CSC263 – Problem Set 1

Si Tong Liu(1004339628), Jing Huang(1003490705), Yifei Gao(1004152640)

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!

Solution:

According to the formula: $E[t_u] = \sum_t^n tPr(t_n = t)$, we need to find out the probability that the ith key of the given list is 42, where i >= 1:

(1) $P(t_n = 1) = \frac{1}{n}$ (where $n \geq 42$, and the first key is 42)

(2) $P(t_n = 2) = \frac{1(n-1)!}{n!} = \frac{1}{n}$(where $n \geq 42$,the second key is 42)

(3) $P(t_n = 3) = \frac{(n-1)1(n-2)!}{n!} = \frac{1}{n}$ (the third key is 42)

...

(5) $P(t_n = n) = 1/$n(the last key of the given list is 42)

so we can conclude that $P(t_n = t) = = \frac{1}{n}$, where $n \geq 42$ and $1 \leq t \leq n$

therefore $E[t_u] = \sum_{t=1}^n t\frac{1}{n} = \frac{1}{n} \frac{(n+1)n}{2} = \frac{1+n}{2} = 22$ (since $n \geq 42$)

In conclusion, the expected number of times that line 3 will be executed is at least 22.

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.

Solution:

(a) Since A[i] = $\{0, \ldots, 263\}$ and $len(A) > 263$, the smallest length of list that we can get is n = 264 => i = 263, and A[263] = $\{0, \ldots, 264\}$.Then the minimum winning = -5.00 + 0.01 = -4.99 (when A[i]=A[263]=263, and you played the game only once)

Because we can choose 263 when index is 262, so the probability that we can choose 263 at index 263 is that: Pr[A[263] = 263] = $\frac{1}{264}$ (need to make sure that we do not choose 263 at index 262)

(b) Assume the maximum length of the given list is n, where $n > 263$. To get the maximum winnings, we need to survive the game. Then we can get the maximum winnings is = -5.00 + $\sum_{n=0}^{n-1} 0.01 = -5.00 + \frac{n}{100}$ (since by line 8 in the given code, every time we go through an element in the given list, the winning will increase 0.01)

In order to get the maximum winnings, we need to make sure that we do not have A[i] = 263, where $262 \leq i \leq n-1$(since start from 262, we can have chance to choose 263 A[262] = $\{0, \ldots, 263\}$). So Pr(no 263 in the given list) = Pr(no 263 in the given list from index 262 to n-1) = ($\frac{262}{263}$x$\frac{263}{264}$x...$\frac{n}{n+1}$) = $\frac{262}{n+1}$

(c) $E[t_n = t] = \sum_{t=0}^{n-1} tPr(t_n = t) =$

Firstly, find $Pr(t_n = t)$, where $1 \leq t \leq n$

(1) $Pr(t_n = 1) = \frac{1}{n+1}$ (A[n-1]=263)

(2) $Pr(t_n = 2) = \frac{1}{n-1}\frac{n}{n+1} = \frac{1}{n+1}$ (A[n-2]=263)

(3) $Pr(t_n = 3) = \frac{1}{n-2}\frac{n-2}{n-1}\frac{n}{n+1} = \frac{1}{n+1}$ (A[n-3]=263)

...

(4) $Pr(t_n = n-262) = \frac{1}{263}\frac{263}{264}\cdots\frac{n}{n+1} = \frac{1}{n+1}$ ((A[262]=263))

(5) $Pr(t_n = n-261) = \frac{262}{263}\frac{263}{264}\cdots\frac{n}{n+1} = \frac{262}{n+1}$ (A[261]=263)

...

(4) $Pr(t_n = n) = \frac{262}{263}\frac{263}{264}\cdots\frac{n}{n+1} = \frac{262}{n+1}$ (A[0]=263)

Therefore, we can conclude that: $Pr(t_n = t) =$

$$\begin{cases} 1/(n+1), 1 <= t <= n-262, \\ 262/(n+1), n-261 <= t <= n, \end{cases} \tag{1}$$

Therefore $E[t_u] = \sum_t^n tPr(t_n = t)$

$= \sum_{t=1}^{n-262} t\frac{1}{n+1} + \sum_{t=n-261}^{n} t\frac{262}{n+1}$

$= \frac{(n-262)(n-261)}{2(n+1)} + \frac{262[131(2n-261)]}{n+1}$

$\leq \frac{n(n-484)}{n+1} \geq 485$ (to make n-484 > 0)

So the expected value of the winnings is: -5 + 485*0.01 = -0.15

(d) - 1 = -5 + winnings $\rightarrow winnings = 4 \rightarrow playedtimes = 4/0.01 = 400 \rightarrow n = 884.45 = 885(length \geq 0)$

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.

Solution: for the function we designed. We make `insert` and `get_min` functions similar to binary search. So the runtime of them are both: $T(n) = 2T(n/2) + \theta(n)$

By using Master Theorem, we can conclude that T(n) = lg(n) for both `insert` and `get_min` function