Agenda.

→ Print numbers from 1 to 100, each from a different thread.

→ Executors

→ Callables & futures.

→ Merge sort in a multithreaded way.

Q. Print numbers from 1 to 100, each from a different thread.

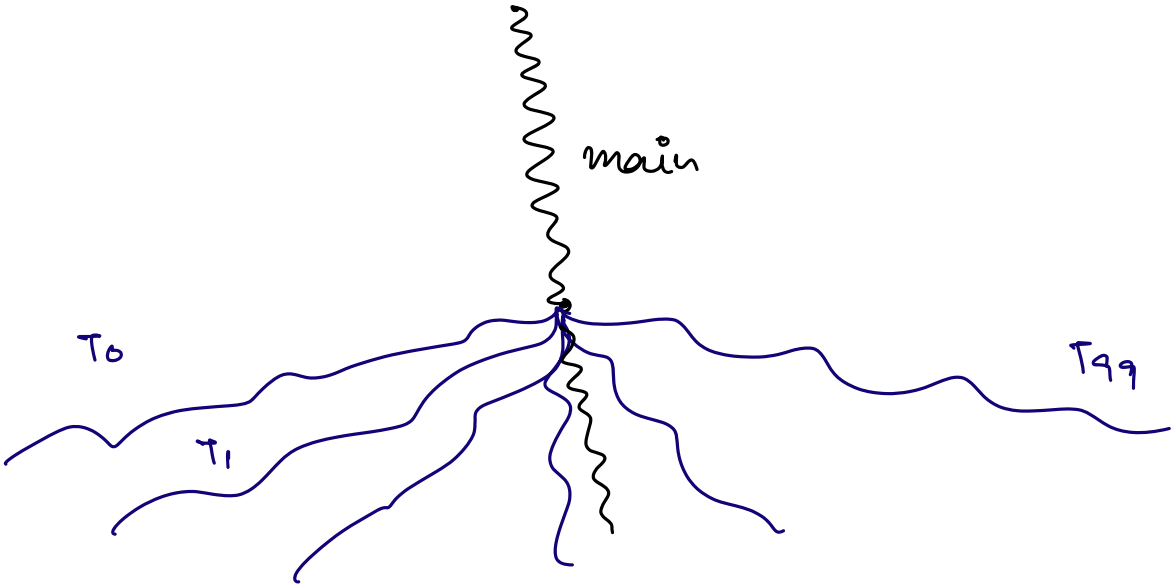
No. of threads => 100

Should be Noun. Task => print number i

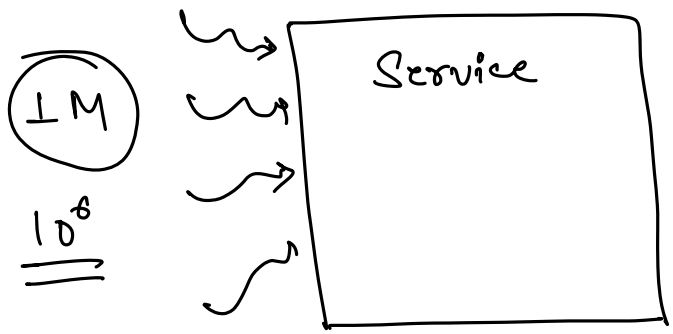
Task Class. => NumberPrinter.

```
Class NumberPrinter implements Runnable {
    int number;

    NumberPrinter (int number) {
        this.number = number;
    }
    void run() {
        sout(number);
    }
}
```



⇒ 100 thread objects.



Thread t = new Thread()

⇒ In real production environments, we don't really create threads manually

⇒ Executors Service.

⇒ Service that helps to manage threads in our code more efficiently

⇒ Efficiently manages multiple threads.

⇒ Executor uses something called as ThreadPool.

