# ASSIGNMENT NO : A2

**1. TITLE :**

Using Divide and Conquer Strategies to design an efficient class for Concurrent Quick sort and the input data is stored using XML. Use object oriented software design method and Modelio / StarUML2.x Tool. Perform the efficiency comparison with any two software design methods. Use necessary USE-CASE diagrams and justify its use with the help of mathematical modeling. Implement the design using Scala/ Python/Java/C++.

**2. PREREQUISITES :**

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Java 1.7.0

**3. OBJECTIVE :**

- To learn the concept of Divide and Conquer Strategy.
- To study the design and implementation of Quick Sort algorithm

**4. INPUT:** I=Set of 8 quuens in 8*8 matrix

**5. OUTPUT:** O1=Success Case: It is the case when all the inputs are given by system are correctly and 8-Queen problem is solved.

O2=Failure Case: It is the case when the input does not match the validation Criteria.

**6. MATHEMATICAL MODEL:**

Let Z be a system={I,O,T,Sc,Fc}

I=Set of inputs.

O=Set of outputs.

T=Set of transitions function.

Sc=Success Case.

Fc=Failure Case.

I={matrix,8q.json}

O={matrix}

T={issafe,place}

Sc=solution found

Fc=solution not found.

## 7. THEORY :

### Divide and Conquer strategy:

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub- problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

### Quick sort:

The sorting algorithm invented by Hoare, usually known as "quicksort" is also based on the idea of divide-and-conquer. In Quick sort, input is any sequence of numbers and output is sorted array(here we will consider ascending order). We start with one number, mostly the first number, and finds its position in sorted array. This number is called pivot element. Then we divide the array into two parts considering this pivot elements position in sorted array. In each part, separately, we find the pivot elements. This process continues until all numbers are processed and we get the sorted array. In concurrent Quick sort, each part is processed by independent thread i.e. different threads will find out pivot element in each part recursively. Check in following diagram. Here PE stands for Pivot Element.

```
private static void quicksort(int[] arr, int low, int high)
{
    if (arr == null || arr.length == 0)
        return;
    if (low >= high)
        return;
    int middle = low + (high - low) / 2;
```

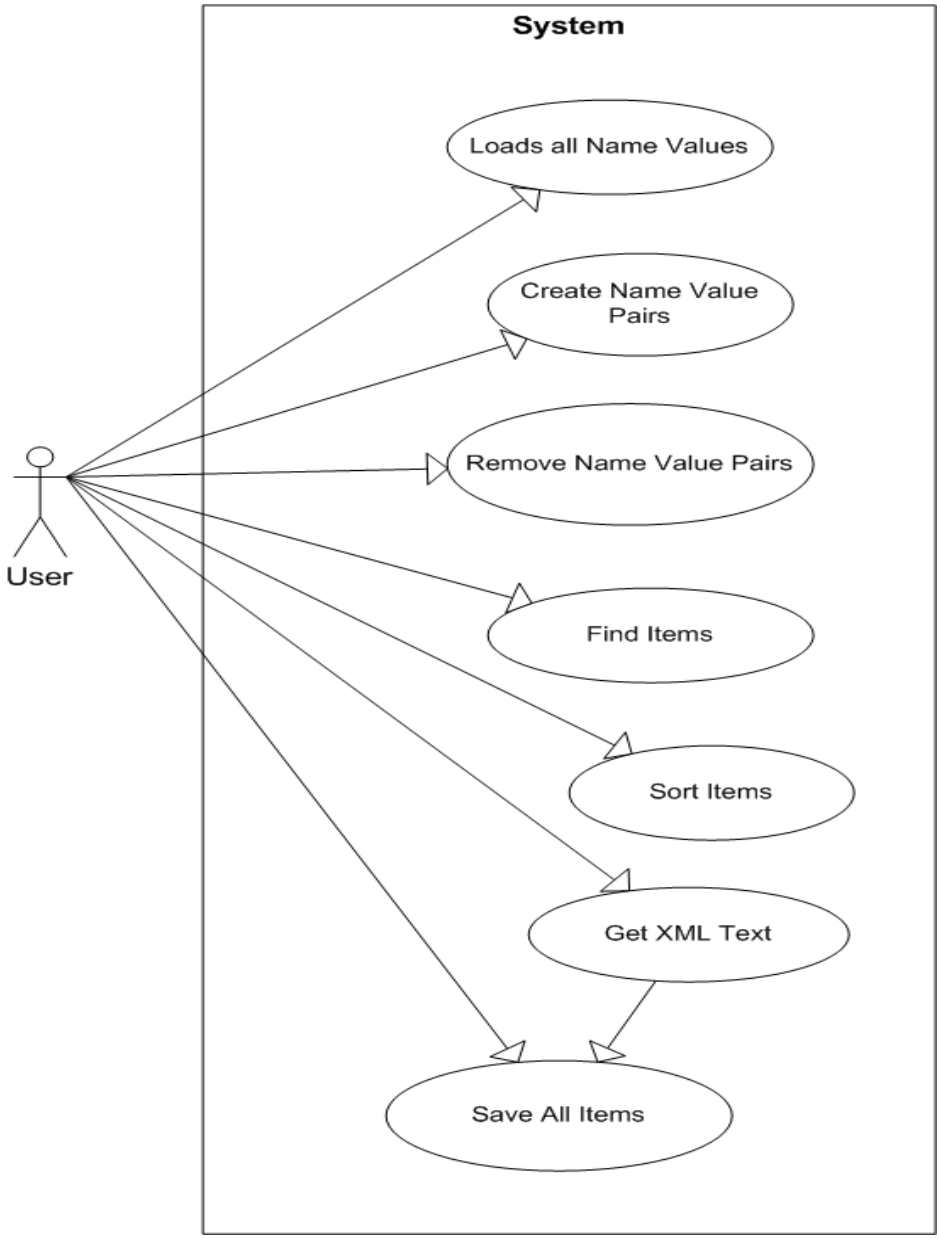
```
int pivot = arr[middle];
int i = low, j = high;
while (i <= j)
{
        while (arr[i] < pivot)
        {
                i++;
        }
while (arr[j] > pivot)
{
        j--;
}
if (i <= j) {
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
i++;
j--;
}
}
// recursively sort two sub parts
if (low < j)
quicksort(arr, low, j);
if (high > i)
quicksort(arr, i, high);
}
```

