Sabancı University
Faculty of Engineering and Natural Sciences

CS 300 Data Structures

Assigned: November 30, 2022　　　　**Due: December 12, 2022 @ 23:55**

PLEASE NOTE:

SOLUTIONS HAVE TO BE YOUR OWN. NO COLLABORATION OR COOPERATION AMONG STUDENTS IS PERMITTED.

10% PENALTY WILL BE INCURRED FOR EACH DAY OF OVERTIME. SUBMISSIONS THAT ARE LATE MORE THAN 3 DAYS WILL NOT GET ANY CREDITS.

SUBMISSIONS WILL BE MADE TO THE SUCOURSE SERVER. NO OTHER METHOD OF SUBMISSION WILL BE ACCEPTED.

1 Introduction

The İstanbul skyline problem, often mistakenly called the New York skyline problem, is the problem of drawing the skyline of a city given a set of buildings of that city. In this homework you will write a program whose input is a list of the locations and sizes of buildings, and whose output is a description of the visible skyline, that is, how the city looks as an outline, when viewed from a distance. For a real-life version of this problem, go to Vaniköy on the Asian side of the Bosphorus, and, as you sip your tea, contemplate on how the skyline will look as the sun sets over the European side.

To make things simple, the buildings will be given using just two dimensions, x and y, rather than three, ignoring the depth value. (Think of x as horizontal and y as vertical.) Each building will be a rectangle with its base on the x-axis. Thus, a building can be completely described by giving the x-coordinates of its left and right sides, and the y coordinate (or height) of its top side. The first line of the input indicates how many buildings there are in the city, and each succeeding line will indicate the x-coordinate of the left side of a building followed by the y-coordinate of the top side of the building, followed by the x-coordinate of the right side of the building. The following is an example of a valid input to the program:

```
4
2 50 10     x_of_left, y_of_top, x_of_right
35 20 55
30 60 45
5 75 25
```

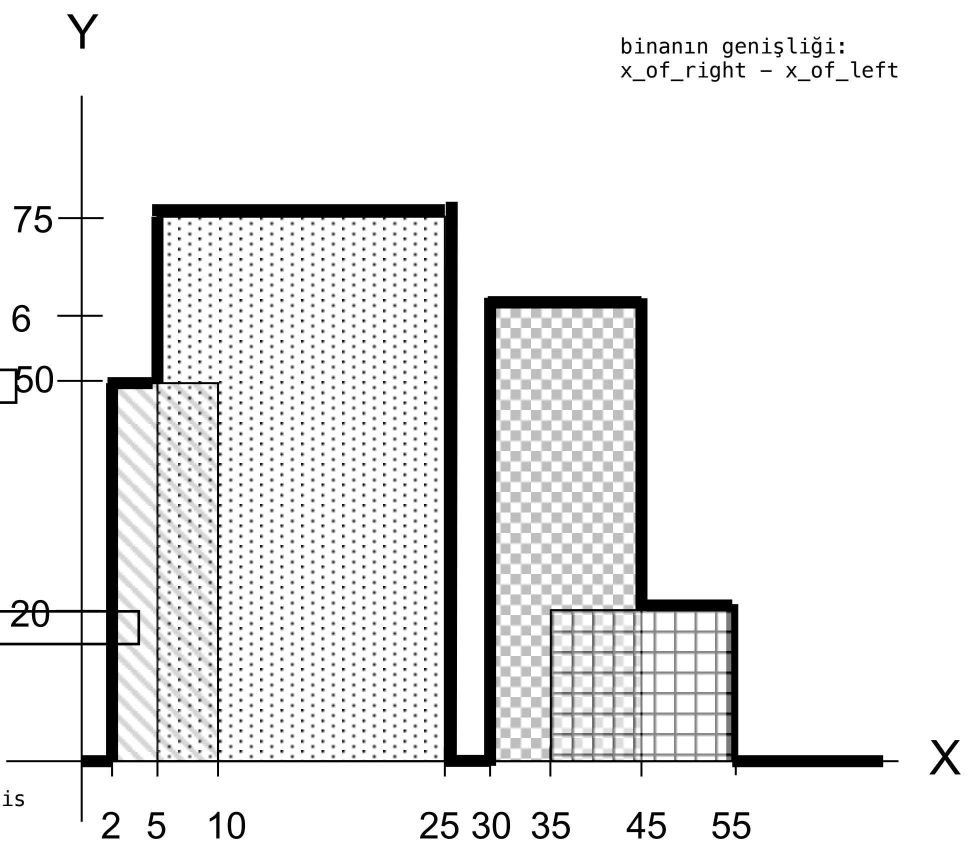
Notice that buildings may overlap. For example, the first building, which spans 2 through 10 on the x-axis, overlaps the fourth building, which spans 5 through 25 on the x-axis. If the right x-coordinate of a building is the same as the left x-coordinate of another building, then the two buildings are considered to be side by side (though their heights may differ!)