100 Threads.

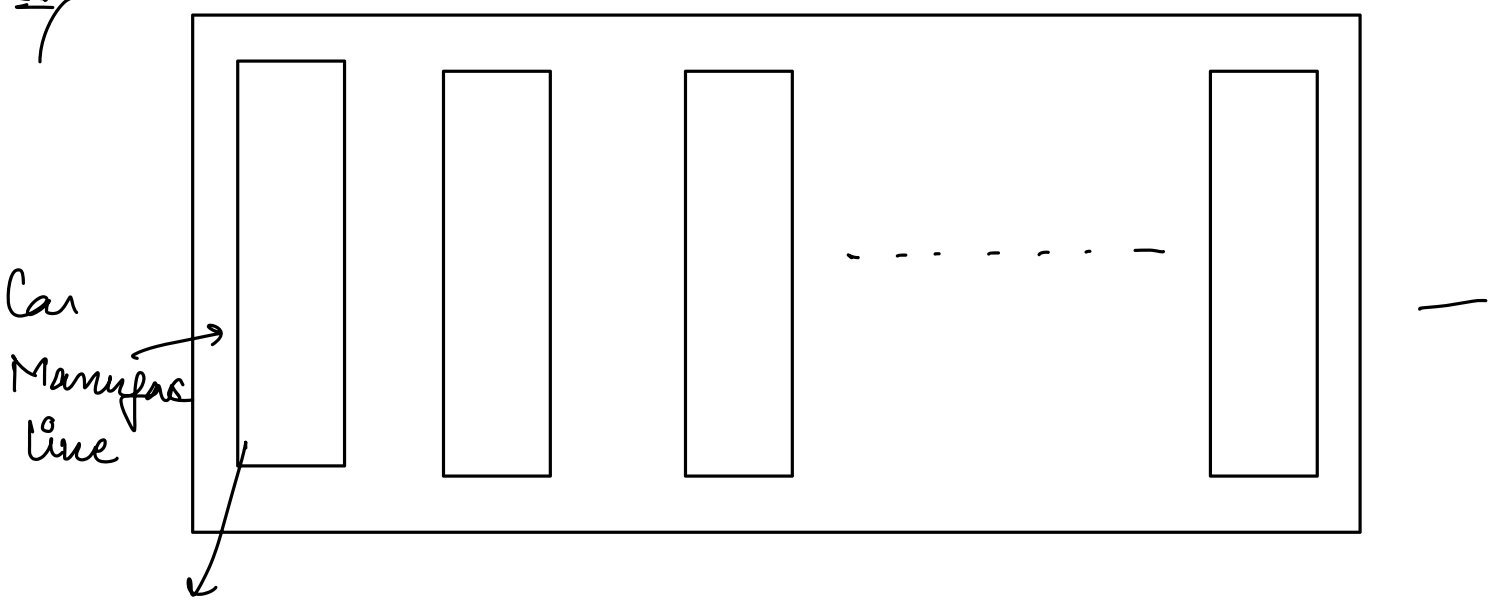
T0 → Print 1 → Done

T1 → Print 2 → Done

⋮

⇒ We can reuse threads for efficient usage.

⇒



It can manufacture
1 car at a time.

⇒ 10000 Safari's in Mar24.

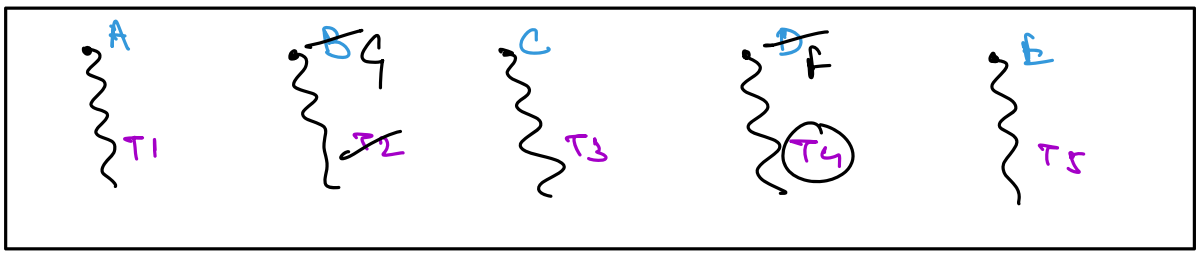
⇒ Reusing the manufacturing line.

