Com S 311, Mid Term Exam Solutions

1. This question has two parts.

   (a) Please give short answers to each of the questions. Do not write any proofs/details.

   - What is the worst-case time complexity of Selection Sort? (1 point)
     $O(n^2)$
   - What is the best-case time complexity of Quick Sort? (1 point)
     $O(n \log n)$
   - Let $G = (V, E)$ be a directed graph with n nodes and m vertices. Let $d_{in}(v)$ be the in-degree of vertex v and $d_{out}(v)$ be the out-degree vertex v. What is the relation between $\sum_{v \in V} d_{in}(v)$ and $\sum_{v \in V} d_{out}(v)$. (1 Point)
     They are equal
   - Let $G = (V, E)$ be an undirected graph with n vertices and m edges. Suppose that the vertex set V is a set of strings (of length k). What is the worst-case time complexity of performing BFS on G? State the time bound. (2 Points)
     $O(kn \log n)$
   - Given a set S of integers, $succ(x)$ is the smallest number in S that is larger than x. If no such number exists, then $succ(x)$ is defined as ∞. Suppose you stored set S in a hash table. What is the time taken to compute $succ(x)$? (2 Points)
     $O(n)$, *where* n *is number of elements of* S.
   - Suppose you stored S in a Balanced Search Tree (such as AVL tree). What is the time take to compute $succ(x)$? (1 Point)
     $O(\log n)$.

   (a) The following questions are on Big-O. Assume that f and g are two functions from natural numbers to natural numbers. (12 Points)

   - Prove that $26n^2 + 24 \in O(n^2)$
     $26n^2 + 24 \leq 26n^2 + 24n^2 = 50n^2$. Take $c = 1/50$, we have $c(26n^2 + 24) \leq n^2$.
   - Prove that $2^{n+\log n} \in O(n^2 2^{n-3})$
     $2^{n+\log n} = n2^n$ and $n^2 2^{n-2} = \frac{n^2 2^n}{8}$.
     Take $c = 1/8$ we have $\frac{n2^n}{8} \leq \frac{n^2 2^n}{8}$.
   - Suppose that $\log f \in O(\log g)$. Is it the case that $f \in O(g)$? If your answer is *true* provide a proof. If your answer is *false*, give examples of f and g such that $\log f \in O(\log g)$, yet $f \notin O(g)$.
     No. Let $f = 2^{2^{n+1}}$ and $g = 2^{2^n}$. Now, $\log f = 2^{n+1}$ and $\log g = 2^n$. Since $2^{n+1} = 22^n$, we have $\log f \in \log g$; however we know that $f \notin O(g)$ (HW problem).

2. Consider the following code fragment:

```
int i = 1;
int s = 1;
while (i <= n) {
  s = s+ 2i+1;
  i = i+1;
}
```

Prove that $s = i^2$ is a loop-invariant of the above loop. (20 Points)

Ans. Base Case: Before the first iteration of the loop, we have $s = 1$ and $i = 1$, thus $s = i^2$.

Let s_k and i_k denote the values of s and i before kth iteration.

Induction Hypothesis: $s_k = i_k^2$.

Goal: We need to prove that $s_{k+1} = i^2k + 1$. At the beginnig of kth iteration, we have $s_k = i^2 + k$. During kth iteration value of s and i change. $s_{k+1} = s_k + 2i_k + 1$ and $i_{k+1} = i_k + 1$. By induction hypothesis, $s_{k+1} = i_k^2 + 2i_k + 1$ which equals $(i_k + 1)^2 = i_{k+1}^2$.

3. This question is on BST and has two parts.

   (a) Consider the following code that performs *in-order traversal* of a BST. Here x refers to a node of a tree, $x.left$ refers to left pointer, $x.right$ refers to right pointer and $x.value$ refers to the value stored in the node.

```
In-Order(Node x)  {
    if (x == null) return null
       else  {
          In-Order(x.left);
          Print (x.value);
          In-Order(x.right)
        }
}
```

   The above algorithm runs in $O(n)$ time, where n is number of nodes in the tree. Suppose you perform an in-order traversal on the following tree, what is the output? (5 Points)

   (b) Let A and B be two disjoint sets of integers. Let T_A and T_B are two BST's that store elements of A and B respectively, further assume that the heights of both the trees are $O(\log n)$. Give an algorithm that gets T_A and T_B as inputs and constructs a BST of height $O(\log n)$ that stores elements of $A \cup B$. Describe the pseudo code of your algorithm and derive the time bound. Your grade partly depends on the run time of your algorithm. It is possible to do this in $O(n)$ time. (15 Points)

   Ans. Consider the following algorithm.

i. Input T_A and T_B.

ii. Perform in-order traversal of T_A and T_B to obtain sorted arrays S_A and S_B.

iii. Merge(S_A, S_B) to obtain sorted array T.

iv. Build $O(\log n)$ depth BST using T (as in HW):

- Consider the following recursive method. This method attempts to create a BST of $A[left], A[left+1] \cdots A[right]$ with $Node$ as root.,

```
CreateTree(A, left, right, Node)
   Node.value = A[(left+right)/2];//Median Element
   CreateTree(A, left, (left+right)/2-1, Node.left);
   CreateTree(A, (left+right)/2+1, right, Node.right);
```

