ASSIGNMENT NO. A2

Problem Statement:

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++.

Learning Objectives:

1. To Study and Understand the concept of Quicksort.
2. To learn implementation of Concurrent Quicksort.

Learning Outcomes:

Learnt about the concurrent implementation of quicksort.

Software and Hardware Requirements:

1. 64-bit operating System(Linux)
2. Text Editor-gedit
3. Modeliio Software
4. g++ Compiler

Theory

Divide and Conquer Strategies

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

This divide and conquer technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quicksort, merge sort), multiplying large numbers (e.g.Karatsuba), syntactic analysis (e.g., top-down parsers), and computing the discrete Fourier transform (FFTs). The correctness of a divide and conquer algorithm is usually proved by mathematical induction, and its computational cost is often determined by solving recurrence relations.

Quicksort

Quicksort (sometimes called partition exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. Quicksort is a comparison sort, meaning that it can sort items of any type for which a less than relation (formally, a total order) is defined. Quick sort is a divide and conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub arrays. In the very early versions of quicksort, the leftmost element of the partition would often be chosen as the pivot element. Unfortunately, this causes worst-case behavior on already sorted arrays, which is a rather common usecase. The problem was easily solved by choosing either a random index for the pivot, choosing the middle index of the partition or (especially for longer partitions) choosing the median of the first, middle and last element of the partition for the pivot. Input array: L = [8, 3, 4, 1, 6, 11, 9] Sorted array: L = [1, 3, 4, 6, 8, 9, 11]

TinyXML

TinyXML is a small, simple, operating system independent XML parser for the C++ language. It is free and open source software, distributed under the terms of the license of zlib/libpng. The principal impetus for TinyXML is its size, as the name suggests. It parses the XML into a DOM-like tree. It can both read and write XML files. TinyXML is a simple, small, C++ XML parser that can be easily integrated into other programs.

What it does?

In brief, TinyXML parses an XML document, and builds from that a Document Object Model (DOM) that can be read, modified, and saved. XML stands for "eXtensible Markup Language." It allows you to create your own document markups. Where HTML does a very good job of marking documents for browsers, XML allows you to define any kind of document markup, for example a document that describes a "to do" list for an organizer application. XML is a very structured and convenient format. All those random file formats created to store application data can all be replaced with XML. One parser for everything. There are different ways to access and interact with XML data. TinyXML uses a Document Object Model (DOM), meaning the XML data is parsed into a C++ objects that can be browsed and manipulated, and then written to disk or another output stream. You can also construct an XML document from scratch with C++ objects and write this to disk or another output stream.

TinyXML attempts to be a flexible parser, but with truly correct and compliant XML output. TinyXML should compile on any reasonably C++ compliant system. It does not rely on exceptions or RTTI. It can be compiled with or without STL support. TinyXML fully supports the UTF-8 encoding, and the first 64k character entities.

1 Mathematical Model

System Description:

Let S be the Solution of the System representing the problem of.....such that ,

S = {s ,e ,I ,O ,F ,DD ,NDD ,S_c , F_c | ϕ }

s = Start state
={ Q_0 , IA }

Q_0 =
= { Null State of the system}

IA =
= { Initial empty array }

Input(I): X = Input to the System
= $\{U_D\}$ = Array of numbers to be sorted

Output(O): Y = Output of the System
= $\{S_D\}$ = Sorted array of numbers

where ,
S_D = Array of sorted elements from U_D
u_{iD} in U_D , where i is an integer $\leq U_D$

Functions :
F=Set of Functions = $\{f_{InputArray}, \quad f_{QuickSort}, \quad f_{Partition}, \quad f_{DisplayOutput}\}$

$f_{Inputarray}$ =Takes $U_D$as input and gives T_D,
$where$,
$T_D = t_i \forall u_i \in U_D where, i \in N.$

$f_{QuickSort} = Itcalls f_{Partition} if lengthof U_D \geqslant 2 elsecalls f_{DisplayOutput}$
$f_{Partition}$ = Partitions the array using pivot
$f_{DisplayOutput}$ =Function to display output

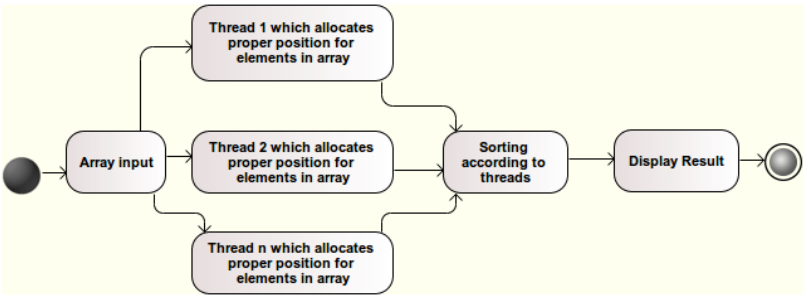
Success Conditions:
$\forall i \in N$,ideal .....from each u_{id} matches the $s_{id}, \forall u_{id} \in U_D$ and $\forall s_{id} \in S_D$.

