Tutorial-1
IIIT-DELHI
Instructor: Debarka Sengupta

Stable Marriage

Que 1.

We have discussed one to one Stable Marriage (SM) in class, can you think of its variations?

a. Formulate a problem of one to many SM and how to solve it using Gale Shapley Algorithm (GS).
b. Formulate a problem of many to many SM and how to solve it using Gale Shapley Algorithm (GS).
c. In one to one SM both sides had the same number (n Men, n Women). What will be the number of entities on both sides now for part a and b?
d. What would be the time complexity for a and b?

Ans 1.

a. College taking admissions of students after IIT JEE exams. Colleges have preference criteria such as AIR Rank & 12th%. Students have a preference based on their interest of stream and location of the college.
b. Students of IIITD taking various courses. Courses have prerequisites and a limited number of seats whereas students have preferences based on their specialization and interests.
c. One to Many:
    - Let there be ***m*** colleges ***C1, C2, …, Cm***
    - Let there be ***n*** students ***S1, S2, …, Sn***
    - ***m <= n***
    - Each college ***Ci*** has capacity ***Li***. For all ***1 <= i <= m***.
    - Then, ***Sum(Li) = n***. For all ***1 <= i <= m***.
    - Not mandatory condition but if not so then some students or colleges may remain unhappy due to no remaining seats or some seats remaining respectively.

    Many to Many:
    - Let there be ***m*** courses ***C1, C2, …, Cm***
    - Let there be ***n*** students ***S1, S2, …, Sn***
    - Course ***Ci*** can intake ***Ki*** students, Student ***Sj*** can take ***Lj*** courses (credits). For all ***1 <= i <= m, 1 <= j <= n***
    - Then, ***Sum(Ki) = Sum(Lj)***. For all ***1 <= i <= m, 1 <= j <= n***.

    - Not mandatory condition but if not so then some students or courses may remain unhappy due to no remaining seats or some seats remaining respectively.

d. If looked on carefully Time complexity is same as GS applied on one to one SM ignoring the constant factor of traversing over preference list of both entities.

Asymptotic Analysis

Que 2.

What is the time complexity of insertion in a Binary Search Tree (BST)?

Ans 2.

Best Case/ Average Case: O(log n) where n is the number of insertions in the tree.

Worst Case: O(n) because of a skewed BST.

Que 3. Time Complexity?

```
void function(int n)
{
    int count = 0;

    // line 1
    for (int i=0; i<n; i++)

        // line 2
        for (int j=i; j< i*i; j++)
            if (j%i == 0)
            {
                // line 3
                for (int k=0; k<j; k++)
                    printf("*");
            }
}
```

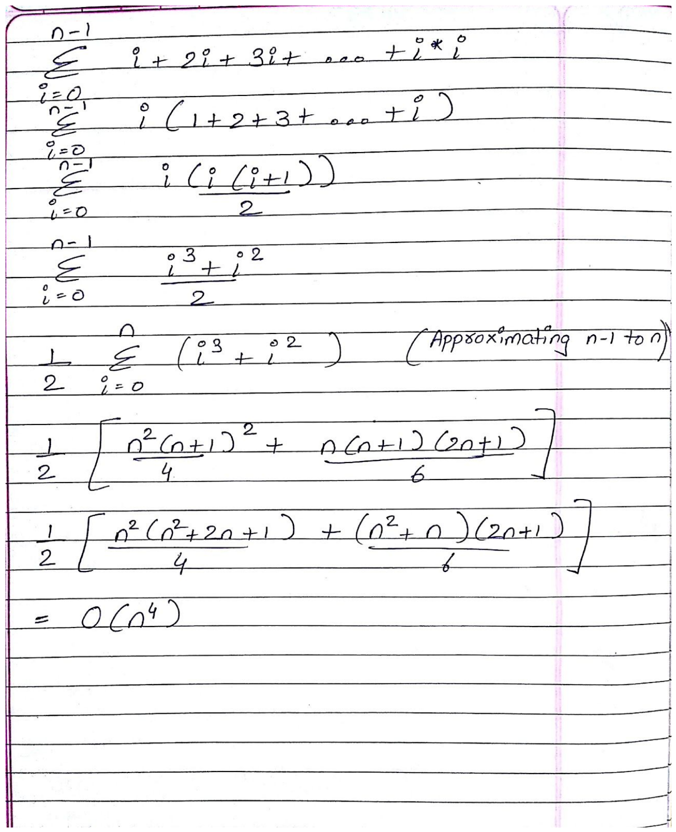
Ans 3.

```
void function(int n)
{
    int count = 0;
```

```
        // executes n times
        for (int i=0; i<n; i++)

            // executes O(n*n) times.
            for (int j=i; j< i*i; j++)
                if (j%i == 0)
                {
                    // executes O(n) times
                    for (int k=0; k<j; k++)
                        printf("*");
                }
    }
```