Executor Service

No. of req = 100

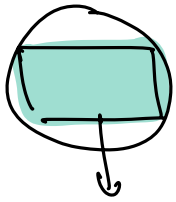
waiting Queue : 94 93 ..... 95 tasks

Thread Pool



CALLABLE.

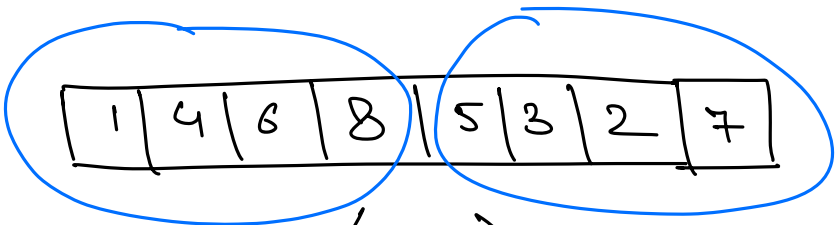
↳ Exactly same as Runnable, the only diff is the return type.

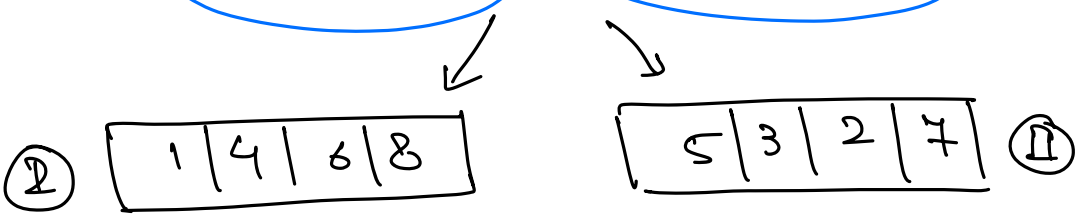


call()