The output of this program should be a description of the skyline of the city. The skyline is the upper outline of the buildings in the city. To be more precise, the skyline is a function that maps each x-coordinate to the y-coordinate of the tallest building that covers that x-coordinate. Since all of our buildings are rectangles, this function will be a curve consisting of horizontal and vertical line segments. Your program should describe the skyline function by listing each x coordinate at which the height of the skyline changes, along with the new coordinate of the skyline at that x. Your program should do this in the direction of increasing x coordinate (i.e., left to right). For example, the following output describes the skyline of the input above:

```
0 0
2 50
5 75
25 0
30 60
45 20
55 0
```

This output indicates that initially the skyline is at height 0 (which may be different if there is a building starting at 0). At x-coordinate 2, the height of the skyline becomes 50. The height remains 50 until x-coordinate 5, at which point (because of the fourth building) it becomes 75. Then, at x-coordinate 25 the height becomes 0, and so forth. The following figure depicts this instance of the problem where the buildings are shown with rectangles and the skyline is shown with the think outline.

binanın genişliği:
x_of_right - x_of_left



at x = 2 the height of the skyline becomes 50

since there is no building at x=0 the height of the skyline is 0

2 The Modified Priority Queue Class

The first part of this assignment is to implement a class that we will call a "Modified Priority Queue," (MPQ) that will be useful in computing the skyline of a set of buildings. The MPQ is a simple variation on the priority queue class that we used heaps to implement. The only difference is that the items that are maintained by a MPQ object internally have two components (say *value* and *label*), as opposed to just a value that is used to compare the items in a priority queue. The first component is again a value with which the items can be compared to each other just as in a priority queue. The other, *label*, is an identifying number that identifies a specific item in the MPQ. Thus for instance you can insert an item and either access or delete an item with the maximum value component or with a specific identifying number.

max queue ??

Your class definition should at least have the following method that could be useful for the rest of the homework.

value is used to compare the items in the MPQ and label is an identifying number (like id) which identifies a specific item in MPQ.

- Constructor
- Destructor
- `void insert(Comparable value, int label):` This method inserts an item with the given value and label into the MPQ.
- `Comparable Remove(int label):` This method removes the value with this label and returns it.
- `Comparable GetMax( ):` This method returns the maximum value that is currently stored. (**Note that contrary to priority queues, we do not remove the value from the MPQ data structure!**)

- Finally, our class must allow us to check if it is empty, using the function `Boolean IsEmpty( ).`

You will use heaps to implement MPQs. Let us assume that you call the array for the heap as *Heap*. The tricky part is to maintain a link between a value and a label assigned to it. For this, it will be convenient to have a parallel private array of integers, called *Location*, whose *i*th entry contains the position in the heap of the item having label *i*. Thus you can locate a heap entry with a given *label* in *O(1)* time, instead of searching for it in the heap in *O(N)* time. At all times, for the valid entries in the heap, *Heap[Location[i]].label* equals *i*, and *Location[Heap[j].label]* equals *j*. When you move an element around in the heap, you should be careful to change the corresponding value in *Location* to contain the new position of the element.