$FailureConditions$:
Information ..... missed by $f_{QuickSort}$.

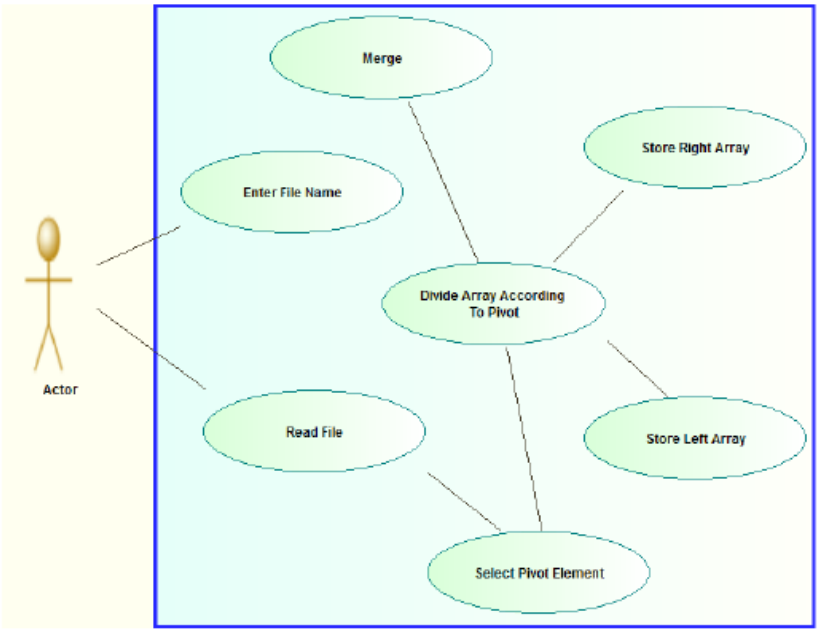
Algorithm

1. Pick an element, called a pivot, from the array.
2. Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way).
3. After this partitioning, the pivot is in its nal position. This is called the partition operation.
4. Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

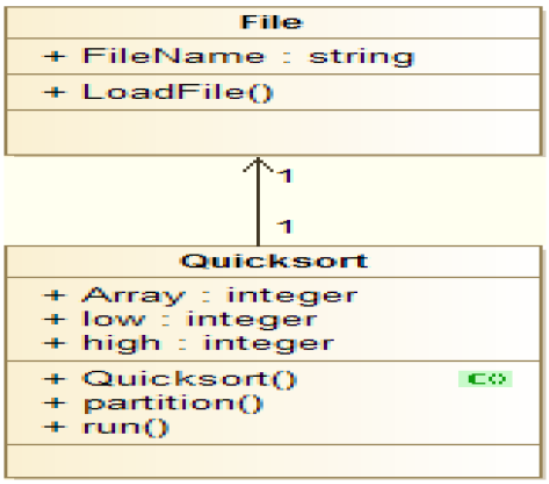
State diagram



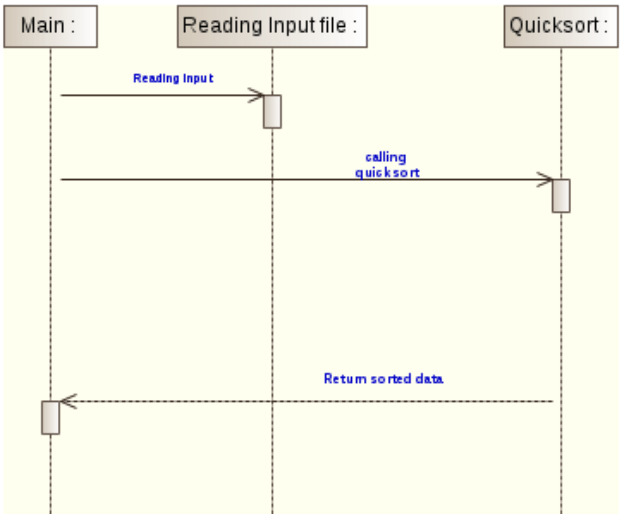
Use-case diagram



Class diagram



Sequence diagram



Positive Testing

Positive testing is a testing technique to show that a product or application under test does what it is supposed to do. Positive testing verifies how the application behaves for the positive set of data.

1. Array size to be specified in integer values, within specified limits.
2. Array elements should be integer values.

Sr. No.	Test Condition	Steps to be executed	Expected Result	Actual result
1.	Entered size of array as int	Press enter	Enter the elements and sorted array	Sorted array
2.	To sort array	Press enter	Sorted array	Sorted array

Negative Testing

Negative testing ensures that your application can specifically handle invalid input or unexpected user behavior.

1. Size of array which exceeds its limit.
2. Any data type other than integer for array size and elements.

Sr. No.	Test Condition	Steps to be executed	Expected Result	Actual result
1.	Entered size of array as character or float	Press enter	Error message	Same as expected
2.	"Enter" without giving size of array	Press enter	Error message	Same as expected

Conclusion

We have successfully implemented Concurrent quick sort using divide and conquer technique using tinyxml.