Homework 8

Version 0.1

COS125 - Instructor: Zachary Hutchinson

Due: Friday December 8th @12AM (midnight)

Submission Instructions

These must be followed or you will lose points.

1. Submit your coded solution as a .py file. Code in any other file format will be rejected.
2. Your .py file should be named: yourlastname_hwX.py. The 'X' in the filename is replaced by the homework number.
3. Your code must have as a comment at the top of the file: your name, the homework and the names of anyone from whom you received help. This includes students in the course as well as course staff. For example, if you went to Boardman 138 and received help from an MLA, put their name on your file. If a classmate helped you debug a bit of code, put their name down. This is for your benefit.
4. If you do not manage to squash all the bugs in your program, include a comment at the top of the file detailing the outstanding bugs. Acknowledging bugs is a sign of a mature programmer. Doing so will not eliminate point deductions but it might mitigate them. It shows you care about your work.

Learning Objectives

- Practice implementing a recursive algorithm
- Use Cartesian coordinates in a graphical context.
- Learn an efficient way to represent space.

Problem Description

Space (physical space) presents many interesting computational problems. One such problem is identifying which objects exist in a specific place. For example, if I have a program that tracks sharks at sea and I want to know which sharks, if any, are off the coast of Maine, my program would need to iterate through all the tracked sharks to find those near Maine.

We have talked about Big-O and complexity. Such a straight-forward solution would be Big-O of N (the number of sharks).

Another way to look at the problem of identifying which objects occupy a region of space is to invert the problem. Instead of finding location via the object, we find the object via the location.

One such method is to represent space using a *tree*. For example, a quad tree continually subdivides a space into four equal quadrants until a space contains less than a minimum amount of objects. The benefit of such a representation is that large empty spaces are eliminated from our search. Locating which objects occupy a space (or if a space is empty) is closer to a logarithmic (Big-O of log(N)) operation. Once a tree is *built*, searching for objects stored in a tree is logarithmic because we eliminate many objects from our search when we eliminate regions of space.

Your task will be to create a graphical, 2D representation of how a simpler tree, a binary tree, subdivides space based on the distribution of a set of random points. Our binary-tree does not split a space into four subregions automatically (like a quadtree) but splits a region in two based on the distribution of the points in the space.

NOTE: Your program will not actually create a tree data structure. That is for a later course. But your program will go through the motions of creating a tree. Instead of storing information for later use it will simply draw the results. My hope is that this will give you an intuitive understanding of the benefits of divide and conqueror algorithms and recursion.

Task Requirements

1. Your program will model a square 2D space that is M-by-M in size (I will give you example values for each of these contants later in this PDF).
2. Your program will generate N random x,y coordinates where each *x* and *y* is between 0 and M-1.
3. Your program will use the algorithm given in the **Algorithm** section to recursively subdivide the 2D space until each sub-region contains either 0 or 1 point.
    1. **IMPORTANT** Your program must use recursion or you will not receive a grade.
4. Your program will draw the subdivided space and the set of points using the Zelle Graphics Library (I have included with the homework the graphics.py module and the PDF documentation for it).
5. Make sure the version you submit has the size and number of points constants set to values that are *reasonable* and that you like. Reasonable means, don't upload a version that creates a 2k by 2k window with 10,000 points. The grader might not have such a big screen. A simple example (512 or 1024 size) with a modest number of points (20 to 500). You aren't confined to these values. I give them just so you know what *reasonable* means.

Modified Binary Tree Algorithm

Given a 2D space, the algorithm will:

- If the space contains 0 points, it will draw the space and fill it with a light gray color.
- If the space contains 1 point, it will draw the space and fill it with white.
- If the space contains more than 1 point it will divide the space using the following heuristic:
    - Find the min and max x-coordinate within the space.
    - Find the min and max y-coordinate within the space.
    - Calculate the differences between max and min coordinates (e.g. maxX - minX).
    - If the difference in x coordinates is larger than the difference in y coordinates, then the space will be split **in half** along the x-axis.
    - Else if the y difference is greater than the x, split the space **in half** along the y-axis.
    - For the case where they are equal, you should add this case to one of the two cases above. It does not matter which one.
    - Once you split the space, you must **recursively** use this algorithm on the two subregions.