Heap arrayinde bir value'nun storelandığı indexi değiştirirsen, Location arrayde value değerine eşit olan index değerinin storeladığı elementi de modify etmelisin

yeşille çizili olan yer ne demek: Location[i] bize aradığımız value'nun Heap'teki indexini veriyor. Heap[Location[i]] bize Heap'te bakmamız gereken indexi gösteriyor .label dediğimizde ise Heap'teki i. indexteki label'ı bize return ediyor.

diyelim ki Heap'te "4" valuesunu 3. index ten 5. indexe taşıdık, bu durumda Location arrayde 4. index'te storelaığımız value'yu da 3'ten 5'e değiştirmek zorundasın

3 The Skyline Program

Once you have implemented the MPQ Class, you will write the code that computes the skyline of a city. We suggest the following algorithm outline but certainly you are welcome to come up with your own, but you should employ the MPQ class.

1. Read in the descriptions of all of the buildings from a file called **input.txt**. The format will be as given earlier.

2. Once you have read all the data in, put all of the x-coordinates of the left and right sides in a separate array (of size twice the number of buildings) together, and sort them into ascending order of the x-coordinates using some fast sorting algorithm (you can use heapsort, but you will have to modify it a bit). With each x-coordinate, also keep the identity of the corresponding building in the same array (e.g., the sequence number of the building as you read them in), and which side (left or right) it belongs to.

3. Make a left-to-right sweep across the city by examining the x-coordinates in the sorted list in ascending order. During the course of the sweep, the MPQ is used to keep track of which buildings span the current x coordinate and which of the buildings has the maximum height. You will have to figure out how to use the MPQ class for this. After each insertion and deletion, check to see if the maximum height of the buildings in the MPQ has changed. If so, output the x- coordinate that has just been examined, followed by the new current maximum height to the standard output.

4 What you should turn in

You should turn in a fully documented source code listing for your MPQ class and your main application program source with any addition classes and procedures. Your programs will be tested with the sample data that we will provide using the format above. Make sure your programs accept input or produce outputs in the specified format and way. **If they do not, then you will not get any credits.**

Your code should be submitted to SUCourse at the deadline given on the first page. We suggest that you take our warning above that indicates that you should submit only your work and not collaborate with others. **We assume that by submitting your homework in the manner below, you are certifying that you are submitting your own work.**

You should follow the following steps:

- Name the folder containing your source files as **XXXX-NameLastname** where **XXXX** is your student number. Make sure you do NOT use any Turkish characters in the folder name. You should remove any folders containing executables (Debug or Release), since they take up too much space.

- Use the Winzip or an equivalent program to compress your folders to a compressed file named, for example, **5432-AliMehmetoglu.zip**. After you compress, please make sure it uncompresses properly and reproduces your folder exactly. We will NOT accept any excuses for files not appearing in your submitted files.

- You then submit this compressed file in accordance with the deadlines above.

Your homework will be graded in the following way:

- If the source code does NOT compile you get 0 points and that is it.

- We will run your program with one or more different **input.txt** files that we will prepare. If your output is completely correct for all the tests you will get 90 points maximum.

- Note that your outputs should be in the exact same format we described above. If they are not, you will lose points.

- The style of your code will be graded on clarity, commenting, etc. This will cover 10 points. You will however NOT get this credit if your program totally fails in the test we perform. A nice looking but useless program is just a useless program no matter how nice it is.

Good luck!

```
the idea behind heap sort: Delete the root
(max element but store its value into a
variable) which will result in putting the
last element into the place of the root,
then place the element that we deleted
into the place of the element that we put
into the roots place
```