**Document Object Model:**

The **Document Object Model** (**DOM**) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents. The nodes of every document are organized in a tree structure, called the DOM tree. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API).

**Relative efficiency:**

| Class | Search algorithm |
|---|---|
| Worst case performance | **$O(N^2)$** |
| Best case performance | **O(NlogN)** |
| Average case performance | **O(NlogN)** |




Patterns to be followed in this assignment are FAÇADE pattern and command pattern which includes 2 tier architecture and client server architecture.

**CONCLUSION:**

Thus we have studied Concurrent Quick Sort using divide and conquer strategy.

Code

XML file input.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<data>
	<number>
		<value>88</value>
	</number>
	<number>
		<value>10</value>
	</number>
	<number>
		<value>34</value>
	</number>
	<number>
		<value>67</value>
	</number>
	<number>
		<value>56</value>
	</number>
</data>
```

```
import java.io.File;
import java.util.Arrays;

import org.w3c.dom.*;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;



import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;

public class QuickSort {

	/**
	* @param args
	*/
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int[] arr = {0,0,0,0,0};
		readFromXML(arr);
		System.out.println("Read data from xml" + Arrays.toString(arr));
		int low = 0;
		int high = arr.length - 1;
		quicksort(arr, low, high);
		System.out.println("Array after sorting" + Arrays.toString(arr));

	}

	private static void quicksort(int[] arr, int low, int high){
		// TODO Auto-generated method stub
		if (arr == null || arr.length == 0)
			return;
```

```
		if (low >= high)
			return;
		// pick the pivot
		int middle = low + (high - low) / 2;
		int pivot = arr[middle];
		// make left < pivot and right > pivot
		int i = low, j = high;
		while (i <= j)//joparyant i ha less ahe j peksha or j ha greter ahe i peksha
			{
				while (arr[i] < pivot){
					i++;
				}
				while (arr[j] > pivot){
					j--;
				}
			if (i <= j){
				int temp = arr[i];
				arr[i] = arr[j];
				arr[j] = temp;
				i++;
				j--;
			}
		}
		// recursively sort two sub parts
		if (low < j)
			quicksort(arr, low, j);
		if (high > i)
			quicksort(arr, i, high);
	}

	public static void readFromXML(int[] arr)
	{
		try {

			File fXmlFile = new File("input.xml");//1

			DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();//2
			DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();//3
			Document doc = dBuilder.parse(fXmlFile);//4
			doc.getDocumentElement().normalize();//5
			NodeList nList = doc.getElementsByTagName("number");//6
			for (int temp = 0; temp < nList.getLength(); temp++) {
				Node nNode = nList.item(temp);
				if (nNode.getNodeType() == Node.ELEMENT_NODE) {
					Element eElement = (Element) nNode;
					arr[temp] =
Integer.parseInt(eElement.getElementsByTagName("value").item(0).getTextContent());
				}
			}
	    } catch (Exception e) {
		    e.printStackTrace();
		}
	}
}
```

```
/*
cipher@blackfury-HP-eNVy:~/be-2/A2$ javac QuickSort.java

cipher@blackfury-HP-eNVy:~/be-2/A2$ java QuickSort
Read data from xml[88, 10, 34, 67, 56]
Array after sorting[10, 34, 56, 67, 88]

cipher@blackfury-HP-eNVy:~/be-2/A2$
*/



*/
```