Submission Guidelines: You have to submit your work on Blackboard by the due date. For each of the programming assignments you must use the header template provided in Blackboard. Make sure that you identify yourself in the header, and any collaborators. If none, write "none". Your answers to questions that do not require coding must be included in this header as well. Your code must follow customary formatting standards, as posted in Blackboard. Format will also be part of your grade. To complete your submission, you have to upload to Blackboard the source file and nothing else. **The submission will not be accepted in any other format.**

Style and Correctness: Keep in mind that your goal is to communicate. Full credit will be given only to the correct solution which is described **clearly**. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Assignment #2 Grading Rubric

Coding:

Program characteristic	Program feature	Credit possible	Coding questions
Design 30%	Algorithm	30%	
Functionality 30%	Program runs without errors	20%	
	Correct result given	10%	
Input 15%	User friendly, typos, spacing	10%	
	Values read in correctly	5%	
Output 15%	Output provided	10%	
	Proper spelling, spacing, user friendly	5%	
Format 10%	Documentation: name, collaborators, header, etc.	5%	
	Clarity: comments, indentation, etc.	5%	
	TOTAL	100%	

Non-coding:
Embedded in questions.

1a(25)	1b(25)	2(10)	3(10)	4(10)	5(20)	**TOTAL**(100)

Assignment:

The intended usage of a data structure is crucial to choose the most efficient one. Hence, good general-purpose data structures minimize assumptions about input characteristics. However, many times data queries can be sped up significantly designing application-specific data structures if input information is available.

Take for instance the segment intersection decision problem. That is, given a set of segments, decide whether there exists in the set at least one pair of segments that intersect. As we learned, we can solve this problem using a sweeping technique that is common to computational-geometry algorithms (Chapter 33.2). This algorithm runs in $O(n \log n)$ time using Heapsort to sort the segments and a self-adjusting BST to maintain the sweeping line status.

The implementation of the sweeping mentioned above is quite efficient taking into account that finding all intersections is known to be in $\Omega(n^2)$. However, a natural question is whether it is possible to decide intersection in $o(n \log n)$ using input information. For instance, the sweeping algorithm studied assumes that all segments are available from start (off-line). Thus, information such as maximum or minimum coordinates of segment end points is easily computable in $O(n)$. Then, one could design an early-stopping algorithm that checks first if the range of coordinates has the same order of magnitude as n, and if so decides intersection using the sweeping algorithm with a $o(n \log n)$ implementation, or otherwise use the standard $O(n \log n)$ implementation.

The purpose of this assignment is to implement the described strategy and compare efficiency experimentally drawing conclusions from the results obtained. To simplify the task, we will assume that **all endpoint coordinates are different and integer**. Your task is the following.

1. (a) **(25 points)** Implement the sweeping algorithm to decide intersection in a set of segments following the pseudocode in Chapter 33. Sort the segments using Heapsort (Section 6.4) and maintain the sweeping status using a self-adjusting BST (e.g., RB trees (Chapter 13)).

   (b) **(25 points)** Implement the sweeping algorithm to decide intersection in a set of segments following the pseudocode in Chapter 33. Sort the segments using Radix Sort (Section 8.3) and maintain the sweeping status using a van Emde Boas tree (Chapter 20). Given

that all coordinates are different, you can maintain the sweep-line status using the y coordinate of the segment endpoint as the key.

Remember that to achieve correctness you must use crossproducts. In other words, you have to follow the sweeping pseudocode in the book. No points for code implementing other algorithms. (You can reuse Heapsort, Radix Sort, BST, and/or vEB tree code if you have it.)

2. **(10 points)** Write a method that does the following for an input set of n segments where all endpoint coordinates are different integers. Find the range of x-coordinates and the range of y-coordinates. Let $u = \max\{x_{\max} - x_{\min}, y_{\max} - y_{\min}\}$. If $2n < u < 10n$ decide intersection calling the method in Part 1b. Else, decide intersection calling the method in Part 1a.

3. **(10 points)** Write a program that does the following for an input value n.

   - Create a "smooth" set of n non-vertical segments where all endpoint coordinates are different integers and $2n < u < 10n$. Recall that the sweeping algorithm stops after finding the first intersection. So, define the set so that only a few intersect at the end of the x range to attain a good evaluation of worst-case running time.
   - Create a "sparse" set of n non-vertical segments based on the smooth set but changing only one segment so that $u > 100n$. Make sure this modified segment does not intersect any of the other segments to avoid impact on running time due to new intersections.
   - Call the method in Part 2 with each of these sets and display the running time of each in nanoseconds.

4. **(10 points)** Run your program and fill the table below (adjust the values of n as needed according to your platform to obtain at least 5 measurements).

nanoseconds	$n = 10^2$	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$
Smooth set					
Sparse set					

5. **(20 points)** What is the best function of n and/or u fitting your measurements? Notice that you are evaluating rate of growth, you are not being asked which one is faster. Are the results consistent with the query time of the data structures and algorithms used? Justify explaining the running time of each.