CSC263 Assignment 1

Akhil Gupta, 1000357071

September 2015

1 Question 1

(a) In the best case, for input (A, v), how many times is Line #2 ("if $A[i] = v$") executed? Justify your answer.
Answer: The best-case is when v occurs at $i = 1$ i.e. $A[1] = v$. In this case, Line #2 is executed exactly once.

(b) What is the probability that the best case occurs? Justify carefully: show your work and explain your calculation.
Answer: The probability that v occurs at the very first position in array i.e. $A[1] = v$. Since, $A[1]$ is randomly chosen from $\{0, 1, \ldots, n\}$ $(n + 1$ elements) with uniform distribution. Therefore, the probability that Line #2 ("if $A[i] = v$") is executed once is $1/(n + 1)$.

(c) In the worst case, for input (A, v), how many times is Line #2 executed? Justify your answer.
Answer: The worst-case is when v does not occur at all in A. In this case, Line #2 is executed n times because i goes from $\{1, \ldots, n\}$.

(d) What is the probability that the worst case occurs? Justify carefully: show your work and explain your calculation.
Answer: The probability that the worst-case occurs is when v does not exist in A. For each element in A, the probability of v not existing is different. For example: the probability that v does not exist in $A[1]$: Since $A[1]$ has $n + 1$ elements ($\{0, 1, \ldots, n\}$), the probability of v existing is $1/(n + 1)$, therefore the probability of v not existing is $1 - (1/(n + 1)) = n/(n + 1)$. Similarly, for $A[2]$, it has n elements ($\{0, 1, \ldots, n - 1\}$), so the probability of v existing is $1/n$, therefore the probability of v not existing is $1 - (1/n) = (n - 1)/n$. In general, the probability that v does not exist in $A[i] = (n - i + 1)/(n - i + 2)$ where $1 \leq i \leq n$. Since all choices are independent from each other, we can take the product of probabilities that v does not exist in A. $P(v \text{ not in } A) = \prod_{i=1}^{n} \frac{n-i+1}{n-i+2} = \frac{n}{n-1}$. Since the worst-case occurs in 2 possible ways: first when v does not exist in A and second when $i = n$. The probability that v does not exist in $A = n/(n - 1)$ and the probability when $i = n$ is also $n/(n - 1)$.

(e) In the average case, for input (A, v), how many times is Line #2 expected to be executed? Justify your answer carefully: show your work and explain your calculation.
Answer: Let $X = \#$ of times Line 2 ("if $A[i] = v$") is executed. The probability distribution of X has 3 different cases. 1. when v is found, 2. when v is not found and 3. 0 otherwise.
Hence, the probability distribution of X is:

$$Pr(X) = \begin{cases} \frac{1}{n-i+2} & v \text{ is found} \\ \frac{n}{n-1} & v \text{ is not found} \\ 0 & \text{otherwise} \end{cases}$$

2 Question 2

1. The input to this problem consists of k sorted lists $L_1, \ldots, L_k$, each one containing a list of n/k integers in non-descending order. We want to output a single list L that contains all n integers in $L_1, \ldots, L_k$, sorted in non-descending order. Devise an algorithm for solving the above problem with worst-case time complexity $(n \log k)$. You should use a heap that can hold at most k elements. Give a detailed description of your algorithm and explain why it works correctly and has the desired worst-case runtime.

Answer: In order to solve this problem, we need to merge k sorted lists into a single list L. We can start by merging 2 sorted lists into 1 (i.e. mergesort). How 2-way mergesort works: We are given 2 lists. Each list has its own pointer. We create a third list which will contain the sorted list from 1 and 2. We start by comparing the first value in list 1 to the first value in list 2 (with pointers). Then, we append the smaller value to the third list and increment the pointers. We do this until the pointers reach the end of their lists. At the end, we are left with a sorted merged list. In order to merge k-sorted arrays, we use the same procedure but instead of comparing 2 lists, we need to find the smallest value from k lists. In order to accomplish this, we need to use a min-heap with k elements, and 2 functions: ExtractMin(Q) and Insert(Q,k). Since the lists $L_1, \ldots, L_k$ are each sorted in non-descending order, the first step of our algorithm is to put the first element of each list in $L_1, \ldots, L_k$ in the min-heap. The algorithm will ExtractMin(Q) = min from the heap and put it in our final list $L[i]$. Then the algorithm will take the next element of the list and insert(Q, k). And since each list has n/k elements, and there are k lists, it takes n/k*k*log$k$ = $n$log$k$ steps to Build-Min-Heap, ExtractMin(Q), Insert(Q,k) and populate L. Therefore the worst-case runtime is O($n$log$k$) and our final list L contains all n integers in $L_1, \ldots, L_k$ sorted in non-descending order.

2. Let a, b, c, d be four integers whose values are between 1 and n, inclusive. Devise an algorithm that finds all solutions to the following equation

$$a^7 + b^7 = c^7 + d^7$$

Your algorithm should have worst-case runtime $(n^2 \log n)$. You should use a heap of size at most n. Give a detailed description of your algorithm, and argue why your algorithm works correctly and has the desired worst-case runtime.
Answer: In order to solve this problem, we create a matrix T such that $T[a][b] = a^7 + b^7$. So, every line and row in the matrix is sorted in a non-descending order. Let's create a min-heap (H) and put the smallest element in it. Next, we add the right adjacent element from T if no other element exists in that row. Then, we add the down adjacent element from T if no other element exists in that column. Hence, every time the min-heap inserts a duplicate value, we have found a solution. Inserting and sorting elements in the min-heap takes O(log$n$) time and since we have a matrix of size $(a * b)$ i.e. we have n^2 elements, therefore, the worst-case runtime of the algorithm is O(n^2log$n$).