Answers to MidSem

Question No.1

Assume that an algorithm has running time of $4n^4$ and the maximum input size that can be run on a computer C1 is m. Then, what would be the new maximum size input that can run on a computer C2 which is 4096 times faster than C1 to compute in same time?

Let us assume it can run input size 'k' at the same time. Then C2 can run in $4k^4$. However, it is $4096 = 2^{12}$ times faster than C1. So,

$4k^4 = 2^{12}\ 4n^4$

$k = 2^3\ n = 8n$

Question No.2

Mention two programming examples where you use stacks naturally.

You could take any examples which uses stack naturally. A few of them:

(a) Process/thread stack for calling functions
(b) Simulation of towers of Hanoi
(c) Computation of postfix notation (stack machines)
(d) Parenthesis checking
(e) String reversals
(f) Checking palindromes

Question No.3

- In an array-based implementation of *circular* queues, what is the problem if we use the check “f=r” (front and rear indexes) for the condition “queue is full”? How do you solve the issue?
- Note queue is a linked list with FIFO (First In First Out) order, where "front" and "rear" indices point to "front" and "rear" end of the queue. Inserts are done at the rear and delete is done from the front.

- For circular queue, this can be a problem, when we need to check f = r + 1 for full and f = r for emptiness. Otherwise, there is no issue.

Question No.4

Design a good hash table for keywords in a programming language. Suggest hash function for this set. Explain why you think that, with this table, the number of collisions will be minimal. How do you handle collisions?

hash (s) = $\sum_i$ s(i) p^i , i = 1, length.

If p = 1, then it becomes simple hash, faster while p > 1, can be reduce collisions. “ab” is not equal to “ba”. s(i) could be asci value but do not start with 0.

Collisions can be handled using linear chaining or quadratic probing. Linear chaining can lead to increased search times while rehashing is bounded. However, the table has to be resized in rehashing.

Question No.5

Strassen's matrix multiplication technique is as follows. Let A and B are two input matrices of size nXn each, then, the product AXB is given as follows:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad A \times B = \begin{pmatrix} D_1 + D_4 - D_5 + D_7 & D_3 + D_5 \\ D_2 + D_4 & D_1 + D_3 - D_2 + D_6 \end{pmatrix}$$

$$\begin{aligned} D_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ D_2 &= (A_{21} + A_{22})(B_{11}) \\ D_3 &= (A_{11})(B_{12} - B_{22}) \\ D_4 &= (A_{22})(B_{21} - B_{11}) \\ D_5 &= (A_{11} + A_{12})(B_{22}) \\ D_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ D_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

where, each of the above matrices (like A11 are (n/2)x(n/2) size matrices). What is the time complexity of this multiplication algorithm in terms of O(f(n))? Hint: Write the recurrence equation for the complexity first and then solve the recurrence.

Since usual matrix multiplication will have 8 matrix multiplication of sizes (n/2)x(n/2) while Strasses's will have 7. #additions are $O(n^2/4*4)=O(n^2)$. So, recurrence equation is: $T(n) = 7T(n/2) + O(n^2)$.

Expanding and solving using master theorem, we will get $O(n^{\log 7}) = O(n^{2.81})$

Question No.6

Write an algorithm which finds the maximum and the second maximum in an array of n numbers. Your algorithm should do better than 2n comparisons. Please state number of comparisons in f(n) and not just O(). Hint: You can use additional array of arbitrary size.

```
int firstHighest = arr[0];
    int secondHighest = arr[0];
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] > firstHighest) {
            secondHighest = firstHighest;
            firstHighest = arr[i];
        } else if (arr[i] > secondHighest) {
            secondHighest = arr[i];
        }
    }
```

Question No.7

Write node structure for a binary tree node in C. Info in the node is a structure with a integer for the studentID, two pointers each for student-name and student-address. Write a C function which deletes a node pointed to by the pointer, p. Make sure that there is no memory leak in the program.

```
struct node {
    int studentID;
    char *student-name;
    char *student-address;
}
```

```
if (p != NULL) {
    free (p->student-name);
    free (p->student-address);
    free (p);
}
```