Algorithm to merge two sorted arrays is give below:

```
Input: Sorted Arrays A and B of length $n$.
Left  = 1; right =1;
Create Array S of size 2n
index = 1;

While(left <=n AND right <= n) {
   If A[left] <= B[right]
       S[index] = A[left]; index++; left++;
  else
      S[index] = A[right]; index++; right++;
}

Copy the remaining element of A or B into S in the order they apppear.
```

Time Analysis: In-order takes $O(n)$ time; Merge takes $O(n)$ Time. Let $T(n)$ be the time taken to construct the BST given a sorted array of size n as input. We have We have the following recurrence.

$$t(n) = 2t(n/2) + 1$$

$$\begin{aligned} t(n) &= 2t(n/2) + 1 \\ &= 4t(n/4) + 2 + 1 \\ &= 8t(n/4) + 4 + 2 + 1 \\ &= \cdot \\ &= \cdot \\ &= 2^k t(n/2^k) + 2^{k-1} + \cdots + 4 + 2 + 1 \\ &= 1 + 2 + 3 + \cdots n \\ &= O(n) \end{aligned}$$

Thus the total time take by the above algorithm is $O(n)$.

4. Given a String A and and integer $k > 0$, A_k is the set of all k-length substrings of A. For example, if A is "mississippi" A_3 contains following strings: `mis, iss, ssi, sis, sip, ipp, ppi`.

   Give an algorithm that gets two strings (of same length n) A, B and an integer $k > 0$ as input and outputs $A_k \cap B_k$. Your algorithm must use hash functions and the idea of rollover hashing; otherwise you will receive zero credit. Describe pseudo code of your algorithm, derive the (expected/average) run-time, and explain the correctness of the algorithm. Note that your run time depends both on n (length of the strings) and k.

   Note that since your algorithm uses hashing, the run time will be "expected/average" (as opposed to worst-case). Use the following function to hash strings to integers.

   $$h(x_1x_2x_3\cdots x_k) = x_k + x_{k-1}\alpha + \cdots x_1\alpha^{k-1}$$

   Your grade depends on efficiency, correctness, and run-time analysis. Finally, your solution must give all details of rollover hashing (you cannot simply say "use rollover hash" without providing details). (20 points)

   Ans. Create a hash table T to store strings. The algorithm works as follows:

   - For every k-length substring s of A, compute $h(s)$ and use it to place s into the hash table T.
   - For every k-length substring t of B, compute $h(t)$ and use it search for t in T; if t is in hash table output t.

   We can compute $h(s)$ for every k-length substring of A (similarly $h(t)$ for every k-length substring of B) using the following algorithm that uses the idea of roll-over hashing. Let $A = x_1x_1\cdots x_n$.

   - Compute $h_1 = h(x_1x_2\cdots x_k) = x_k + x_{k-1}\alpha + \cdots + x_1\alpha^{k-1}$.
   - For i in range 2 to $n-k+1$ compute h_i from h_{i-1} as below:
     - $h_i = (h_{i-1} - x_{i-1}\alpha^{k-1}) \times \alpha + x_{i+k-1}$

   Time take to compute h_1 is $O(k)$; Time taken to compute h_i from h_{i-1} is $O(1)$. Thus time taken to compute $h(s)$ for every k-length substring of A is $O(n+k)$. We know expected time to add/search in hash table is $O(1)$; thus the expected time taken by the above algorithm is $O(n+k)$.

5. Let A be an array of n elements. The **only** operation that can be performed on the array elements is to check if two elements of the array are equal or not. More specifically, you can not perform other comparisons such as $:<, \leq, >, \geq$. Thus **it is not possible to sort the array**. The array has a *majority element* if there is an element appears at least $\lceil \frac{n}{2} \rceil$ times in the array. Write an algorithm that gets A as

input and checks if A has a majority element or not. If a majority element exists, then the algorithm must output it. Your grade depends on the run time of your algorithm. It is possible to do this in $O(n \log n)$. Describe pseudo code of your algorithm, derive the run-time, and explain the correctness of the algorithm. You may assume that n is a power of two (if it helps). If the array has two majority elements, your algorithm may return one of them or both. *Hint: Divide and Conquer for* $O(n \log n)$. (20 Points)

Ans. Consider the following recursive procedure that compute Majority element of an array (if exists).

Procedure Majority(A)

- If A has only element return that element; If A has two elements return both the elements if they are distinct; otherwise return the majority element;
- Let C the first $n/2$ elements of A and D be the last $n/2$ elements of A.
- $\{m_1, m_2\} = Majority(C)$ and $\{m_3, m_4\} = Majority(D)$.
- For each of m_1, m_2, m_3 and m_4 count number of times they appear in A. Return all elements that appear at least $n/2$ times.

Let $t(n)$ be the time taken by the above algorithm. Note that time taken for second and fourth steps is $O(n)$. Thus we have following recurrence relation;

$$t(n) = 2t(n/2) + n$$

$$\begin{aligned} t(n) &= 2t(n/2) + n \\ &= 4t(n/4) + 2n \\ &= 4t(n/8) + 4n \\ &= \cdot \\ &= \cdot \\ &= 2^{\ell} t(n/2^{\ell}) + \ell n \end{aligned}$$

When $\ell = \log n$, we have $t(n/2^{\ell}) = t(1) = 1$. Thus

$$t(n) = 2^{\log n} t(1) = n \log n \in O(n \log n)$$