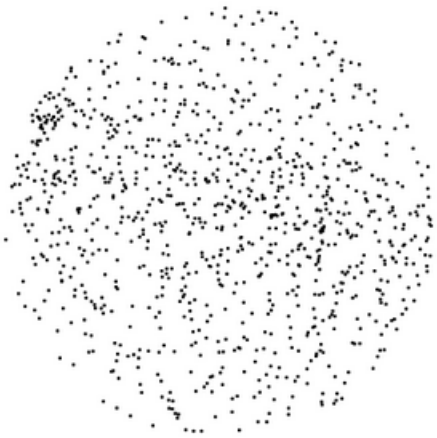
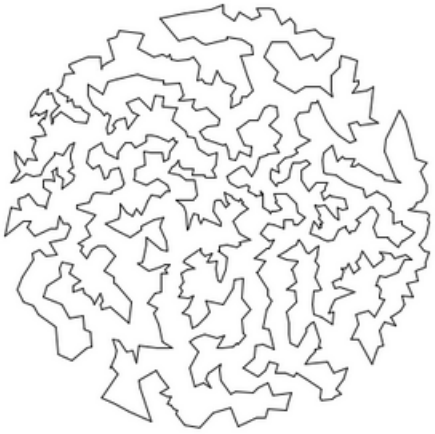
Given N points in the plane, the goal of a traveling salesperson is to visit all of them (and arrive back home) while keeping the total distance traveled as short as possible. The purpose of this project is to implement two greedy heuristics to find good (but not optimal) solutions to the *traveling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

Data Test data files (`*.txt`) and reference solutions (`*.ans`) 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, `tsp10.txt` contains the following data:

```
$ cat data/tsp10.txt
600 600

110.0 225.0
161.0 280.0
325.0 554.0
490.0 285.0
157.0 443.0
283.0 379.0
397.0 566.0
306.0 360.0
343.0 110.0
552.0 199.0
```

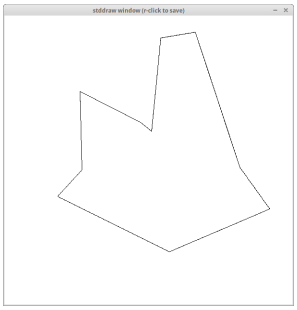
Text and Visual Clients We provide the following text and visual client programs that you may use to test and debug your code:

- `nearest_insertion.py` is a text-based client that reads in points from standard input, runs the *nearest neighbor* heuristic, and prints to standard output the resulting tour along with its distance and the number of points.

```
$ python nearest_insertion.py < data/tsp10.txt
(110.0, 225.0)
(161.0, 280.0)
(157.0, 443.0)
(283.0, 379.0)
(306.0, 360.0)
(325.0, 554.0)
(397.0, 566.0)
(490.0, 285.0)
(552.0, 199.0)
(343.0, 110.0)
Tour distance = 1566.136305
Number of points = 10
```

- `nearest_insertionv.py` is a visual client that reads in points from standard input, runs the *nearest neighbor* heuristic, prints to standard output the distance of the resulting tour along with the number of points, and displays the tour on standard draw.

```
$ python nearest_insertionv.py < data/tsp10.txt
Tour distance = 1566.136305
Number of points = 10
```

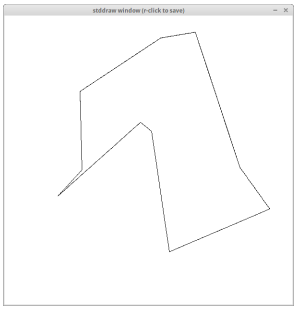


- `smallest_insertion.py` is a text-based client that reads in points from standard input, runs the *smallest increase* heuristic, and prints to standard output the resulting tour along with its distance and the number of points.

```
(110.0, 225.0)
(283.0, 379.0)
(306.0, 360.0)
(343.0, 110.0)
(552.0, 199.0)
(490.0, 285.0)
(397.0, 566.0)
(325.0, 554.0)
(157.0, 443.0)
(161.0, 280.0)
Tour distance = 1655.746186
Number of points = 10
```

- `smallest_insertionv.py` is a visual client that reads in points from standard input, runs the *smallest increase* heuristic, prints to standard output the distance of the resulting tour along with the number of points, and displays the tour on standard draw.

```
$ python smallest_insertionv.py < data/tsp10.txt
Tour distance = 1655.746186
Number of points = 10
```



Files to Submit

1. `tour.py`
2. `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 -v [<problems>]
```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) 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 isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling/grammatical mistakes.

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