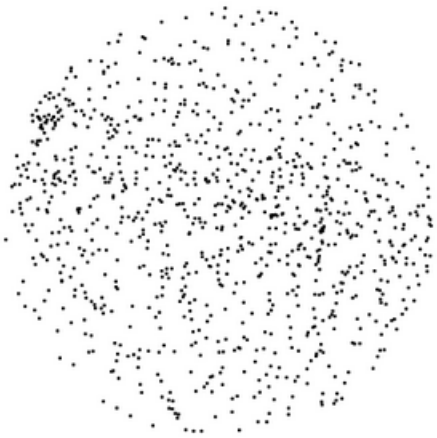
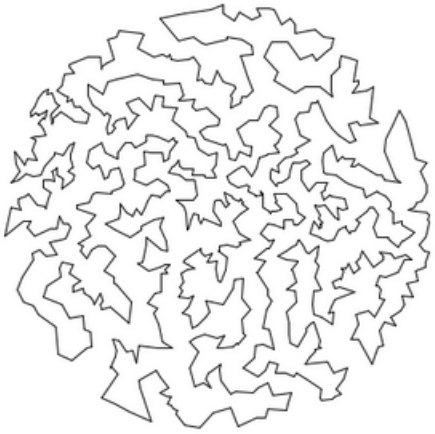
Given N points in the plane, the goal of a traveling salesperson is to visit all of them (and arrive back home) while keeping the total distance traveled as short as possible. The purpose of this project is to implement two greedy heuristics to find good (but not optimal) solutions to the *traveling salesperson problem* (TSP).



1,000 points



optimal tour

Perspective The importance of the TSP does not arise from an overwhelming demand of salespeople to minimize their travel distance, but rather from a wealth of other applications such as vehicle routing, circuit board drilling, VLSI design, robot control, X-ray crystallography, machine scheduling, and computational biology.

Greedy Heuristics The TSP is a notoriously difficult *combinatorial optimization* problem. In principle, one can enumerate all possible tours and pick the shortest one; in practice, the number of tours is so staggeringly large (roughly $N!$) that this approach is useless. For large N, no one knows an efficient method that can find the shortest possible tour for any given set of points. However, many methods have been studied that seem to work well in practice, even though they are not guaranteed to produce the best possible tour. Such methods are called *heuristics*. Your main task is to implement the *nearest neighbor* and *smallest increase* insertion heuristics for building a tour incrementally. Start with an empty tour and iterate the following process until there are no points left:

- *Nearest neighbor heuristic*: Read in the next point, and add it to the current tour *after* the point to which it is closest. If there is more than one point to which it is closest, insert it after the first such point you discover.
- *Smallest increase heuristic*: Read in the next point, and add it to the current tour *after* the point where it results in the least possible increase in the tour length. If there is more than one point, insert it after the first such point you discover.

Point Data Type The `Point` data type defined in `point.py` represents a point in the plane, as described by the following API:

method	description
`Point(x, y)`	construct a new point p at (x, y)
`p.distanceTo(q)`	the Euclidean distance between points p and q
`p.draw()`	draw point p to standard draw
`p.drawTo(q)`	draw a line segment between points p and q
`str(p)`	the string representation of point p

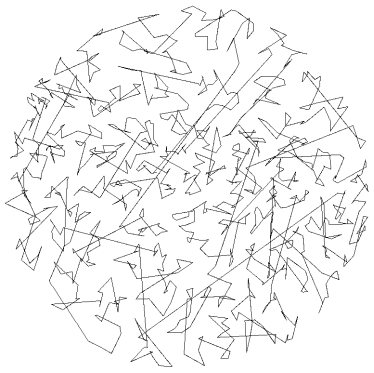
Problem 1. (*Tour Data Type*) Implement a data type `Tour` in `tour.py` that represents the sequence of points visited in a TSP tour. We'll represent the tour as a list of `Point` objects. The first element of the list is the point that the salesperson visits first and the last element is the point that the salesperson visits last before returning to the first point and completing the tour. The data type must support the following API:

method	description
`Tour()`	create an empty tour t
`t.show()`	write the tour t to standard output
`t.draw()`	draw the tour t to standard draw
`t.size()`	the number of points in the tour t
`t.distance()`	the total distance of the tour t
`t.insertNearest(p)`	insert the point p to the tour t using the nearest neighbor heuristic
`t.insertSmallest(p)`	insert the point p to the tour t using the smallest increment heuristic

```
$ python tour.py
Tour 1:~
(200.0, 400.0)
(300.0, 100.0)
(300.0, 200.0)
(100.0, 100.0)
Tour distance = 956.062330
Number of points = 4
Tour 2:~
(200.0, 400.0)
(100.0, 100.0)
(300.0, 100.0)
(300.0, 200.0)
Tour distance = 839.834564
Number of points = 4
```

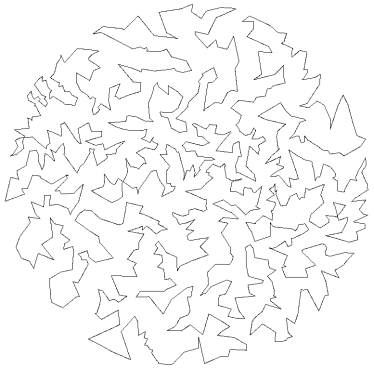
Problem 2. (*Nearest Neighbor Heuristic*) Implement a client program `nearest_neighbor.py` that reads in points from standard input; runs the nearest neighbor heuristic; writes the resulting tour, its distance, and its number of points to standard output; and displays the resulting tour on standard draw for t milliseconds, where t is a command-line argument.

```
$ python nearest_insertion.py 5000 < data/tsp1000.txt
(185.0411, 457.8824)
(198.3921, 464.6812)
(195.8296, 456.6559)
(216.8989, 455.126)
(213.3513, 468.0186)
(241.4387, 467.413)
(259.0682, 473.7961)
(221.5852, 442.8863)
...
(264.57, 410.328)
Tour distance = 27868.710635
Number of points = 1000
```



Problem 3. (*Smallest Insertion Heuristic*) Implement a client program `smallest_insertion.py` that reads in points from standard input; runs the smallest insertion heuristic; writes the resulting tour, its distance, and its number of points to standard output; and displays the resulting tour on standard draw for t milliseconds, where t is a command-line argument.

```
$ python smallest_insertion.py 5000 < data/tsp1000.txt
(185.0411, 457.8824)
(195.8296, 456.6559)
(193.0671, 450.2405)
(200.7237, 426.3461)
(200.5698, 422.6481)
(217.4682, 434.3839)
(223.1549, 439.8027)
(221.5852, 442.8863)
...
(186.8032, 449.9557)
Tour distance = 17265.628155
Number of points = 1000
```



Data Test data files and reference solutions are available under the `data` directory. The input format begins with two integers w and h, followed by pairs of x- and y-coordinates. All x-coordinates are real numbers between 0 and w; all y-coordinates are real numbers between 0 and h. As an example, `tsp1000.txt` contains the following data:

```
$ cat data/tsp1000.txt
775 768
185.0411 457.8824
247.5023 299.4322
701.3532 369.7156
563.2718 442.3282
144.5569 576.4812
535.9311 478.4692
383.8523 458.4757
329.9402 740.9576
...
254.9820 302.2548
```

Files to Submit

1. `tour.py`
2. `nearest_insertion.py`
3. `smallest_insertion.py`
4. `report.txt`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

  ```
  $ python run_tests.py [<problems>]
  ```

  where the optional argument `<problems>` lists the numbers of the problems you want to test; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

  ```
  $ pep8 <program>
  ```

  where `<program>` is the `.py` file whose style you want to check.

- Make sure your report doesn't exceed 400 lines, doesn't contain spelling mistakes, and doesn't contain lines that exceed 80 characters.

Acknowledgements This project is an adaptation of the Traveling Salesperson Problem assignment developed at Princeton University by Robert Sedgewick and Kevin Wayne.