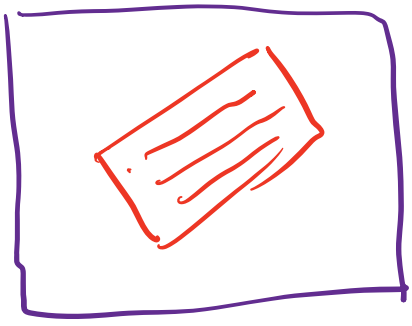
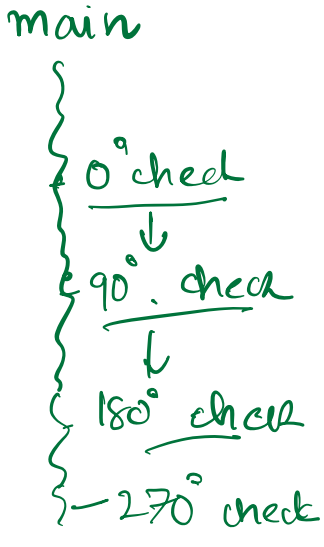
Agenda

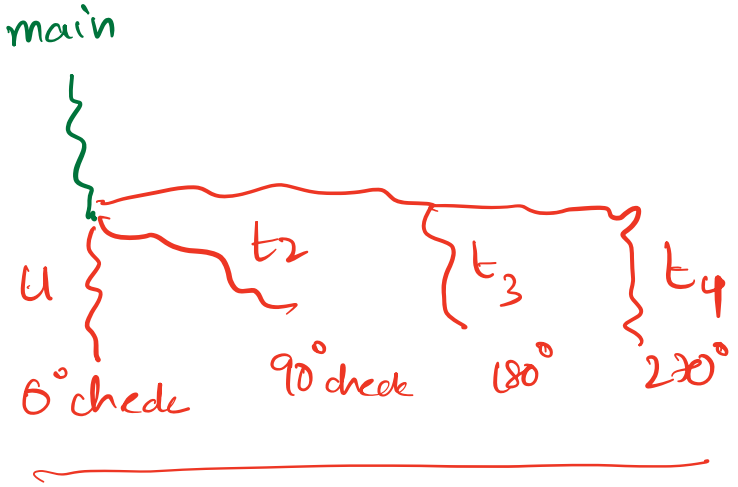
1. Print 1-100 numbers using diff threads
2. Executors
3. Thread pools & types of thread pools
4. Callables
5. Merge Sort - Multithreaded.

Delivery Scan issue



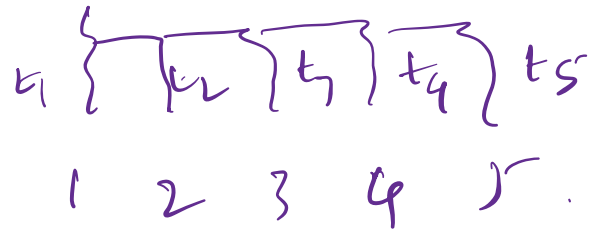
→ 0°, 90°, 180°, 270°
↓ ↓ ↓ ↓
check check check check.





Sequential

Q1 Print 1-100 numbers using diff. threads? (not in order)



(1) Task – Print a number.

```
class PrintNumber implementing Runnable {
    int num;
    @Override
    public void run() {
        System.out.println(num);
    }
    public PrintNumber(int num) {
        this.num = num;
    }
}
```

