Tutorial-3
IIIT Delhi
Instructor: Debarka Sengupta

Problem 1.

There are n balls of identical weights except one, means n-1 balls of 1 kg and one ball of 2 kg, there is one scale to measure balls as shown in figure. But every time when you will compare the balls using scale than it will count 1 step.

1. Calculate the minimum number of steps required to find out that heavy ball.
2. Write algorithm for above mentioned problem using recursion and analyze complexity. Prove the correctness of your algorithm.

Answer 1.

Algorithm:

1. Divide the balls into three groups of the same size (or two of the same size and one with one different)
2. Compare two groups with equal number of balls on the scale. If they weigh the same, return the third group; else return the heavier one.
3. Continue step 2 till one ball is left

At each step, we divide the number of balls into 3 groups, and conduct one weighing

Therefore, minimum number of steps required is $\log_3(N)$.

Recurrence:

$T(n) = 1 + T(n/3)$
With $T(1)=T(2)=T(3)=1$

The algorithm terminates in a finite number of steps.

Problem 2.

Solve the following recurrences using the master method where possible:

a. T(n) = 9*T(n/3) + n
b. T(n) = 2*T(n/2) + nlog(n)
c. 2. T (n) = 16T (n/4) + n!

Answer 2.

a. Case 1 of the theorem: a= 9, b = 3, f(n) = n
   T(n) = theta(n^2)

b. Can't apply master method because nlog(n) is not polynomially larger than n ($n^{log(a)/log(b)}$ = n; a=b=2).
   The ratio f(n)/n = log(n) is asymptotically less than n^e for and positive constant e.

   T(n) = 2T(n/2) + n*log n

   T(1) = 1

   Substitute n = 2^k, k = log(n)

   $T(2^k) = 2*T(2^{k-1}) + k*2^k$

   Divide both sides by 2^k

   $T(2^k)/2^k = T(2^{k-1})/2^{k-1} + k$

   Let $T(2^k)/2^k = S(k)$

   Then S(k) = S(k-1) + k ---> **1**

   Solve **1** to get $S(k) = \Theta(k^2)$

   Now $T(2^k)/2^k = \Theta(k^2)$

   $T(2^k) = 2^k * \Theta(k^2)$

   $T(n) = n * \Theta(log^2(n))$

   $T(n) = \Theta(n * log^2(n))$

c. Case 3 of the theorem
   T(n) = theta(n!)

Problem 3.

Analyze the recurrence function and find its time complexity via master's theorem. (Assume constant time taken for comparing and returning the value = 1)

```
A(n)
{
   if(n<=1)
     return 1;
   else
     return A(√n);
}
```

Answer 3.

We can write a recurrence relation for the given code as- $T(n) = T(\sqrt{n}) + 1$

where 1 = Constant time taken for comparing and returning the value.

We can not directly apply Master's Theorem on this recurrence relation but we can somehow modify it and bring it in the general form for applying Master's Theorem.

Let, $n = 2^m$ ……(1)

Then, $T(2^m) = T(2^{m/2}) + 1$

Now, let $T(2^m) = S(m)$, then $T(2^{m/2}) = S(m/2)$

So, we have-

$$S(m) = S(m/2) +1$$

Now, we can easily apply Master's Theorem.

On comparing the given recurrence relation with-

$$S(m) = aS(m/b) + \theta (m^k)$$

we have-

$a = 1, b = 2, k = 0$

Now, $a = 1$ and $b^k = 2^0 = 1$

Clearly, $a = b^k$

So, we follow case-02. Therefore $S(m) = \theta(\log m)$

Hence $T(n) = \theta (\log\log_2 n)$

Problem 4.

You are given a sorted array A with n integers and an integer w and you want to determine whether there exists two distinct indices i, j in the array such that A[i] + A[j] = w. Design a recursive algorithm for this problem. Analyze the running time of your algorithm.

Answer 4.

CheckSum(A, w, i, j)
- if (j ≤ i) return("No")
- if (A[i] + A[j] = w) return("Yes")
- if (A[i] + A[j] > w) return(CheckSum(A, w, i, j − 1))
- else return(CheckSum(A, w, i + 1, j))

The recurrence relation for the running time T(n) is given by: $T(n) = T(n - 1) + \Theta(1)$; $T(1) = \Theta(1)$. Solving the recurrence we get that $T(n) = \Theta(n)$.

Problem 5.
Master's theorem is not applicable in the following cases. Explain why?

- T (n) = (2^n)T (n/2) + n^n
- f(n) is smaller than n^logba but not polynomially smaller.
- T (n) = 2T (n/2) + n/log n
- T (n) = 64T (n/8) − 2nlog n
- T (n) = (0.5)T (n/2) + n^n

Answer 5.

T (n) = (2^n)T (n/2) + n^n ⇒ Does not apply (a is not constant, ie subproblems are not constant)

non-polynomial difference between f(n) and n^(logba) (Ratio is less than n^0)

T (n) = 2T (n/2) + n/log n ⇒ Does not apply (non-polynomial difference between f(n) and n^(logba))

T (n) = 64T (n/8) − 2nlog n ⇒ Does not apply (f(n) is not positive)

a<1 there must be atleast one sub problem