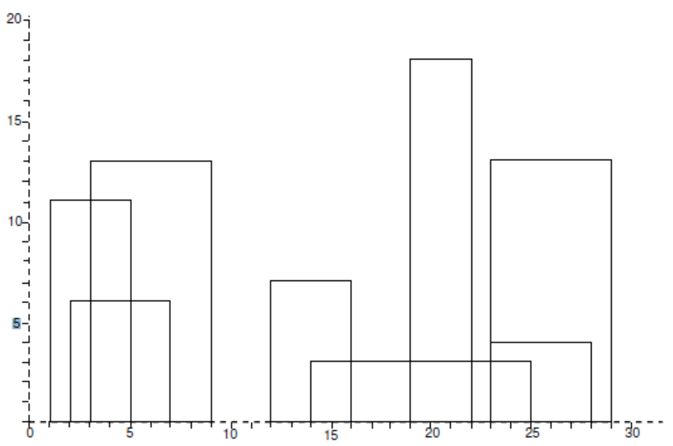
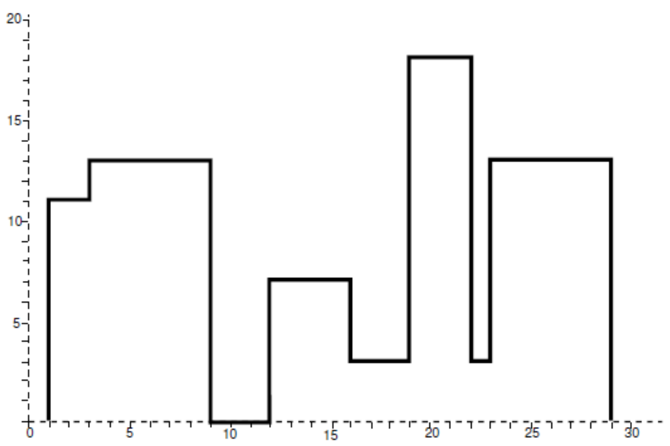
Programming Exercise 3: Divide and Conquer Problems Instructor: Neil Patrick Del Gallego
Individual

[45 pts] General Instructions: This is a programming exercise for you to test your algorithmic design skills, specifically for implementing **divide and conquer** solutions. You are to devise a **DC solution** for all problems specified. The most efficient solution for these problems are clearly **DC!** **<u>Not providing a DC solution will have no merit.</u>** For convention and consistency, the mini-program are to be done in JAVA. Please use only 1 java class for this exercise.

1. **[15 pts] The Skyline Problem**



buildings



skyline

Given the figure above, you are to identify the range of skylines, given a set of buildings. You are to design a program to assist an architect in drawing the skyline of a city given the locations of the buildings in the city. To make the problem tractable, all buildings are rectangular, and they share a common bottom (the city they are built in is very flat). The city is also viewed as two-dimensional. A building is specified by an ordered triple **(L_i, H_i, R_i)** where **L_i** and **R_i** are the left and right X coordinates, respectively, of building **i** (**$0 < L_i < R_i$**) and **H_i** is the height of the building. In the diagram above, buildings are shown with the following triples:

(1, 11, 5), (2, 6, 7),(3, 13, 9),(12, 7, 16),(14, 3, 25),(19, 18, 22),(23, 13, 29),(24, 4, 28)

In the diagram above, the skyline is represented as a sequence that represents the "path" of the skyline in terms of X and Y coordinates. X coordinates are underlined.

(**1**, 11, **3**, 13, **9**, 0, **12**, 7, **16**, 3, **19**, 18, **22**, 3, **23**, 13, **29**, 0)

Input: The input is a sequence of building triples. All coordinates of buildings are integers less than 10,000 and there will be at least one and at most 5,000 buildings in the input. Each building triple is on a line by itself in the input file. All integers in a triple are separated by one or more spaces. The triples will be sorted by L_i, the left x-coordinate of the building, so the building with the smallest left x-coordinate is first in the input.

Output: The output should consist of the vector that describes the skyline as shown in the example above. In the skyline vector (v_1, v_2, v_3, . . ., v_{n-2}, v_{n-1}, v_n), the v_i such that **i is an even number represent a horizontal line** (height). The vi such that **i is an odd number represent a vertical line (x-coordinate).** The skyline vector should represent the "path" taken, starting at the minimum x-coordinate and traveling horizontally and vertically over all the lines that define the skyline. Thus, the last entry in all skyline vectors will be a '0'.

Sample Input:	Sample Output:

1 11 5
2 6 7
3 13 9
12 7 16
14 3 25
19 18 22
23 13 29
24 4 28
	1 11 3 13 9 0 12 7 16 3 19 18 22 3 23 13 29 0

Coding Instructions
For #1, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public class Building {
  public int left;  //the left X-coordinate
  public int right; //the right X-coordinate
  public int height; //the height

}
```

```
public static void skyline (Building[] B) {
   //put your code here and the print statements for the output.
}
```

2. **[15 pts] Iterative Merge Sort**
   You are to implement an iterative merge sort version that still performs in O (n log n) time.

   Input: The input will be space-separated integers
   Output: The output should be space-separated integers sorted in ascending order.

Sample Input:	Sample Output:
7 8 9 4 3 2 1 9 8 7 6 2 2 0	1 2 3 4 7 8 9 0 2 2 6 7 8 9

   Coding Instructions
   For #2, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void mergesort (int[] A) {
   //put your code here and the print statements for the output.
}
```

3. **[15 pts] Closest Pair of Points**
   You are going to implement the DC solution for the closest pair of points. You are given **N** points on a 2D plane. Find the pair of points with the smallest euclidean distance. Assume that all points will be unique and there is only one pair with the smallest distance.

   Input: First line of input will contain 2 ≤ N ≤ 50000 and then **N** lines follow each line containing two integers giving the X and Y coordinate of the point.

   Output: Output 3 numbers **i, j,** and **d**, where **i, j(i<j)** are the 0-based indexes of the point pair in the input and d is the smallest distance at most 6 decimal places.

Sample Input:	Sample Output:

5
0 0
0 1
100 45
2 3
9 9
	0 1 1.000000

   Coding Instructions
   For #3, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public class Pt {
  public int x;  //X-coordinate
  public int y; // Y-coordinate}

public static void closestPair (Pt[] P) {
   //put your code here and the print statements for the output.
}
```

Submission Instructions
- Use only 1 Java class for this exercise. You are free to use as many functions, inner classes or structs to organize your code.
- Follow the coding instructions! Not following the coding instructions will merit deductions.
- Test your code thoroughly and try out different test cases, especially the edge cases (0 or N values). Your instructor will run with at least 3 test cases to check your code.
- Optimize your performance. If your code executes for more than 5 seconds, it will automatically fail the test case! Your code will be tested on an i7 processor with 8GB of RAM.
- Submit this exercise on or before the indicated deadline in Canvas.