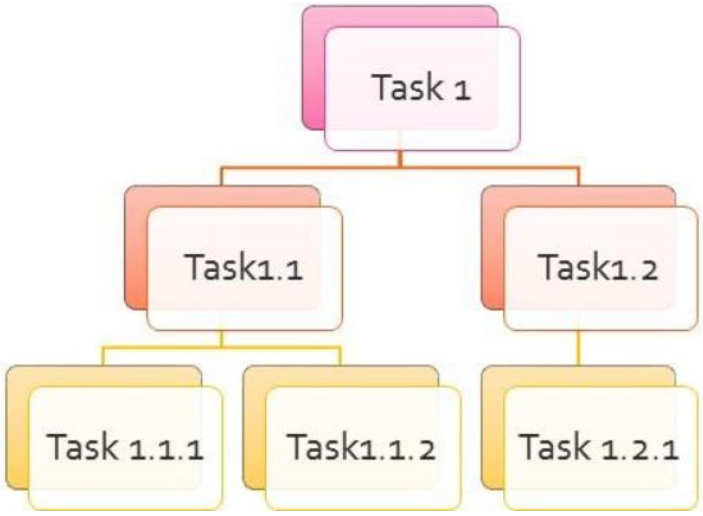
JAVA CONCURRENCY FORK JOIN POOL

FORK JOIN POOL

The ForkJoinPool was intoduced in Java 7. sameis similar to the **Executor framework** but with one difference. Forkjoin pool act in recursive way unlike Executor thread it splits the task then submit task to worker Thread but ForkJoin pool takes a big task then

Split into smaller task again those smaller task splits them to sub tasks until each subtask is atomic or not divisible. So it work's recursively. Please remember this difference.

Please see the following picture to understand ForkJoin pool concept.



Divide Task in to smaller task

Fork: To split the tasks from Bigger task. Ex: Task 1.1 splits to Task 1.1.1 and Task 1.1.2

Join: Getting result from immediate subtasks. Ex: Task 1.1 take results from Task 1.1.1 and Task 1.1.2

Fork Join pool is faster than Executor service.

Fork join Vs Executor service.

Example:

Suppose we want search an element in a sorted array. So obvious we take the help

Of Binary Search.

Binary Search Algorithm:

Step 1: First we split and determine the mid element of the array then check

Step 2 : Check element is equal to mid element. If it is equal then return the element as Found .

Step 3 : If element is less than mid element then we create a new subtask in this sub task we take left half of the array.

Step 4: If element is greater than mid element then we create a new subtask in this sub task we take right half of the array.

Step 5: Until element is not found continue the step 4 and 5.

Step 6: If array size is 1 and element is not equal to that element. Return Element Not Found.

Coding:

```
package com.example.concurrency;

import java.util.Arrays;
import java.util.concurrent.RecursiveTask;

public class ForkJoinSearcher extends RecursiveTask<Boolean>{

	int[] arr;
	int searchableElement;
	ForkJoinSearcher(int[] arr,int search)
	{
		this.arr = arr;
		this.searchableElement=search;

	}
	@Override
	protected Boolean compute() {

		int mid=( 0 + arr.length)/2;
		System.out.println(Thread.currentThread().getName() + " says : After
splliting the arry length is :: "+ arr.length + " Midpoint is " + mid);

		if(arr[mid]==searchableElement)
		{
			System.out.println(" FOUND !!!!!!!!!");
			return true;
		}
		else if(mid==1 || mid == arr.length)
		{
			System.out.println("NOT FOUND !!!!!!!!!");
			return false;
		}
		else if(searchableElement < arr[mid])
		{
			System.out.println(Thread.currentThread().getName() + " says ::
Doing Left-search");
			int[] left = Arrays.copyOfRange(arr, 0, mid);
			ForkJoinSearcher forkTask = new
ForkJoinSearcher(left,searchableElement);
			forkTask.fork();
			return forkTask.join();
		}
		else if(searchableElement > arr[mid])
		{
			System.out.println(Thread.currentThread().getName() + " says ::
Doing Right-search");
			int[] right = Arrays.copyOfRange(arr, mid, arr.length);
```

```
                    ForkJoinSearcher forkTask = new
ForkJoinSearcher(right,searchableElement);
                    forkTask.fork();
                    return forkTask.join();
             }

             return false;

      }

}



package com.example.concurrency;

import java.util.Arrays;
import java.util.concurrent.ForkJoinPool;

public class BinarySearch {

      int[] arr = new int[100];

      public BinarySearch()
      {
             init();
      }

      private void init()
      {
             for(int i=0; i<arr.length;i++)
             {
                    arr[i]=i;
             }

             Arrays.sort(arr);


      }

      public void createForJoinPool(int search)
      {
             ForkJoinPool forkJoinPool = new ForkJoinPool(50);
             ForkJoinSearcher searcher = new ForkJoinSearcher(this.arr,search);

             Boolean status = forkJoinPool.invoke(searcher);

             System.out.println(" Element ::" + search +" has been found in array? ::
" + status );

      }


      public static void main(String[] args) {
```

```
            BinarySearch search = new BinarySearch();
            search.createForJoinPool(10);
            System.out.println("**********************");
            search.createForJoinPool(104);

      }


}
```

Output :

```
ForkJoinPool-1-worker-57 says : After splliting the arry length is :: 100 Midpoint is 50
ForkJoinPool-1-worker-57 says :: Doing Left-search
ForkJoinPool-1-worker-57 says : After splliting the arry length is :: 50 Midpoint is 25
ForkJoinPool-1-worker-57 says :: Doing Left-search
ForkJoinPool-1-worker-50 says : After splliting the arry length is :: 25 Midpoint is 12
ForkJoinPool-1-worker-50 says :: Doing Left-search
ForkJoinPool-1-worker-57 says : After splliting the arry length is :: 12 Midpoint is 6
ForkJoinPool-1-worker-57 says :: Doing Right-search
ForkJoinPool-1-worker-50 says : After splliting the arry length is :: 6 Midpoint is 3
ForkJoinPool-1-worker-50 says :: Doing Right-search
ForkJoinPool-1-worker-43 says : After splliting the arry length is :: 3 Midpoint is 1
 FOUND !!!!!!!!!
 Element ::10 has been found in array? :: true
**********************
ForkJoinPool-2-worker-57 says : After splliting the arry length is :: 100 Midpoint is 50
ForkJoinPool-2-worker-57 says :: Doing Right-search
ForkJoinPool-2-worker-57 says : After splliting the arry length is :: 50 Midpoint is 25
ForkJoinPool-2-worker-57 says :: Doing Right-search
ForkJoinPool-2-worker-50 says : After splliting the arry length is :: 25 Midpoint is 12
ForkJoinPool-2-worker-50 says :: Doing Right-search
ForkJoinPool-2-worker-57 says : After splliting the arry length is :: 13 Midpoint is 6
ForkJoinPool-2-worker-57 says :: Doing Right-search
ForkJoinPool-2-worker-50 says : After splliting the arry length is :: 7 Midpoint is 3
ForkJoinPool-2-worker-50 says :: Doing Right-search
ForkJoinPool-2-worker-57 says : After splliting the arry length is :: 4 Midpoint is 2
ForkJoinPool-2-worker-57 says :: Doing Right-search
ForkJoinPool-2-worker-43 says : After splliting the arry length is :: 2 Midpoint is 1
NOT FOUND !!!!!!!!!
 Element ::104 has been found in array? :: false
```