50:198:213 Data Structures
Rutgers University–Camden February 24, 2021
Professor Desmond Lun Handout 6

Problem Set 3

This problem set is due **Wednesday, March 10** at **11:55PM**.
Solutions should be turned in through the course Sakai site in PDF form or scanned hand-written solutions.
Remember, 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.

1. **(18 points)** Heap Delete

   In this problem you will implement an additional operation in a min heap. Namely, `heap.delete(i)` deletes the item at index i in the array.

   Download `ps3_heap.zip`. In `heap_delete.py`, which inherits from the code in `heap.py`, implement `delete(i)`. Your code should run in $O(\log n)$ time where n is the number of nodes currently in the heap.

   The added code doesn't need to be more that a couple lines, but be sure to add comments explaining why it works.

   Run `test-heap_delete.py` to help determine if your new delete method works, and submit `heap_delete.py` to the class Sakai site.

2. Gas Simulation

   In this problem, we consider a simulation of n bouncing balls in two dimensions inside a square box. Each ball has a mass and radius, as well as a position (x, y) and velocity vector, which they follow until they collide with another ball or a wall. Collisions between balls conserve energy and momentum. This model can be used to simulate how the molecules of a gas behave, for example. The world is $400\sqrt{n}$ by $400\sqrt{n}$ units wide, so the area is proportional to the number of balls. Each ball has a minimum radius of 16 units and a maximum radius of 128 units.

   Download `ps3_gas.zip` from the class Sakai site. For the graphical interface to work, you will need to have pygame or tkinter installed. They currently run slightly different interfaces. Feedback is appreciated.

   You may notice that performance, indicated by the rate of simulation steps per second, is highly dependent on the number of balls. Your goal is to improve the running time of the `detect_collisions` function. This function computes which pairs of balls collide (two balls are said to *collide* if they overlap) and returns a set of `ball_pair` objects for collision handling. You should not need to modify the rest of the simulation. (If you think something else should be modified, e-mail me with your feedback.)

(a) **(3 points)** What is the running time of `detect_collisions` in terms of n, the number of balls?

(b) **(8 points)** Argue that the following algorithm is asymptotically faster:

Divide the world into square bins of width 256. For each ball, put it in its appropriate bin. Then for each bin, check for collisions where either both balls are in the bin, or one ball is in the bin, and the other ball is in an adjacent bin.

(c) **(28 points)** Implement the `detect_collisions` algorithm described in part (b). Put your code in `detection.py`, and uncomment the line in `gas.py` just below `detect_collisions` that imports your new code.

Your new code must still find all the same collisions found by the old code (any pair of balls for which `colliding` returns true). To check that you are detecting the same collisions, run your code and the original code with the same parameters, and make sure that they detect the same number of collisions.

In your documentation, do not assume that the reader has read this problem set.

Submit `detection.py` to the class Sakai site.

(d) **(3 points)** Using your improved code, after 1000 timesteps with 200 balls, how many collisions did you get? How many simulation steps per second did you run? How many simulation steps per second could you run with the original code and the same number of balls?