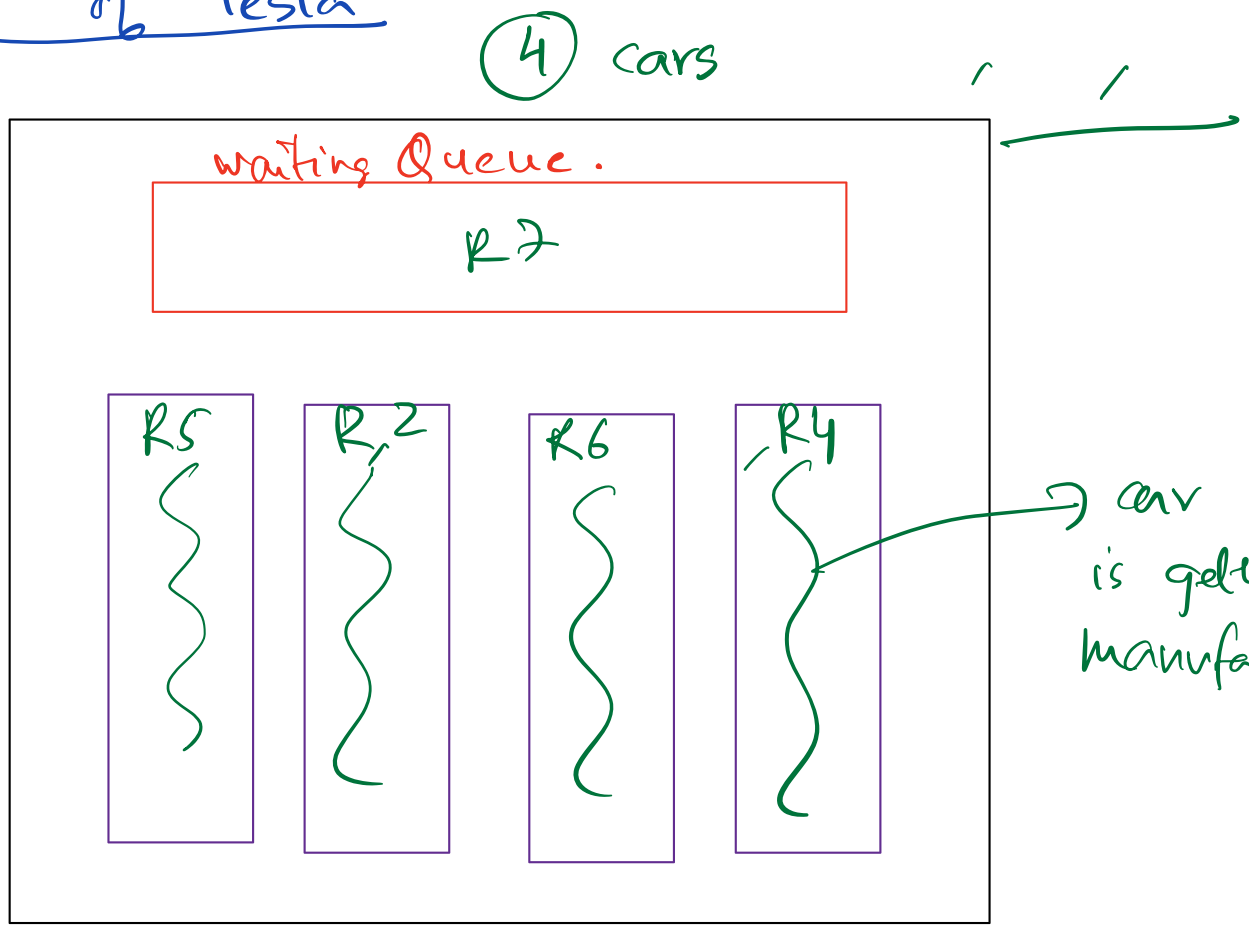
Main :

```
for (i → 100) {

    PrintNumber p = new PrintNumber(i);
→   Thread t = new Thread(p);
    t.start();
}
```

- creation
- Execute Run
- deletes the thread

Executors, Thread Pools

Analogy of Tesla



```
ExecutorService es =
        Executors.typeOfThreadPool();
```

Types of Thread Pool :

① newFixedThreadPool

```
ExecutorService es =
        Executors.newFixedThreadPool(no.of threads);
```

Note :

1. No. of threads given to thread pool is the maximum no. of threads it can create.
2. Only when required thread will be created.

   4 tasks ⇒ 4 threads will be created.

