**Master Theorem**

Master Theorem The Master Theorem applies to recurrences of the following form:

T (n) = aT (n/b) + f(n)

where a ≥ 1 and b > 1 are constants and f(n) is an asymptotically positive function.

There are 3 cases:

1. If f(n) = O(n^(logb a) for some constant > 0, then T (n) = Θ(n^(logb a) ).

2. If f(n) = Θ(n^(logb a) \* log^(k)n) with1 k ≥ 0, then T (n) = Θ(n logb a logk+1 n).

3. If f(n) = Ω(n logb a) with > 0, and f(n) satisfies the regularity condition, then T (n) = Θ(f(n)). Regularity condition: af(n/b) ≤ cf(n) for some constant c < 1 and all sufficiently large n.

**Questions**

1. T (n) = 2T (n/4) + n^0.6

Solution → T (n) = Θ(n^0.6 ) (Case 3)

2. T (n) = 3T (n/3) + n/2

Solution → T (n) = Θ(n log n) (Case 2)

3. T (n) = T (n/2) + 2^n

Solution → Θ(2^n ) (Case 3)

4. T (n) = 2^(n)T (n/2) + n^n

Solution → Does not apply (a is not constant)

5. T (n) = 16T (n/4) + n

Solution → T (n) = Θ(n^2 ) (Case 1)

**---------------------------------------------------------------------------------------------------------------------**

**Question**

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

**Code**

package com.greatlearning.algorithm;

class TowerOfHanoi {

static void towerOfHanoiimplementation(int no, char from\_rod, char to\_rod, char auxilliary\_rod) {

if (no == 1) {

System.out.println("Move disk 1 from rod " + from\_rod + " to rod " + to\_rod);

return;

}

towerOfHanoiimplementation(no - 1, from\_rod, auxilliary\_rod, to\_rod);

System.out.println("Move disk " + no + " from rod " + from\_rod + " to rod " + to\_rod);

towerOfHanoiimplementation(no - 1, auxilliary\_rod, to\_rod, from\_rod);

}

public static void main(String args[]) {

int no = 5; // Number of disks

towerOfHanoiimplementation(no, 'X', 'Y', 'Z'); // X, Y and Z are names of rods

}

}

**---------------------------------------------------------------------------------------------------------------------**

**Concurrent Exception**

package com.greatlearning.exception;

import java.util.ArrayList;

import java.util.Iterator;

public class ConcurrentModification {

public static void main(String[] args) {

ArrayList<Integer> arrayList = new ArrayList<>();

arrayList.add(10);

arrayList.add(20);

arrayList.add(30);

arrayList.add(40);

arrayList.add(50);

Iterator<Integer> it = arrayList.iterator();

while (it.hasNext()) {

Integer value = it.next();

System.out.println("List Value:" + value);

if (value.equals(30))

arrayList.remove(value);

}

}

}

Concurrent exception

List Value:10

List Value:20

List Value:30

Exception in thread "main" java.util.ConcurrentModificationException

at java.base/java.util.ArrayList$Itr.checkForComodification(ArrayList.java:1013)

at java.base/java.util.ArrayList$Itr.next(ArrayList.java:967)

at com.greatlearning.exception.ConcurrentModification.main(ConcurrentModification.java:19)

**HashMap doesn’t throw Concurrent Exception**

package com.greatlearning.exception;

import java.util.HashMap;

import java.util.Iterator;

public class ConcurrentExceptionHandling {

public static void main(String[] args) {

HashMap<Integer, Integer> map = new HashMap<>();

map.put(1, 10);

map.put(2, 20);

map.put(3, 30);

Iterator<Integer> it = map.keySet().iterator();

while(it.hasNext()) {

Integer key = it.next();

System.out.println("Map Value:" + map.get(key));

if (key.equals(2)) {

map.put(1, 40);

}

}

}

}