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 January 28, 2019, 22:00; required files: ps1sol.pdf, ps1sol.tex, moving_min.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 that 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 THREE to complete these questions.

1. [**4**] Recall this code from lecture.

```
1 Search42(L):
2   z = L.head
3   while z != None and z.key != 42:
4       z = z.next
5   return z
```

Rather than supposing that each key in the list is an integer chosen uniformly at random from 1 to 100, let's instead suppose that the list length n is at least 42 and that the list keys are a random permutation of $1, 2, 3, \ldots, n$.

Under these new assumptions, what is the expected number of times that line 3 is executed?

Give your answer in **exact form**, i.e., **not** in asymptotic notations. Show your work!

$P(T = 1) = \frac{1}{n}$

$P(T = 2) = \overbrace{(\frac{n-1}{n})}^{\text{a number thats not 42}} \times \underbrace{(\frac{1}{n-1})}_{\text{42, but theres only (n-1) numbers left}} = \frac{1}{n}$

$P(T = 3) = (\frac{\cancel{n-1}}{n})(\frac{\cancel{n-2}}{\cancel{n-1}})(\frac{1}{\cancel{n-2}}) = \frac{1}{n}$

$\vdots$

$P(T = n) = (\frac{\cancel{n-1}}{n})(\frac{\cancel{n-2}}{\cancel{n-1}})(\frac{\cancel{n-3}}{\cancel{n-2}})\ldots(\frac{\cancel{n-(n-1)}}{\cancel{n-(n-2)}})(\frac{1}{\cancel{n-(n-1)}}) = \frac{1}{n}$

Probability is $\frac{1}{n}$ for all of the samplespace.

$E(T) = \sum_{t=1}^{n}(t)P(T = t))$
$= \sum_{t=1}^{n}(t)(\frac{1}{n}) = \frac{1}{n}\sum_{t=1}^{n} t$
$= (\frac{1}{n})(\frac{n(n+1)}{2}) = \frac{n+1}{2}$

2. [**12**] Consider the following algorithm that describes the procedure of a casino game called "`Survive263`". The index of the array A starts at 0. Let n denote the length of A.

```
1    Survive263(A):
2        '''
3        Pre: A is a list of integers, len(A) > 263, and it is generated
4             according to the distribution specified below.
5        '''
6        winnings = -5.00   # the player pays 5 dollars for each play
```

```
7       for i from n-1 downto 0:
8           winnings = winnings + 0.01  # winning 1 cent
9           if A[i] == 263:
10              print("Boom! Game Over.")
11              return winnings
12      print("You survived!")
13      return winnings
```

The input array A is generated in the following specific way: for $A[0]$ we pick an integer from $\{0, 1\}$ uniformly at random; for $A[1]$ we pick an integer from $\{0, 1, 2\}$ uniformly at random; for $A[2]$ we pick an integer from $\{0, 1, 2, 3\}$ uniformly at random, etc. That is, for $A[i]$ we pick an integer from $\{0, \ldots, i + 1\}$ uniformly at random. All choices are independent from each other. Now, let's analyse the player's expected winnings from the game by answering the following questions. All your answers should be in **exact form**, i.e., **not** in asymptotic notations.

(a) Consider the case where the player **loses the most** (i.e., minimum winnings), what is the return value of `Survive263` in this case? What is the probability that this case occurs? Justify your answer carefully: show your work and explain your calculation.

(b) Consider the case where the player **wins the most** (i.e., maximum winnings), what is the return value of `Survive263` in this case? What is the probability that this case occurs? Justify your answer carefully: show your work and explain your calculation.

(c) Now consider the **average case**, what is the **expected value** of the winnings of a player (i.e., the expected return value of `Survive263`) according to the input distribution specified above? Justify your answer carefully: show your work and explain your calculation.

(d) Suppose that you are the owner of the casino and that you want to determine a length of the input list A so that the expected winnings of a player is between -1.01 and -0.99 dollars (so that the casino is expected to make about 1 dollar from each play). What value could be picked for the length of A? You are allowed to use math tools such as a calculator or WolframAlpha to get your answer.

(a) They lose the most if at A[262] they get 263. Len(A) = 264. (261)(0.01) = 2.61 Therefore the function has to run 261 times minumum.

$P(263 \text{ at } A[262]) = \frac{1}{265}$

The return value is -2.39

(b) 263 appears as the last element at A[n-1] on a list of size n.

returns 0.1n-5

$P(\text{this happens}) = (1 - \frac{1}{n})^{n-1} \cdot (\frac{1}{n})$

$= (\frac{n-1}{n})^{n-1} \cdot (\frac{1}{n}) = (\frac{n-1}{n})^{n}(\frac{n-1}{n})^{1}(\frac{1}{n})$

$= \frac{(n-1)^{n-1}}{(n)^{n}}$

$= (\frac{n-1}{n})^{n}(\frac{1}{n-1})$

(c) Probability we get no 236 from list position 262 to n-1 :

$(\frac{262}{\cancel{263}})(\frac{\cancel{263}}{\cancel{264}})(\frac{\cancel{264}}{\cancel{265}}) \ldots (\frac{\cancel{n-2}}{\cancel{n-1}})(\frac{\cancel{n-1}}{n}) = (\frac{262}{n})$

So the probability we get atleast one 263 is 1-$(\frac{262}{n})$

In the case we get atleast one 263;we need the expected value of position of first 263 from right to left.