② newCachedThreadPool :

```
ExecutorService es =
        Executors.newCachedThreadPool();
```

no thread number is required

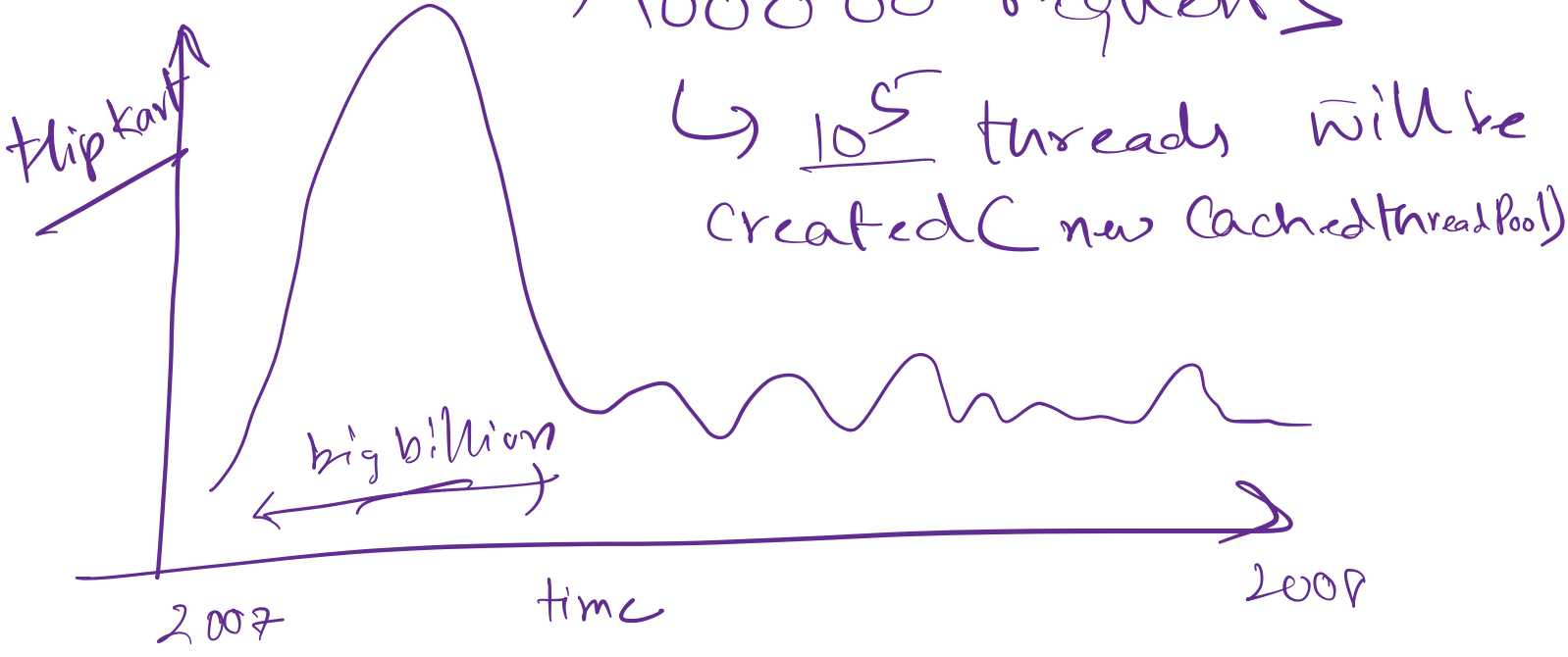
Note:

1. No threads gets deleted.
2. It doesn't have a queue,
   if thread is available,
   task is assigned to thread.
   Else, if thread not available
   new thread created and task assigned.