generic

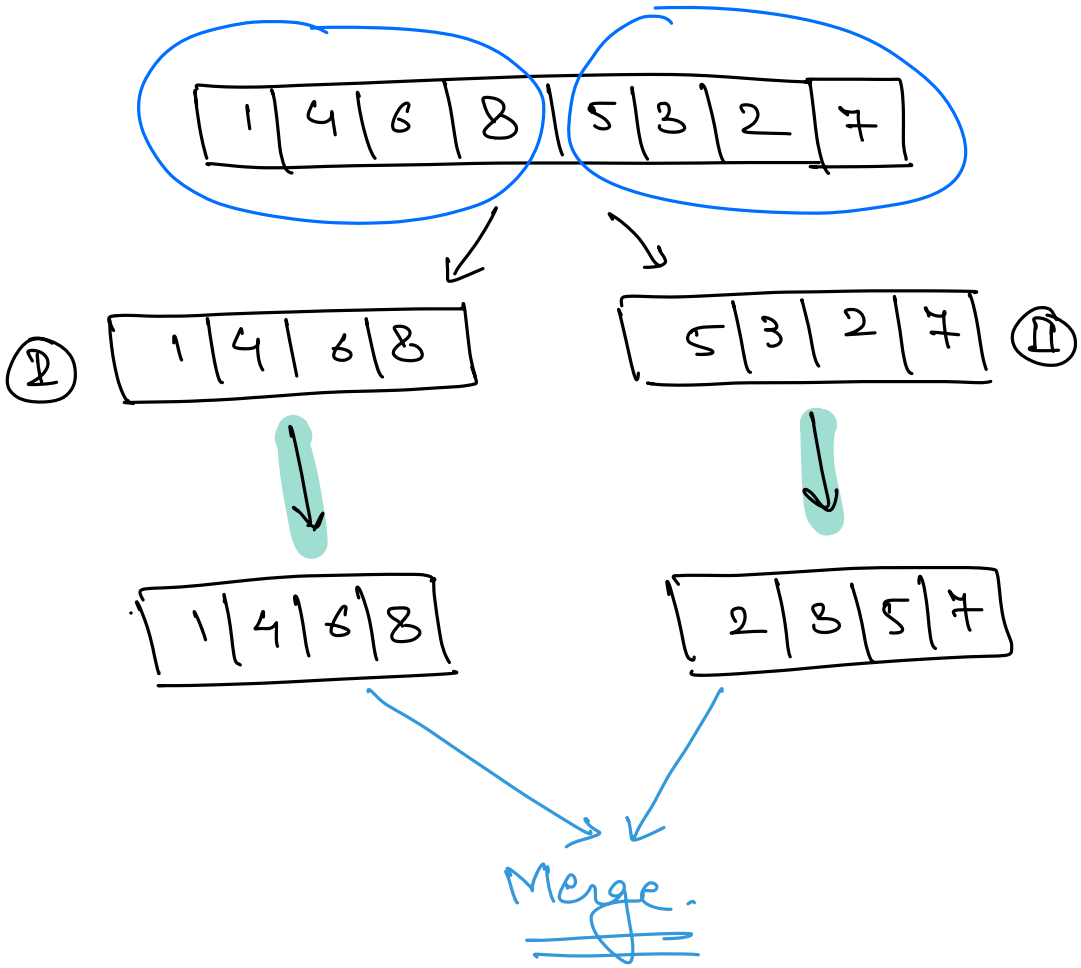
Problem Statement : Sort a list of Int using Merge Sort.





⇒ Sorting I & II halfs are completely independent of each other.

⇒ So they can be sorted parallelly



Callable : like runnable, it is used to implement a task. Unlike runnable, task can return some data.

(1) Create the task class.

Sorter

(2) Class Sorter implements Callable<List<Int>>

(3) Implement call method

Class Sorter implements Callable<List<Int>>{

List<Int> Call(){

// Merge Sort. logic

}

}

⇒ Implementation

FUTURES.

→ Sout("Hello") ✓

→ int x = fun(—);

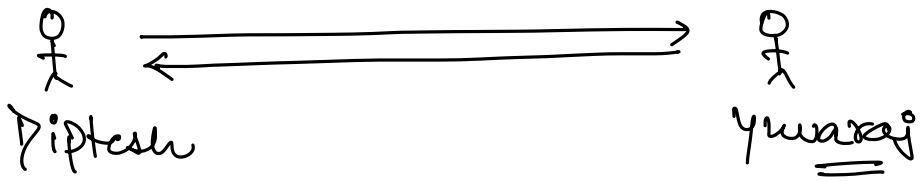
Sout(x)

⇒ List<Int> left = executor.submit(—);

List<Int> right = executor.submit(—);

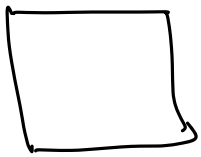
Future<List<Int>> leftFuture = executor.submit(—);