Drawing the Tree and Points

Use the Zelle graphics library to create a window, draw the base case subregions and the points. Read through the PDF documentation for instructions on how to use the library. The library uses classes and

objects, which we will start discussing next week. Part of the problem solving of this assignment is the use of written documentation to understand the purpose and uses of a library.

Storing The Points

You are free to store the points in any way you like. As stated above, you are not actually creating a tree data structure. You are just creating a graphical representation so there's no need to intelligently store the points but it will simply your algorithm if you do. Here are two suggestions:

You could store all the points in a 2D list representing every coordinate in the original space. This is rather space inefficient but it would make looking for points easier.

Probably a better approach is to partition the points once you divide the space into two sets, one for each subregion. Each set would contain only those points in the subregion. This would allow you to pass in smaller and smaller lists with each recursive call.

It is also acceptable to simply pass in the entire list of points with each recursive call and search through the entire list to find the points in each subregion. The goal is to practice recursion.

Constants

Your program should use several constants:

- SIZE: the size of the space (and window).
- NUM_POINTS: the number of points to generate.

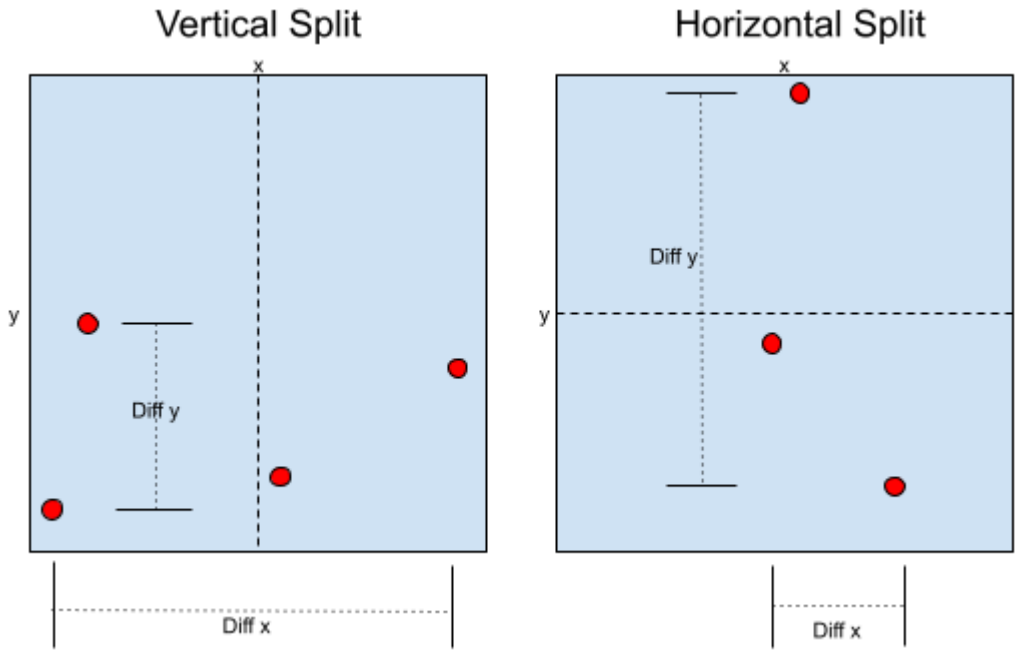
I suggest that for SIZE, you use powers of 2: 512 or 1024. If you don't use a power of two, you can get some misalignment of subregions without additional coordinate tweaks.

Hints

1. Layout the algorithm on paper. Identify the base and recursive cases. Use paper and pencil to figure out how to divide a rectangle in half vertically and horizontally.
2. Practice using the Zelle library before jumping into the homework. Make sure you know how to open a window and draw shapes to it. Identify which classes in the library will help you.
3. You must draw the final subregions before the points otherwise you will cover the points.
4. You only need to draw the final subregions (those that won't be split because they have 0 or 1 point in them).