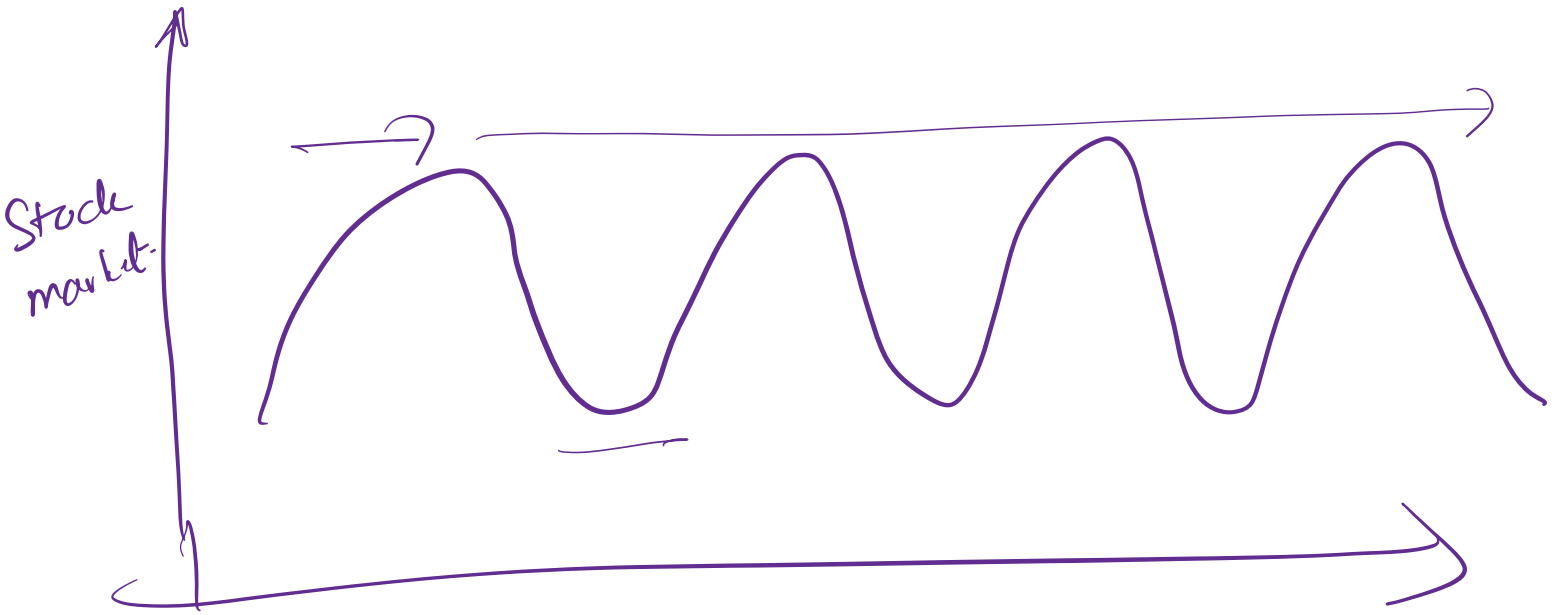
Note:

1. No. of threads depends on the task.
2. General estimation → Min no. of threads you should use = 2 × no. of cores.
   ie. 4 cores means 8 threads min.

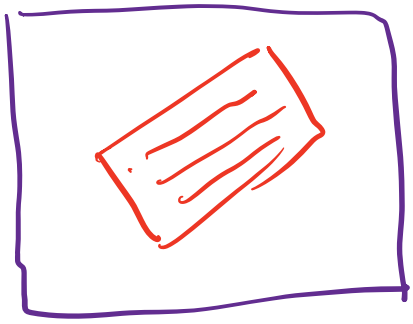
→ 100000 requests
↳ 10^5 threads will be created (new CachedThreadPool)



In case of newFixedThreadPool → 100 threads.



Callables :



→ 0° , 90° , 180° , 270°

check check check check.