CLRS



Future

Bag

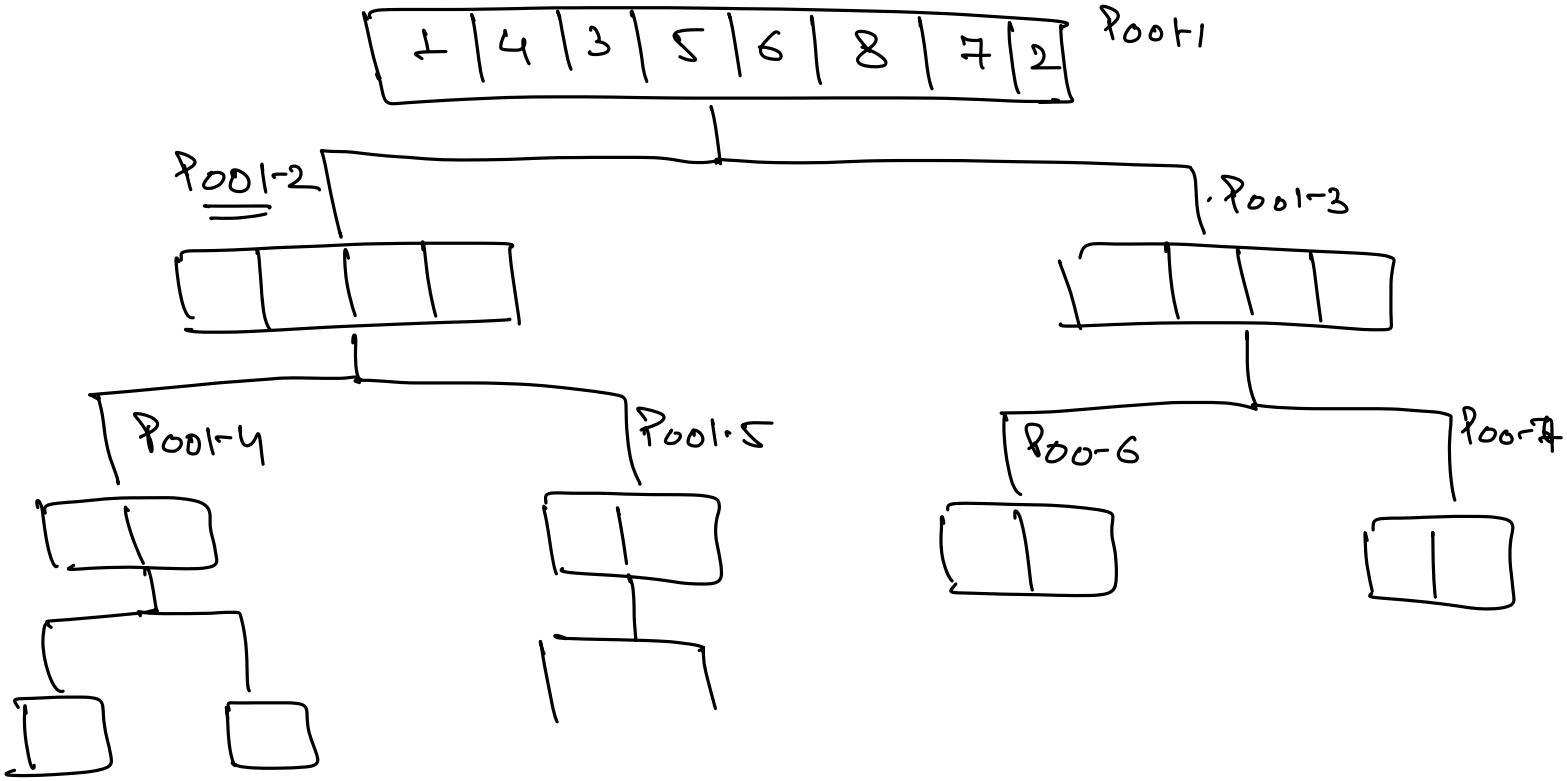


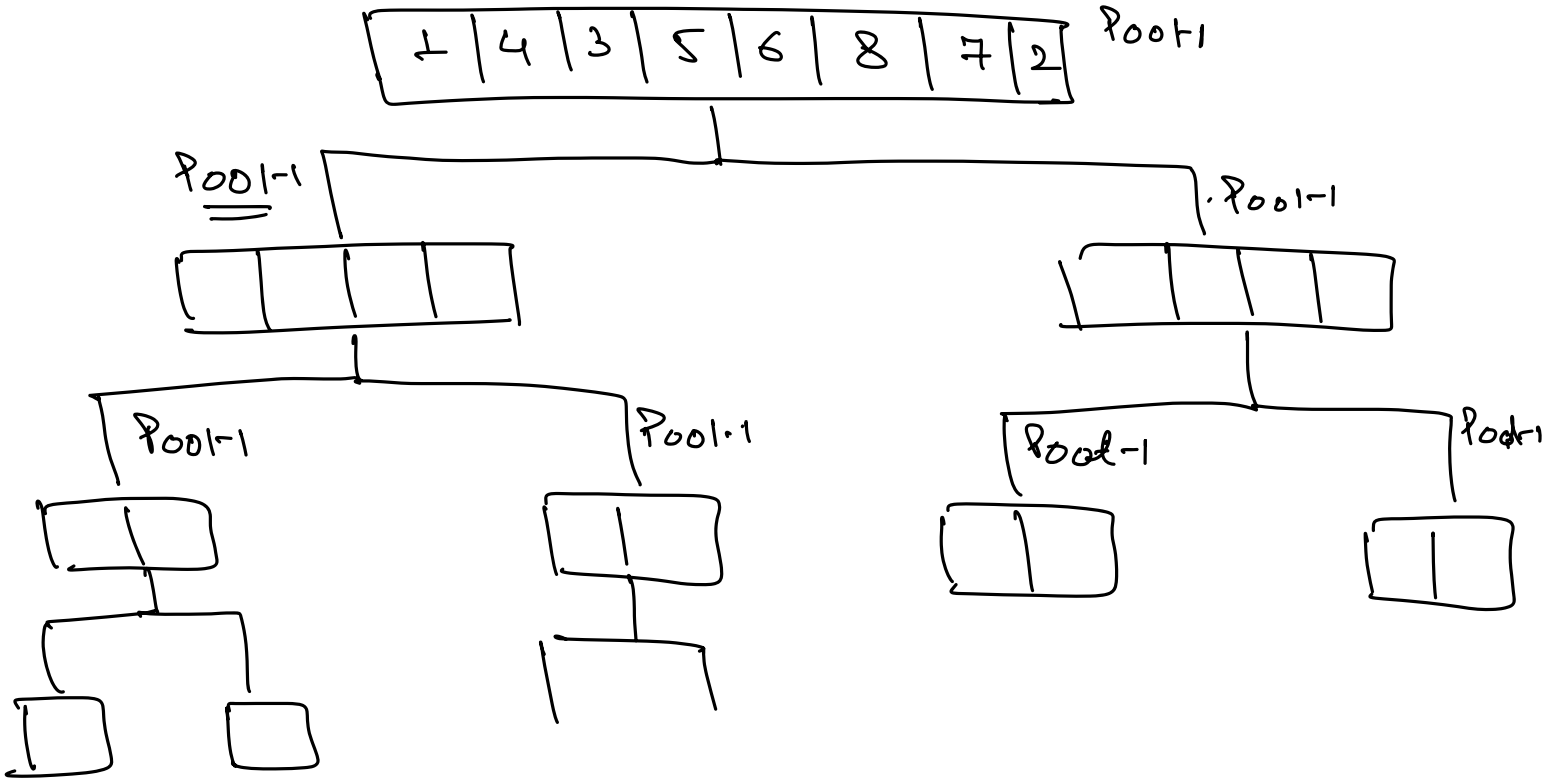
Assurance from Yuvaraj

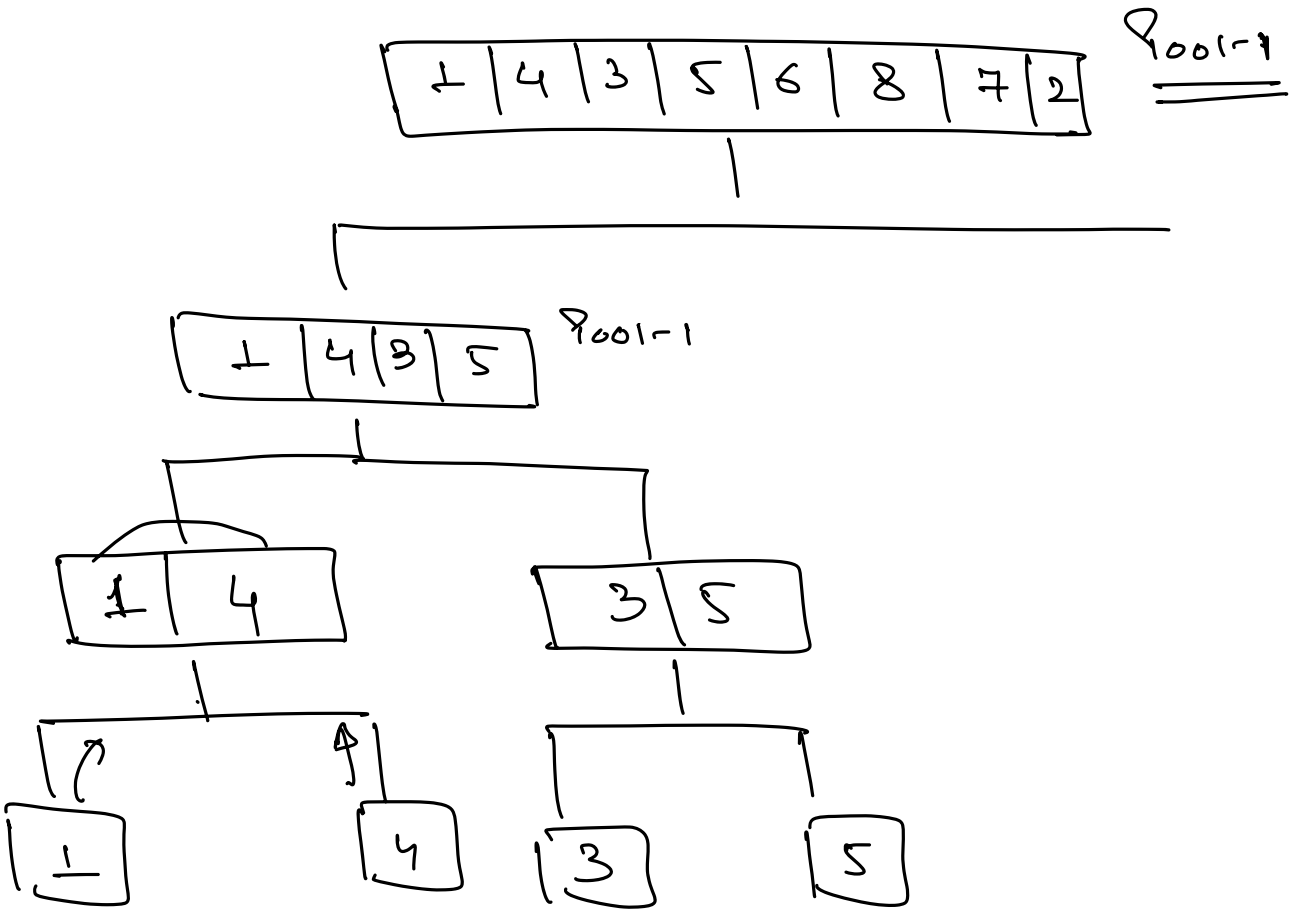
⇒ When Diptesh checks the Bag :

(1) If the Book is already, Diptesh will get it.

(2) Else, Diptesh will have to wait-


1 | 4 | 3 | 5 | 6 | 8 | 7 | 2
Pool-1
Pool-2
Pool-3
Pool-4
Pool-5
Pool-6
Pool-7


1 | 4 | 3 | 5 | 6 | 8 | 7 | 2
Pool-1
Pool-1
Pool-1
Pool-1
Pool-1
Pool-1
Pool-1



⇒ Implementing Merge Sort in a multithreaded env using Executor Service.