CSC263 – Problem Set 1

Remember to write your **full name** and **student number** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

Remember that you are required to submit your problem sets as both LaTeX `.tex` source files and `.pdf` files. There is a 10% penalty on the assignment for failing to submit both the `.tex` and `.pdf`.

Due Jan 23, 2025, 22:00; required files: ps1.pdf, ps1.tex, ps1.py

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct but are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to TWO to complete these questions.

1. [**12 points**] Consider the following algorithm, where `locations` is a list of geographic coordinates, and `radius` specifies the range of interest in kilometers.

```
ProximityCheck(locations, x_q, y_q, radius = 10):
  for (x, y) in locations:
    d = \sqrt{(x - x_q)^2 + (y - y_q)^2}
    if d < radius:
      print("Within range!")
    else:
      print ("Out of range!")
```

Suppose the input list `locations` contains n points distributed uniformly and independently over a square region of side length L. The query point (x_q, y_q) is fixed at the center of the square. Each location is equally likely to fall anywhere within the square.

(a) (1 point) What is the probability that `ProximityCheck` prints "Out of range!" for every point in `locations`? Assume the default value `radius = 10`. Justify your answer carefully: show your work and explain your calculation.

For an individual point uniformly distributed inside a square, the probability that the point is within another shape is the ratio between the area of the shape that overlaps with the square and the area of the square itself. Notice that in the algorithm, "Out of range!" is printed when the distance between the points (x, y) and (x_q, y_q) is greater than or equal to 10, in other words, when (x, y) lies outside the circle centered at (x_q, y_q) with radius 10. We will find the probability that the point lies within the circle, and then compute one minus the probability to find the desired number.

We assume the square entirely contains the circle, meaning that $L \geq 20$, so to find the probability we only need to calculate the ratio between the areas of the circle and the square, which comes out to be $\frac{100\pi}{L^2}$. Thus our answer is

$$1 - \frac{100\pi}{L^2}$$

For n points, the location of each of the points is independent from each other, so the probability that "Out of range!" will be printed for all n points is $\left(1 - \frac{100\pi}{L^2}\right)^n$.

(b) (1 point) What is the probability that `ProximityCheck` prints "Within range!" for all n points? Assume the default value `radius = 10`. Justify your answer carefully: show your work and explain your calculation.

The probability that "Within range!" is printed for any singular point was calculated in the previous part to be $\frac{100\pi}{L^2}$. Using the independence of the location of each point, the probability that all points are within range is $\left(\frac{100\pi}{L^2}\right)^n$.

(c) (2 points) What is the expected number of times "Within range!" is printed for $n = 5$ locations? Assume the default value `radius = 10`. Justify your answer carefully: show your work and explain your calculation.

Let X be the number of times that "Within range!" is printed for $n = 5$. Then since each point is chosen independently from each other, $X \sim Bin\left(5, \frac{100\pi}{L^2}\right)$. Thus

$$E[X] = 5\frac{100\pi}{L^2} = \frac{500\pi}{L^2}$$

(d) (4 points) Consider the following modified algorithm. Please calculate the worst-case and best-case runtime, as well as their probability.

```
  ProximityCheck(locations, x_q, y_q, radius):
    total_distance = 0
    for (x, y) in locations:
      d = \sqrt{(x - x_q)^2 + (y - y_q)^2}
      if d < radius:
        total_distance += d
      elif d >= radius:
        print("Terminating...")
        return total_distance
    return total_distance
```

The worst-case occurs when every point in `locations` is within the circle centered around (x_q, y_q) with radius `radius`, because this causes the loop to iterate through all the elements in `locations`.

The best-case occurs when the first point in `locations` is outside the circle, as the loop will terminate on the very first iteration.

Let n be the number of points in `locations`. Let X be the position of the first point is out of range of the circle. For convenience, $X = n + 1$ represents a list where all its elements are in range. To find the probability of the worse-case occuring, we find $P(X = n + 1)$. Likewise, the probability of the the best-case occuring is $P(X = 1)$. The distribution of X is given by

$$f_X(x) = \begin{cases} \left(\frac{100\pi}{L^2}\right)^{x-1}\left(1 - \frac{100\pi}{L^2}\right), & \text{if } x = 1, ..., n; \\ \left(\frac{100\pi}{L^2}\right)^{n} & \text{if } x = n+1 \\ 0, & \text{otherwise.} \end{cases}$$

Thus $P(X = n+1) = \left(\frac{100\pi}{L^2}\right)^n$ and $P(X = 1) = \left(1 - \frac{100\pi}{L^2}\right)$.

(e) (4 points) What is the expected number of times the for loop runs in the modified algorithm? Show your work and explain your calculation. You don't have to simplify your final answer.

Let X be the number of times the loop runs in the algorithm. The support of X is $1, ..., n$, where n is the size of `locations`.

Note that X is entirely dependent on the position of the first point in `locations` that is outside of the circle. Thus the distribution of X is given by

$$f_X(x) = \begin{cases} \left(\frac{100\pi}{L^2}\right)^{x-1}\left(1 - \frac{100\pi}{L^2}\right), & \text{if } x = 1, ..., n; \\ \left(\frac{100\pi}{L^2}\right)^{n} & \text{if } x = n+1 \\ 0, & \text{otherwise.} \end{cases}$$

Now, we compute the expected value of X:

$$E[X] = \sum_{x=1}^{n+1} x f_X(x) = (n+1)\left(\frac{100\pi}{L^2}\right)^n + \left(1 - \frac{100\pi}{L^2}\right)\sum_{x=1}^{n} x\left(\frac{100\pi}{L^2}\right)^{x-1}$$

Hint: The probability that a single point is within the circular region of radius r centered at (x_q, y_q) is proportional to the ratio of the circle's area to the square's area:

$$p = \frac{\pi r^2}{L^2}.$$

For multiple independent points, probabilities combine according to the binomial distribution.