main

0° check

90° check

180° check

270° check

Sequential

main

t1 t2 t3 t4

0° check

90° check

180°

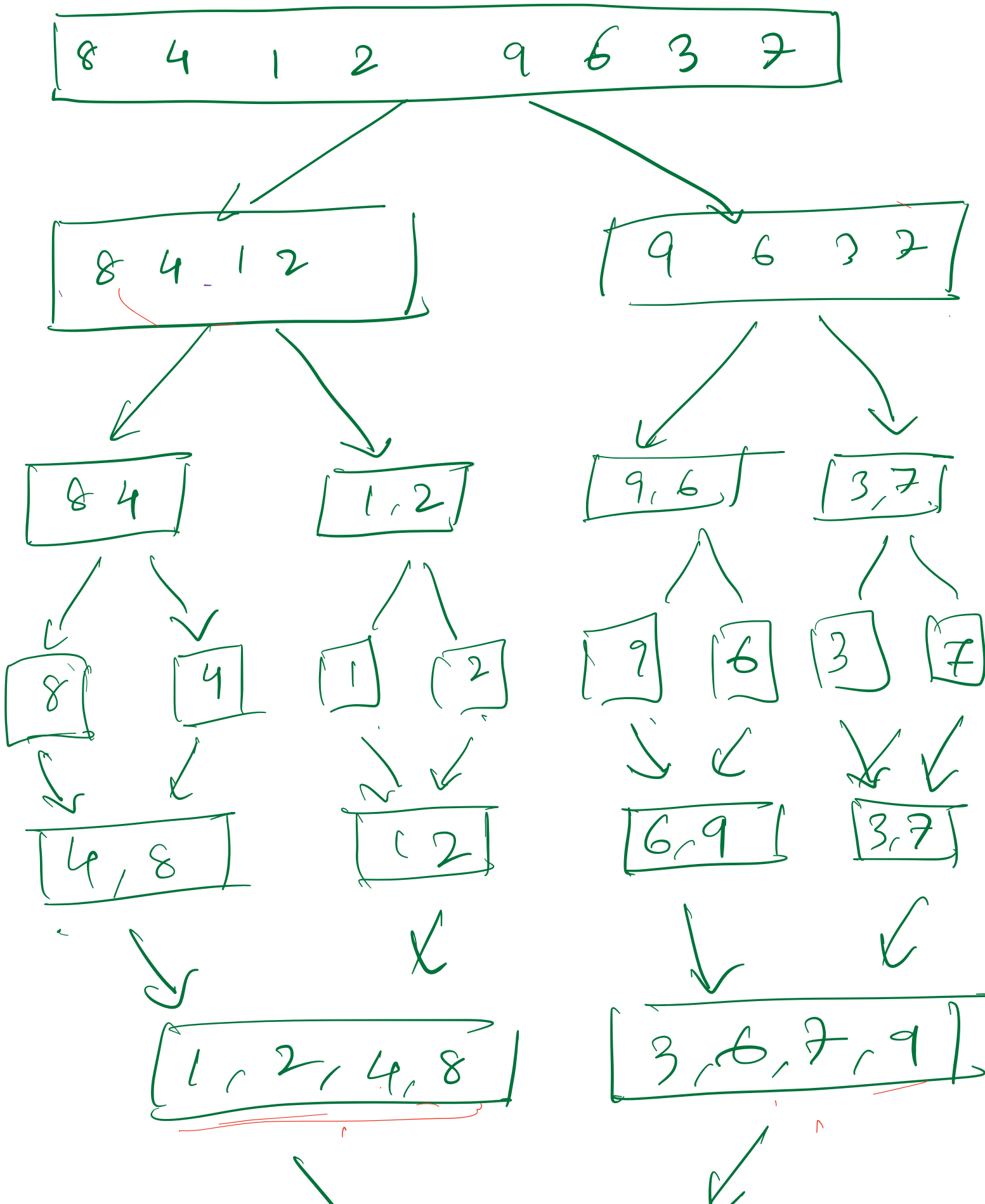
270°

```
interface Runnable {
    public void run();
}
```

notify

```
interface Callable<T>{

    public <T> call();

}

class FetchData imp. Callable<String> {
    @ public String call(){
        return "data";
    }
}
```

