CSC263 Problem Set # 1

By: Arjun Dhiman, Ryan Persaud, Nick Gong
Date: Friday, Sept, 23rd, 2016

Solution to Question 1.a

Proof.

Case 1: If the element x, is what we are searching for does not exist in the list A. Then the algorithm will take n + 1 steps, an extra one because the algorithm needs to make an extra comparison to make sure element is actually not present.

Case 2: For every n, and **EVERY** input size of n, the following statements are true:

1. The for loop and conditional statement is executed at most n times.
2. Each iteration takes β steps for some contact $\beta \in \mathbb{R}$
3. d steps are taken outside of the loop, for some constant d $\in \mathbb{R}$

So, to conclude. For all inputs of size n, the time needed for the entire algorithm is at most $\underline{\beta n + d}$.

So the worst-case run time complexity of the algorithm T(n) is $\mathcal{O}(n)$. ∎

Solution to Question 1.b

Proof.

Suppose we have the following input list $A[x] = \{x, \cdots, x_i\}$. And we choose some element x to search for. Now, in order to show the lower bound we need to show that we have a bad input. There are two bad input cases for this algorithm.

Case 1: If $x \notin A$, in this particular case **search** take n steps to traverse through the list, to search for x, and an extra step to verify that it indeed in not in A.

Case 2: If $x \in A$. Then for every n, and **EVERY** input size of n, the following statements are true:

1. The for loop and conditional statement is executed at most n times.
2. Each iteration takes γ steps for some contact $\gamma \in \mathbb{R}$.
3. d steps are taken outside of the loop, for some constant d $\in \mathbb{R}$.

So, to conclude. For all inputs of size n, the time needed for the entire algorithm is at least $\gamma n + d$.

So the worst-case run time complexity of the algorithm T(n) is $\Omega(n)$. ■

Solution to Question 1.c

Proof.

1. Step #1: Input distribution for A and x are chosen uniformly random and independently from 1 - 263 inclusive, uniformly random and independent.
2. Step #2: Counting items visited.
3. Step #3: Let T be the random variable counting items visited.
4. Step #4:

   $\mathbf{E}[T] = \sum_{t=1}^{n} tPr(T = t)$

   $\mathbf{E}[T] = \sum_{t=1}^{n} t(\frac{262}{263})^{t-1}(\frac{1}{263})$

   $= 263 - (\frac{262}{263})^n(n + 263)$

   $\lim_{x\to\infty}(\frac{262}{263})^n(n + 263) = 0$, the expected value evaluates to 263.

   Conclusion: The average case analysis shows that the algorithm runs in constant time $\Theta(1)$ on the input space defined in step 1.

■

Solution to Question 1.d

Proof.

1. Step #1: Input distribution for A is chosen uniformly random and independently from 1 - 263 inclusive, uniformly random and independent. Input distribution for x is chosen uniformly random and independently from 1 - 373 inclusive, uniformly random and independent.
2. Step #2: Counting items visited.
3. Step #3: Let T be a random variable counting items visited.
4. Step #4:

   $\mathbf{E}[T] = (2t)(Pr[T = t])$

$= \sum_{t=1}^{n} t(\frac{262}{263})^{t-1} + \sum_{t=1}^{n} t(\frac{263}{373})^{t-1}(\frac{110}{373})$

$= 263 - (\frac{262}{263})^n(n + 263) + \frac{373}{110} - (\frac{263}{373})^n(n + \frac{373}{110})$

$= 263 + \frac{373}{110} - (\frac{262}{263})^n(n + 263) - (\frac{263}{373})^n(n + \frac{373}{110})$

$= \lim_{x\to\infty} -(\frac{262}{263})^n(n + 263) - (\frac{263}{373})^n(n + \frac{373}{110}) = 0$, the expected value evaluates to 263

Conclusion: The average case analysis shows that the algorithm runs in constant time $\Theta(1)$ on the input space defined in step 1.

■

Solution to Question 2.a

Proof.

Case 1: If n is the size of the list A, and $\sqrt{n}$ tell us the range of numbers and the maximum number that A will have. And all elements in A are not distinct and repeat with the highest number $\sqrt{n}$ then there will be a 100% probability of getting the maximum value.

Case 2: If n is the size of the list A, and $\sqrt{n}$ tell us the range of numbers that A will have. And all elements in A are not distinct and repeat with some other number other than the highest number $\sqrt{n}$ then there will be a 0% probability of getting the maximum value.

Case 3: If $\sqrt{n} \notin A$, and all elements in A are distinct there is a

P(getting the probability that this algorithm returns the correct answer (in terms of n)) $= 1 - \frac{(\sqrt{n}-1)^n}{(\sqrt{n})^n}$. Where $(\sqrt{n}-1)^n$ is the number of failure of getting the maximum number, and $(\sqrt{n})^n$ is number of items in A. Then we subtract by 1 to get the success of get the probability getting the maximum number in the list A.

■

Solution to Question 2.b

Proof.

As per definition in 2.a and 2.b

$P_k = Pr[A[0] = k$ and the other elements $\leq k]$
$= ((\frac{1}{\sqrt{n}}) + (1 - \frac{1}{\sqrt{n}})) + ((\frac{1}{\sqrt{n}}) + (1 - \frac{1}{\sqrt{n}}))\cdots$

We add this n times, then it becomes $(\frac{1}{\sqrt(n)}+1-\frac{1}{\sqrt(n)})^n$, which then becomes 1^n

Now, we can use the identity:

$\sum_{k=1}^{d} k^n = \Theta(d^{n+1})$, we know k = 1 exclusively from the above calculations, meaning the summation only sums over 1, we can then conclude that d = 1. Putting them into the identity:

$\sum_{k=1}^{1} 1^n = \Theta(1^{n+1}) = 1$ regardless of n. ■