Due to j%i == 0. Inner loop will execute only i times because for loop is executed only when the if condition is fulfilled which returns true for i, 2i, 3i, … ,i*i



$$\sum_{i=0}^{n-1} i + 2i + 3i + \dots + i * i$$

$$\sum_{i=0}^{n-1} i(1+2+3+\dots+i)$$

$$\sum_{i=0}^{n-1} i\left(\frac{i(i+1)}{2}\right)$$

$$\sum_{i=0}^{n-1} \frac{i^3+i^2}{2}$$

$$\frac{1}{2}\sum_{i=0}^{n} \left(i^3+i^2\right) \quad (\text{Approximating } n-1 \text{ to } n)$$

$$\frac{1}{2}\left[\frac{n^2(n+1)^2}{4} + \frac{n(n+1)(2n+1)}{6}\right]$$

$$\frac{1}{2}\left[\frac{n^2(n^2+2n+1)}{4} + \frac{(n^2+n)(2n+1)}{6}\right]$$

$$= O(n^4)$$

A strict analysis could show:
Time Complexity: `O(n⁴)`

Que 4.

Use the definition of Big-Oh to prove that $n^{1+0.001}$ is not O(n).

Ans 4.

We prove that $n^{1+0.001}$ is not **O(n)** by contradiction.
Suppose that $n^{1+0.001}$ is **O(n)**, which means that we can find **c > 0** and $\mathbf{n_0 \geq 1}$ such that

$$n^{1+0.001} \leq c.n, \ \forall n \geq n_0 \qquad (4)$$

By dividing both sides of the inequality of (4) by **n (n ≥ 1)** we obtain the following:

$$c \geq n^{0.001} \qquad (5)$$

Inequality (5) can not be true since **c** must be a constant but $n^{0.001}$ is unbounded. In fact, as soon as $n > e^{1000 \log(c)}$ we have $c < n^{0.001}$.

This is a contradiction with the assumption that we can find such a constant **c**. Therefore, $n^{1+0.001}$ is not **O(n)**.

Que 5.

Given an array of n elements, where each element is at most k away from its target position, devise an algorithm that sorts the given array efficiently. What would be its time complexity?

```
Input : arr[] = {6, 5, 3, 2, 8, 10, 9}
            k = 3
Output : arr[] = {2, 3, 5, 6, 8, 9, 10}
```

Ans 5.

Trivial Methods:

Apply bubble sort over complete array O(n^2)
Apply Merge, Quick sort over complete array O(n log n) [or O(n^2) worst-case Quicksort]

Method-1:

Use min-heap, Create min heap of first k elements.
Extract-min 1 element & insert (k+1)th element in the heap.
Keep on doing this until the nth element is inserted.
Then perform k Extract-min operations to get a completely sorted array.
Time Complexity:

Create min heap -> O(k)
n-k insertions: n * log k
n Extract-mins: n * log k
Overall: O(n log k)

Method-2:

Let us take the size of the sub-array to be sorted in one iteration be k'.

Then, k' should be greater than equal to k so that 1 or more elements are placed at its correct sorted position, i.e., k' ≥ k

This will be a total time complexity of O(n k' log k')
We want some value of k' such that **=> `O(n k' log k') = O(n log k)`**

Case: K' = 2k
In this case, we will surely get k minimum elements in sorted order.
This sorting operation could then be applied on to remaining elements excluding the first k.

Repeating this step (n/k) times, each time will give k sorted elements hence n sorted elements.

`=> n * k' * log k' = (n / k) * (2k) * log(2k)`
`=> O(n log k)`

Que 6.

When $n = 2^{2k}$ for some k ≥ 0, the recurrence relation

T(n) = √(2) T(n/2) + √n, T(1) = 1

evaluates to?

Ans 6.

Explanation: Please note that the question is asking about the exact solution. Master theorem provides results in the form of asymptotic notations. So we can't apply the Master theorem here. We can solve this recurrence using a simple expansion or recurrence tree method.

T(n) = √(2) T(n/2) + √n
= √(2) [√(2) T(n/4) + √(n/2)] + √n
= 2 T(n/4) + √2 √(n/2) +√n
= 2[√2 T(n/8) + √(n/4)]+√2 √(n/2)+√n
= √(2^3) T(n/8)+ 2 √(n/4) + √2 √(n/2) +√n

= √(2^3) T(n/8)+√n +√n +√n
= √(2^3) T(n/(2^3))+3√n
...
= √(2^k) T(n/(2^k))+k√n
= √(2^logn) + logn √n
= √n + logn √n
= √n(logn +1)

Que 7. Apply Master's Theorem

$T(n) = 64\ T(n/8) - n^2 \log n$

Ans 7.

Master's Theorem cannot be applied here because the recurrence has a -ve sign which denotes that if you don't do extra work ($n^2 \log n$) the problem gets divided into subproblems. Master's Theorem says this is an invalid equation form.

Que 8. Apply Master's Theorem

$T(2^k) = 3T(2^{k-1}) + 1;\ T(1) = 1$

Ans 8.

Let $2^k = n$
=> $2^{k-1} = 2^k/2 = n/2$
=> $T(n) = 3T(n/2) + 1$
=> $T(n) = Ø(n^{\log 3})$ [theta]
=> $T(n) = Ø(2^{k\log 3})$
=> $T(n) = Ø(3^k)$