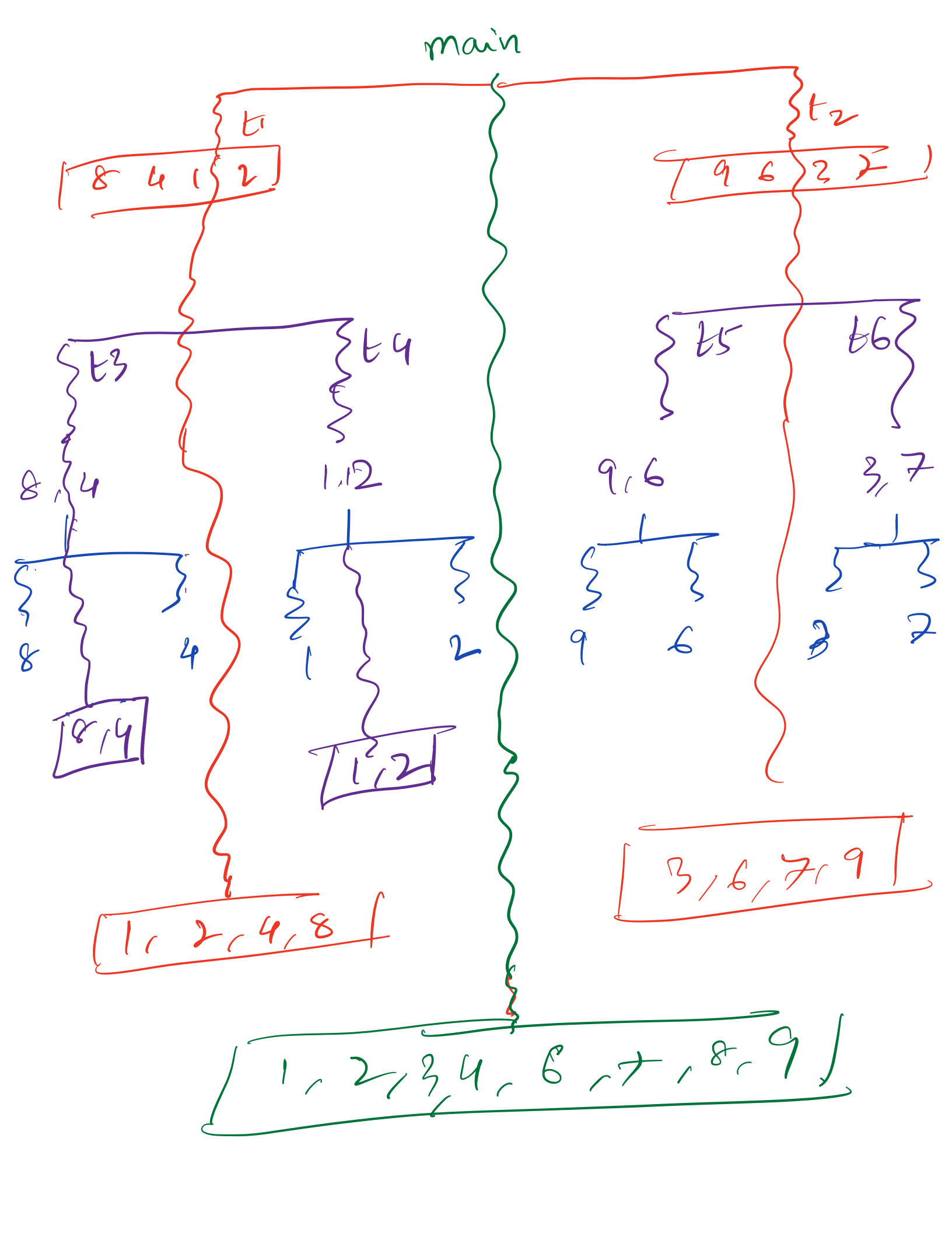
Multithreaded Merge Sort:



[1, 2, 3, 4, 6, 7, 8, 9]

Pseudo Code:

```
sorter( array) {
    if (array.size <= 1) return array;

    leftArr  = array(0 .... mid);
    rightArr = array(mid+1 .... end);

    sortedLeft  = sorter(leftArr);
    sortedRight = sorter(rightArr);      } can be done parallely

sortedArr = merge(sorted left, sorted Right)

    return sortedArray;

}
```


main
t1
8 4 1 2
t2
9 6 3 7
t3
t4
t5
t6
8, 4
1, 12
9, 6
3, 7
8
4
1
2
9
6
3
7
8, 4
1, 2
1, 2, 4, 8
3, 6, 7, 9
1, 2, 3, 4, 6, 7, 8, 9