If we get a 263 on our first position(right to left):

$P(X = 1) = (\frac{1}{n})$

$P(X = 2) = (\frac{\cancel{n-1}}{n})(\frac{1}{\cancel{n-1}}) = \frac{1}{n}$

$\vdots$

$P(X = n - 262) = (\frac{\cancel{n-1}}{n})(\frac{\cancel{n-2}}{\cancel{n-1}}) \ldots (\frac{\cancel{263}}{\cancel{264}})(\frac{1}{\cancel{263}}) = \frac{1}{n}$

$E(X) = \sum_{x=1}^{n-262}(x)(\frac{1}{n})$

$= \frac{1}{n}\sum_{x=1}^{n-262}(x)$

$= (\frac{1}{n})(\frac{(n-262)(n-261)}{2})$
$= \frac{(n-262)(n-261)}{2n}$

Putting this all together:

$E(W) = 0.01n \cdot (\frac{262}{n}) + 0.01 \cdot (\frac{(n-262)(n-261)}{2n}) - 5$
$= 2.62 + 0.01 \cdot (\frac{(n-262)(n-261)}{2n}) - 5$
$= -2.38 + (\frac{(n-262)(n-261)}{200n})$
$= \frac{(n-262)(n-261)}{2n}$

(D) Finding a value of n for which the expected value is between -0.99 and -1:

$-2.38 + (\frac{(n-262)(n-261)}{200n}) = -1$
$\frac{(n-262)(n-261)}{200n} = 1.38$
$(n-262)(n-261) = 1.38 \cdot 200n$
$n^2 - 261n - 262n + 68382 - 276n = 0$
$n^2 - 799n + 68382 = 0$
$n = \frac{799}{2} + \frac{\sqrt{364873}}{2}$
$n \approx 701.52$
if $n = 702$:
$-2.38 + \frac{440 \times 441}{140400} = -09979487179$
This value for n works, so we can use n = 702

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.

3. **[12]** In this question, you will solve the **Moving Minimum Problem**. The function `solve_moving_min` 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 `get_min` 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 `get_min` command is simply the string `get_min`. The first `get_min` command results in the smallest element currently in the collection; the next `get_min` command results in the second-smallest element currently in the collection; and so on. That is, the `j`th `get_min` command results in the `j`th-smallest element in the collection at the time of the command. You can assume that the collection has at least `j` elements at the time of the `j`th `get_min` command.

   Your goal is to implement `insert` and `get_min` each in $O(\lg n)$ time, where n is the number of elements currently in the collection. The list returned by `solve_moving_min` consists of the results, in order, from each `get_min` command.

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

```
solve_moving_min(
  ['insert 10',
   'get_min',
   'insert 5',
   'insert 2',
   'insert 50',
   'get_min',
   'get_min',
   'insert -5'
```

```
])
```

This corresponds to the following steps:

- The collection begins empty, with no elements.
- We insert 10. The collection contains just the integer 10.
- We then have our first `get_min` command. The result is the smallest element currently in the collection, which is 10.
- We insert 5. The collection now contains 10 and 5.
- We insert 2. The collection now contains 10, 5, and 2.
- We insert 50. The collection now contains 10, 5, 2, and 50.
- Now we have our second `get_min` command. The result is the second-smallest element currently in the collection, which is 5.
- Now we have our third `get_min` command. The result is the third-smallest element currently in the collection, which is 10.
- We insert -5. The collection now contains 10, 5, 2, 50, and -5.

`solve_moving_min` returns `[10, 5, 10]` (the three values produced by the `get_min` commands).

Requirements:

- Your code must be written in Python 3, and the filename must be `moving_min.py`.
- We will grade only the `solve_moving_min` function; please do not change its signature in the starter code. include as many helper functions as you wish.

Write-up: in your `ps1sol.pdf/ps1sol.tex` files, briefly and informally argue why your code is correct, and has the desired runtime.

```
  Brief arguement on why solve_moving_min is correct and insert and get_min is O(lgn).

           solving_moving_min uses a min-Heap, the first thing it does is create a heap, h = Heap().
In the commands list, when it says to insert, solving_moving_min calls h.insert(x),
where x is the number to insert.

           The Heap creates a list and when insert is called, if the list is empty it appends the
 element into the empty list. O(1)

           If the list is not empty, the Heap appends the element to the end of the list,
then calls the bubble_up function with parameter, size of the list.
The bubble up function takes i (the index where the new element just got appended)
and while i//2 is not 0 it checks if the node at position i is smaller than its parent node at i//2,
 if it is swap them. i divides by two.

           The reason why the while loop is i//2 >0 is because if i is 0 then
we are already at the root node and can't go back any furthar, otherwise go back to i's
 parent (this is what the while loop and i=i//2 do).

           Index i traverses through the tree going from child to parent, from the bottom most leaf
to the root. So it traverses the height of the tree, since the Heap is a balanced binary tree,
 we are traversing logn steps. So, O(lgn)

get_min is called everytime it appears in the command list. It returns the smallest i'th node in the heap.
In get_min if it the first smallest node it simply returns the root of the heap. O(1)

           Otherwise, it checks which level (height) of the tree the i'th smallest node will appear.
```

It does this by checking if i > 2^(count+1)-1 where count is a counter (from 0) and if that condition is true count is increased by one. since 2^(count+1)-1 gives the number of nodes in a balanced binary tree where count is the level if i <= 2^(count+1)-1 then i is at that level. This takes lgn steps as it goes through the height which is lgn (n is nodes).

Next it stores that level of the tree in a list and sorts it which is O(nlgn), but since it occurs only on s nodes where s is the number of nodes in that level:

this takes O(lgn) time. it returns the node at the appropriate index, returning i'th smallest.

So in total get_min takes lgn+lgn time which is O(lgn).