Programming Question

The best way to learn a data structure or an algorithm is to code it up. In each problem set, we will have a programming exercise for which you will be asked to write some code and submit it. You may also be asked to include a write-up about your code in the PDF/TEXfile that you submit. Make sure to **maintain your academic integrity** carefully, and protect your own work. The code you submit will be checked for plagiarism. It is much better to take the hit on a lower mark than risking much worse consequences by committing an academic offence.

2. **(12 points)** In this question, we will solve a problem that we call **spy263**. The function `spy263` takes a list of commands that operate on the current collection of data. Your task is to process the commands in order and return the required list of results. There are two kinds of commands: `insert` commands and `find_spy` commands.

   An `insert` command is a string of the form `insert x`, where `x` is an integer. (Note the space between `insert` and `x`.) This command adds `x` to the collection.

   A `find_spy` command is simply the string `find_spy`. It retrieves the $\lceil \phi \times n \rceil$-th smallest element in the collection, where $\phi = 0.263$ is a position where a spy chooses to hide, and n is the current size of the collection.

   Your goal is to implement `insert` in $O(\lg n)$ time worst-case, and `find_spy` in $O(1)$ time worst-case. Your algorithm should also have $O(n)$ space complexity. Here, n is the number of elements currently in the collection. The list returned by `spy263` consists of the results, in order, from each `find_spy` command.

   Let's go through an example. Here is a sample call of `spy_263`:

```
spy263(
  ['insert 15',
   'find_spy',
   'insert 6',
   'insert 2',
   'insert 8',
   'find_spy',
   'insert -5'
   'insert -8'
   'insert 3'
   'insert 20'
   'find_spy',
  ])
```

   These commands corresponds to the following steps:

   - The collection begins empty, with no elements.
   - We insert 15. The collection contains just the integer 15.
   - We then have our first `find_spy` command. The result is the $\lceil \phi \times 1 \rceil = \lceil 0.263 \rceil =$ 1st smallest element currently in the collection, which is 15.
   - We insert 6. The collection now contains 15 and 6.
   - We insert 2. The collection now contains 15, 6, and 2.
   - We insert 8. The collection now contains 15, 6, 2, and 8.
   - Now we have our second `find_spy` command. The result is the $\lceil \phi \times 4 \rceil = \lceil 1.052 \rceil =$ 2nd smallest element currently in the collection, which is 6.

- We insert -5. The collection now contains 15, 6, 2, 8, and -5.
- We insert -8. The collection now contains 15, 6, 2, 8, -5, and -8.
- We insert 3. The collection now contains 15, 6, 2, 8, -5, -8 and 3.
- We insert 20. The collection now contains 15, 6, 2, 8, -5, -8, 3 and 20.
- Now we have our third and final `find_spy` command. The result is the $\lceil \phi \times 8 \rceil = \lceil 2.104 \rceil =$ 3rd smallest element currently in the collection, which is 2.

So, the above call `spy263` returns `[15, 6, 2]`, which are the three values produced by the `find_spy` commands.

Requirements:

- Your code must be written in Python 3, and the filename must be `ps1.py`.
- We will grade only the `spy263` function; please do not change its signature in the starter code. Include as many helper functions as you wish.
- All code that you submit must be your own, including any helper functions.
- Your code must compile and otherwise be testable in order to earn credit for the auto-graded portion of this question.
- Your written explanation and runtime analyses must be reasonable to earn points on the coding components of this question, regardless of your autotest result.
- You can earn up to 12 points if you **only use materials from weeks 1-2, and previous data structures covered in CSC148**
- You can earn up to 6 points if you use other materials.
- For each test-case that your code is tested on, your code must run within 10x the time taken by our solution. Otherwise, your code will be considered to have timed out.

Write-up: in your `ps1.pdf/ps1.tex` files, briefly but clearly describe the main ideas behind your algorithms (3 points). Informally argue why your code is correct, and has the desired runtime (3 points). Your code will be tested via unit tests (6 points)—however, **your written explanation and runtime analyses must be reasonable to earn points on the coding components of this question, regardless of your autotest result.**

Hint: It is up to you to decide what data structure or *combination of data structures* you wish to use to store the collection.

The idea behind the algorithm is to store the values inserted into the collection in two heaps: one max heap for all the elements that are less than the $\lceil \phi \times n \rceil$th element and including the the $\lceil \phi \times n \rceil$th element, and one min heap for all the other elements. By doing this, the spy is guaranteed to be at the root of the first heap, and the next element in magnitude is in the root of the min heap. In general, the number of nodes in the first heap is exactly $\lceil \phi \times n \rceil$.

Each time an insert operation is called, we check whether the position of the spy changes. If it does, then the old max heap does not contain the right number of elements anymore and the spy is not necessarily at the root anymore. To fix this, we need to insert a new element into the first heap. Particularly, we need to insert the next least element after the old spy. The possible candidates are the root of the second heap, or the new element to be inserted, so we simply take the lower one. If we take the element from the second heap, we remove it from its original position in the second heap and re-insert it into the first heap. If we take the new element, just insert it into the first heap.

If the position of the spy does not change, the size of the first heap should stay the same. Then all we need to do is check if the new element is ordered before or after the spy. If it is after, there is no problem and we insert the element into the second heap. If it is before, we need to insert it into the first heap. In order to keep the size of the heap the same, we "kick" the spy out of the first heap and insert it into the second heap.

In terms of time complexity, the insersion operation utilizes heap insertion/deletion along with constant time comparisons, so the worse-case runtime ends up being logarithmic. Finding the spy is done in constant time, as we only need to access and return the value at the top of the first heap.