PIMPRI CHINCHWAD COLLEGE OF ENGINEERING
COMPUTER LABORATORY - IV

Assignment No - A2

1 Aim :

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 Objective :

- To study and implement Quick Sort algorithm concurrently.
- To use object oriented software design method.
- To design necessary UML diagrams.

3 Mathematical Model :

This problem can be mathematically modelled as follows, Consider a set S consisting of all the elements related to a program.The mathematical model is given as below,

S={s,e,X,Y,Fme}

Where,
s = Initial State
e = End State
X = Input.

- x={Unsorted Elements}

Y = Output.

- y={Sorted Elements}

Fme = Algorithm

```
int createPartition(int elements[], int begin, int end)
{
int temp, temp1;
int x = elements[end];
int i = begin - 1;
for(int j = begin; j¡= end - 1; j++)
if(elements[j] ¡= x)
i = i+1;
temp = elements[i];
elements[i] = elements[j];
elements[j] = temp;
}
}
temp1 = elements[i + 1];
elements[i + 1] = elements[end];
elements[end] = temp1;
return i + 1;
}
```

4 Theory :

4.1 Divide and Conquer strategy :

- Given a function to compute on an inputs the divide and conquer strategy suggest splitting the input into k distinct input sets ,such that 1¡k¡n yielding k sub problems.
- These sub problems must be solved and then method must be found to combine the sub solution into a solution of a whole.
- If the sub problems are relatively large then the divide and conquer strategy can be reapplied.
- Often the sub problems are of the same type as the same original problem for which the reapplication of divide and conquer principle is naturally expressed by a recursive algorithm.

4.2 Quick Sort Algorithm :

4.2.1 Features of Quick Sort :

- Similar to mergesort - divide-and-conquer recursive algorithm
- One of the fastest sorting algorithms
- Average running time O(NlogN)
- Worst-case running time O(N2)

4.2.2 Basic idea of Quick Sort :

1. Pick one element in the array, which will be the pivot.
2. Make one pass through the array, called a partition step, re-arranging the entries so that:
    - the pivot is in its proper place.
    - entries smaller than the pivot are to the left of the pivot.
    - entries larger than the pivot are to its right.
3. Recursively apply quicksort to the part of the array that is to the left of the pivot, and to the right part of the array.

4.2.3 Algorithm :

We will dicuss the quicksort algorithm in detail,

1. Choosing the pivot
   Choosing the pivot is an essential step. Depending on the pivot the algorithm may run very fast, or in quadratic time.:

   (a) Some fixed element: e.g. the first, the last, the one in the middle,this is a bad choice - the pivot may turn to be the smallest or the largest element, then one of the partitions will be empty.

   (b) Randomly chosen (by random generator) - still a bad choice.

   (c) The median of the array (if the array has N numbers, the median is the [N/2] largest number. This is difficult to compute - increases the complexity.

   (d) The median-of-three choice: take the first, the last and the middle element. Choose the median of these three elements.

   Example:
   8, 3, 25, 6, 10, 17, 1, 2, 18, 5
   The first element is 8, the middle - 10, the last - 5.
   The median of [8, 10, 5] is 8

2. Partitioning
   Partitioning is illustrated on the above example.

   (a) The first action is to get the pivot out of the way - swap it with the last element
   5, 3, 25, 6, 10, 17, 1, 2, 18, 8

   (b) We want larger elements to go to the right and smaller elements to go to the left.Two "fingers" are used to scan the elements from left to right and from right to left:

   ```
   [5, 3, 25, 6, 10, 17, 1, 2, 18, 8]
    ^                              ^
    i                              j
   ```

   i. While i is to the left of j, we move i right, skipping all the elements less than the pivot. If an element is found greater then the pivot, i stops

ii. While j is to the right of i, we move j left, skipping all the elements greater than the pivot. If an element is found less then the pivot, j stops

iii. When both i and j have stopped, the elements are swapped.

iv. When i and j have crossed, no swap is performed, scanning stops, and the element pointed to by i is swapped with the pivot .In the example the first swapping will be between 25 and 2, the second between 10 and 1.

