Assignment 3
Analysis and Design of Algorithms CS-1401

Q1. While taking average of two numbers we usually forget to pay heed towards the limit of the registers and we do like (a+b)/2; the correct method is to do like ((a/2)+(b/2)). In this context answer the following questions:

a. Is this a divide and conquer strategy?
b. What is the time difference between two methods?
c. Implement a divide and conquer strategy to find |(a*b*c)/2| mod 7.

Q2. Applying Divide and Conquer strategy, write a recursive algorithm for finding the maximum and the minimum elements from a list.

Q3. Show that the equation $33n^2 + 4n^2 = \Omega\ n^2$.

Q4. Given the complexity f(n) of various types, find the data size than can be solved in 1 second and 1 minute.

***f*(n) in µ secs**	**1 second**	**1 minute**
$\lg n$		
$\sqrt{n}$		
n		
$n \lg n$		
n^2		
n^3		
2^n		

Q5. Which of the following is asymptotically smallest?
a. 2^n
b. $n^{\lg n}$
c. $n^{\sqrt{n}}$

Q6. State Master's Theorem and solve

a. $T(n) = 2\,T(n^{1/2}) + \lg n$.
b. $T(n) = 4T(\sqrt[3]{n}) + n$

Q7. For the following segment of code, find the exact value of T(n).

```
T(n)
1 if n=1
2         then return 1
3         else     sum = 0;
4                  for i= 1 to n -1
5                  do sum = sum + T(i)
6                       return(sum + n)
```

Q8. You are in front of a wall that stretches finitely in both directions. You know that there is a door in the wall, but it is dark and you only have a dim flashlight that allows you to see no more than one step in either direction. Write an algorithm to find the door.

Q9. Another way of looking at radix sort is to consider the key as just a bit pattern. So, if the keys are 4-byte integers, they are just considered as 32 bits, and if the keys are strings of 15 alphanumeric characters (15 bytes), they are just considered as 120 bits. These bit streams are then subdivided into pieces, which determine the number of passes and the number of buckets. So, if we have 120-bit keys, we might do 12 passes with 10-bit pieces, 10 passes with 12-bit pieces, or 5 passes with 24-bit pieces.

Q10. A counting sort can be used on a list that has no duplicate keys. In a counting sort, you would compare the first key in the list with every other element (including itself), counting the number of values that are less than or equal to this key. If we find that there are X keys that are less than or equal to this key, it belongs in location X. We can repeat this for all of the other keys, storing the values we get into a separate array that has the same number of elements as our list. When we have completed the counting, the extra array has all of the locations where the elements need to be for the list to be sorted, and it can be used to create the new sorted list.

a. Write a formal algorithm that will accomplish this counting sort.
b. Do a worst-case analysis of your algorithm from part (a).
c. Do an average-case analysis of your algorithm from part (a).

Q11. When a sorting algorithm is applied to a list of values, we are sometimes interested in knowing what happens when there are duplicate entries in the list. This is important in an application that sorts large records on a number of different fields and doesn’t want the efforts of a previous sort lost. For example, let’s say that records store a person’s first and last names in two separate fields. We could first sort the records based on the first name field and then sort them again on the last name field. If the sort algorithm keeps records with the same last name in the same order they were in after the first sort, the entire list would be properly sorted by full name.

If for every case where list[i] = list[j] (i < j), the sorting algorithm moves list[i] into location i’, moves list[j] into location j’, and i’ < j’, the sorting algorithm is called stable. In other words, if there are two records with the same key, a sorting algorithm is stable when those keys stay in the same relative order in the list even though they may move.

Prove which of the following sorting algorithms are stable:
a. Insertion sort
b. Bubble sort
d. Radix sort
e. Heapsort
f. Merge sort
g. Quicksort