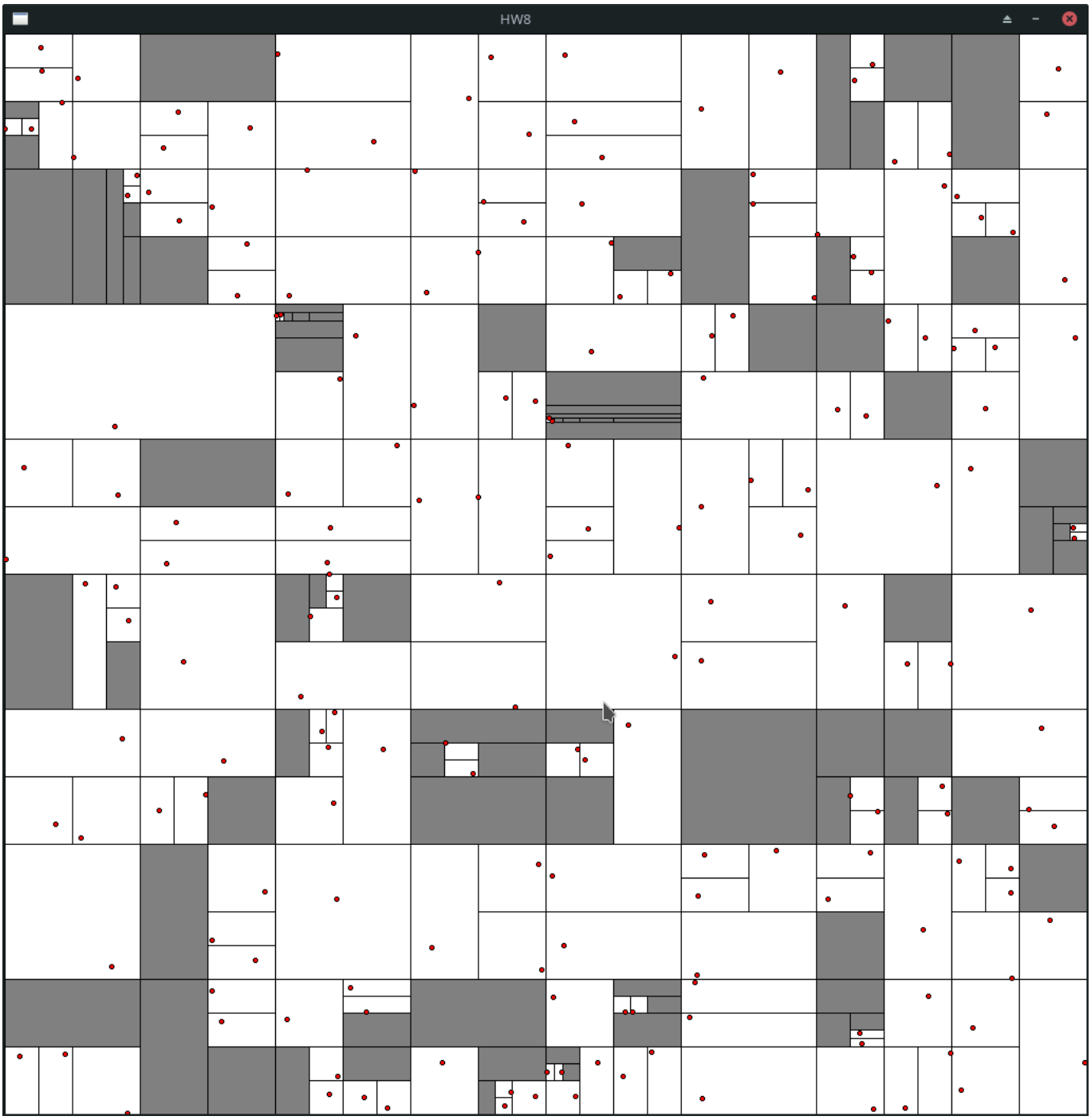
Split Example

The following figure shows how the two recursive cases work to split a space based on the distribution of the points in the space. On the left, the maximum difference between x-coordinates is much larger than y-coordinates, so the space is split evenly along the x-axis. On the right, we have the opposite case. Points are spread out more along the y-axis. So the space is split on the y-axis.



Output Example

On the next page is an example of the algorithm's output using a size of 1024 and 200 points.


HW8