(c) Restore the pivot. After restoring the pivot we obtain the following partitioning into three groups:
$[5, 3, 2, 6, 1][8][10, 25, 18, 17]$

3. Recursively quick sort the left and the right parts

4.3 Software Functions :

Here is the function, that implements the partitioning.
left points to the first element in the array currently processed, right points to the last element.

```
if( left + 10 $<$= right)
{
int i = left , j = right - 1;
for ( ; ; )
{
while (a[++i] $<$ pivot  ) {}  // move the left finger
while (pivot  $<$ a[--j] ) {}  // move the right finger
if (i $<$ j) swap (a[i],a[j]);  // swap
else  break;      // break if fingers have crossed
}
swap (a[I], a[right-1]);   // restore the pivot
quicksort ( a, left , i-1);  // call quicksort for the
                             //left part
quicksort (a, i+1, right);// call quicksort for the
                            //left part
}
else
insertionsort (a, left , right);
```

If the elements are less than 10, quicksort is not very efficient. Instead insertion sort is used at the last phase of sorting.

4.3.1 Implementation notes:

Compare the two versions:

A. while (a[++i] < pivot) while (pivot < a[–j])
   if (i < j) swap (a[i], a[j]); else break;

B. while (a[i] < pivot) i++; while (pivot < a[j]) j–;
   if (i < j) swap (a[i], a[j]); else break;

If we have an array of equal elements, the second code will never increment i or decrement j, and will do infinite swaps. i and j will never cross.

4.4 Concurrency in Quick sort :

Concurrent Quicksort

Simple concurrent implementation uses a collection of worker threads and a coordinator thread.The coordinator sends a message to an idle worker telling it to sort the array and waits to receive messages from the workers about the progress of the algorithm.

A worker partitions a sub-array, and every time that worker gets ready to call the partition routine on a smaller array, it checks to see if there is an idle worker to assign the work to. If so, it sends a message to the worker to start working on the sub-problem; if not the current worker makes calls the partition routine itself.

After each partitioning, two recursive calls are (usually) made, so there are plenty of chances to start other workers.The diagram below shows two workers sorting the same 5-element array. Each blue line represents the flow of control of a worker thread, and the red arrow represents the message sent from one worker to start the other. (Since the workers proceed working concurrently, it is no longer guaranteed that the smaller elements in the array will be ordered before the larger; what is certain is that the two workers will never try to manipulate the same elements.)

A worker can complete working either because it has directly completed all the work sorting the subarray it was initially called on, or because it has ordered a subset of that array but has passed some or all of the remaining work to other workers.In either case, it reports the number of elements it has ordered back to the coordinator.(The number of elements a worker has ordered is the number of partitions of sub-arrays that have 1 or more members).

When the coordinator hears that all the elements in the array have been ordered, it tells the workers that there is nothing left to do, and the workers exit. Thats the basic idea.

Multithreading in C++ :
Demo Example:
Create a function that you want the thread to execute. I'll demonstrate with a trivial example:
void task1(std::string msg)
{
std::cout << "task1 says: " << msg;
}
Now create the thread object that will ultimately invoke the function above like so: std::thread t1(task1, "Hello");
(You need to #include <thread> to access the std::thread class)
As you can see, the constructor's arguments are the function the thread will execute, followed by the function's parameters.
Finally, join it to your main thread of execution like so:
t1.join();
(Joining means that the thread who invoked the new thread will wait for the new thread to finish execution, before it will continue it's own execution).

5 Testing

5.1 BLACK BOX TESTING :

Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle.This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.
Black box testing techniques are :
1) Equivalence Class Partitioning
2) Boundary Value Analysis
3) Decision Tables
4) State Transition Diagrams (or) State Transition Diagrams
5) Orthogonal Arrays
6) All Pairs Technique

5.2 WHITE BOX TESTING :

White Box Testing (WBT) is also known as Code-Based Testing or Structural Testing. White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.In this method of testing the testcases are calculated based on analysis internal structure of the system based on Code coverage, branches coverage, paths coverage, condition Coverage etc. Typically such method are used at Unit Testing of the code but this different as Unit testing done by the developer & White Box Testing done by the testers, this is learning the part of the code & finding out the weakness in the software program under test. For tester to test the software application under test is like a white/transparent box where the inside of the box is clearly seen to the tester (as tester is aware/access of the internal structure of the code), so this method is called as White Box Testing.

