CMPE 300 ANALYSIS OF ALGORITHMS
MIDTERM ANSWERS

1.

- Knowing the execution time on a particular computer is not useful information. In this case, we also need to know the type of the computer, whether it is single or multiple users machine, how fast its clock is, how well the compiler optimizes the code, etc.
  We are primarily concerned with comparing different algorithms for a particular problem. In this respect, execution time is not a good measure for comparison.
  The type of the computer on which an algorithm executes is irrelevant. It does not change the nature of the algorithm. So, we must analyze an algorithm independent of the computer.

  Therefore, the performance should be measured in terms of number of operations, rather than a clock measure.
- The time complexity of an algorithm is different for different data sizes. Therefore, the performance of an algorithm should be measured as a function of input size.
- It was indicated above that the complexity is calculated by counting the number of times operations are executed. But it does not make sense taking into account all types of operations. An algorithm may have lots of different operations, its interior may be complex, and all the operations may not have the same importance in terms of execution speed. This is both a difficult and an unnecessary approach.

  Therefore, the performance should be measured in terms of a basic operation, rather than all types of operations. The basic operation is the most important operation that contributes the most to the total execution time.
- The time complexity of an algorithm depends not only on the input size, but also on the specifics of a particular input. So, we cannot usually arrive at a single complexity figure for all inputs.

  Therefore, the performance should be measured for some typical cases. These are best-case, worst-case, and average case analyses.
- Expressing the complexity of an algorithm in terms of exact number of basic operations is not so useful. This number may be quite complex, so it may be difficult to understand the behavior of the algorithm.

  Therefore, the performance should be stated in terms of growth rate of the complexity function, rather than the exact form of that function. In other words, we should express the complexity using asymptotic notation.

2.

a) $\sum_{i=1}^{n} \frac{i}{n} = \frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2} \in \theta(n)$

b) $\sum_{i=1}^{n} \frac{n}{i} = n\sum_{i=1}^{n} \frac{1}{i} = nH(n) \in \theta(n \log n)$

c) Writing any positive integer x in binary notation requires exactly $\lfloor \log_2 x \rfloor + 1$ bits. Thus, writing 10^n in binary requires $\lfloor \log_2 (10^n) \rfloor + 1 = \left\lfloor \frac{\log(10^n)}{\log 2} \right\rfloor + 1 = \lfloor 3.32n \rfloor + 1 \in \theta(n)$ bits.

d) This recurrence has the form of the recurrence in the Master Theorem, with a=9, b=3, and d=1. Since $a>b^d$, the solution is $\theta(n^{\log_b a}) = \theta(n^{\log_3 9}) = \theta(n^2)$.

e) $T(n) = T(n-2) + \frac{3}{n} = ... = \sum_{i=1}^{n/2} \frac{3}{2i}$ by backward substitution

$$= \frac{3}{2} H(\frac{n}{2}) \cong \frac{3}{2} \ln \frac{n}{2} \in \theta(\log n)$$

3. Probability that X=L[i], 1≤i≤n, is 1/m.
Thus, probability that X≠L[i] is (m-1)/m.

For 1≤i≤n-1 :

p_i = P(X is not in L[1:i-1])*P(X=L[i])

$$= \left(\frac{m-1}{m}\right)^{i-1} \frac{1}{m}$$

For i=n :

p_i = P(X is not in L[1:n-1])*P(X=L[n]) + P(X is not in L)

$$= \left(\frac{m-1}{m}\right)^{n-1} \frac{1}{m} + \left(\frac{m-1}{m}\right)^{n} = \left(\frac{m-1}{m}\right)^{n-1}$$

Thus,

$$A(n,m) = \sum_{i=1}^{W(n,m)} i * p_i = \sum_{i=1}^{n-1} \left[i \left(\frac{m-1}{m}\right)^{i-1} \frac{1}{m} \right] + n \left(\frac{m-1}{m}\right)^{n-1}$$

$$= \frac{1}{m} \left[\frac{(n-1)\left(\frac{m-1}{m}\right)^{n} - n\left(\frac{m-1}{m}\right)^{n-1} + 1}{\left(\frac{1}{m}\right)^{2}} \right] + n \left(\frac{m-1}{m}\right)^{n-1}$$

$$= m \left[n \left(\frac{m-1}{m}\right)^{n} - \left(\frac{m-1}{m}\right)^{n} - n \left(\frac{m-1}{m}\right)^{n-1} + 1 \right] + n \left(\frac{m-1}{m}\right)^{n-1}$$

As n→∞, $n\left(\frac{m-1}{m}\right)^{n}$ and $\left(\frac{m-1}{m}\right)^{n}$ go to zero. Thus,

$$\cong m$$

4. procedure LocalMin (A[low:high])
    m=$\lfloor (low + high)/2 \rfloor$
    if (A[m-1]≥A[m]) and (A[m]≤A[m+1]) then
        return A[m]
    else
        if (A[m-1]<A[m]) then
            call LocalMin (A[low:m])
        else
            call LocalMin (A[m:high])
        endif
    endif
end

The logic of the algorithm is as follows: We compare the middle element A[m] to its two neighbors A[m-1] and A[m+1]. If A[m] is the smallest of the tree, we have found a local minimum. Otherwise, at least one of the neighbors is smaller than A[m]. Suppose that A[m-1]<A[m]. Note that, in this case, the subarray A[1:m] satisfies the boundary condition of the problem, i.e. $A[1] \geq A[2]$ and $A[m-1] \leq A[m]$. Thus, we can call the procedure recursively. The process continues in this way until a local minimum is found, which is guaranteed to exist due to the boundary condition. Also note that there is no need for a terminating condition in the recursion, again due to this guarantee.

The algorithm divides the data into nearly two parts at each iteration. So, it is called recursively about log n times. If we accept the comparisons within the if statements as the basic operation, there will be at most three operations in each call. Thus, the worst case complexity will be O(log n).