The White-box testing is one of the best method to find out the errors in the software application in early stage of software development life cycle. In this process the deriving the test cases is most important part. The test case design strategy include such that all lines of the source code will be executed at least once or all available functions are executed to complete 100 percent code coverage of testing. For this, we will use Flow Graphs. Flow graphs are, Syntactic abstraction of source code Resembling to classical flow charts Forms the basis for white box test case generation principles.Conventions of flow graph notation.

Why and When White-Box Testing:

White box testing is mainly used for detecting logical errors in the program code. It is used for debugging a code, finding random typographical errors, and uncovering incorrect programming assumptions . White box testing is done at low level design and implementable code. It can be applied at all levels of system development especially Unit, system and integration testing. White box testing can be used for other development artefacts like requirements analysis, designing and test cases . White box testing techniques are:

1. Static white box testing

a. Desk checking

b. Code walkthrough

c. Formal Inspections

2. Structural White box testing

a. Control flow/ Coverage testing

b. Basic path testing

c. Loop testing
d. Data flow

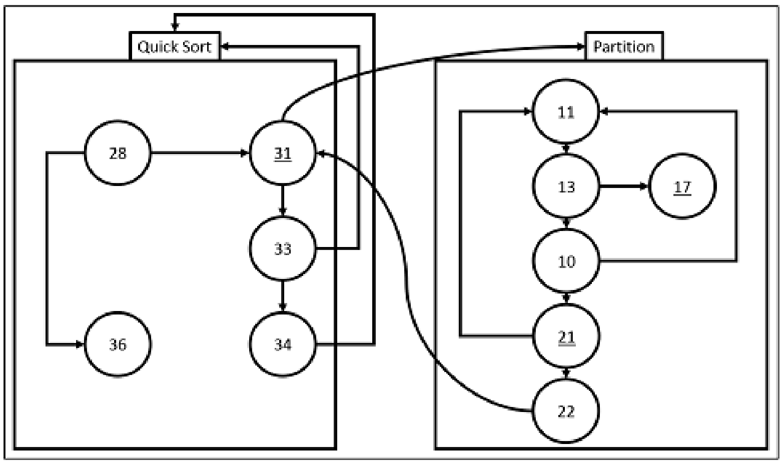


Figure 1: Flow Graph

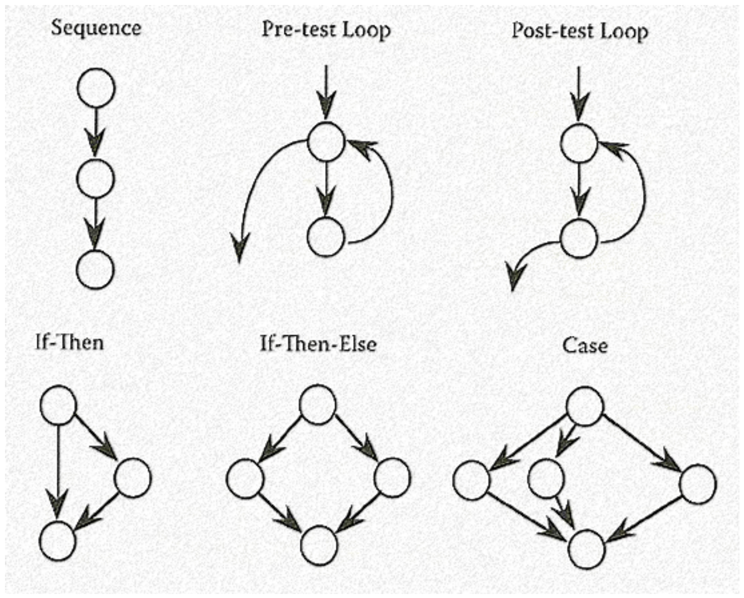


Figure 2: Flow Graph Notation

5.3 Class Diagram

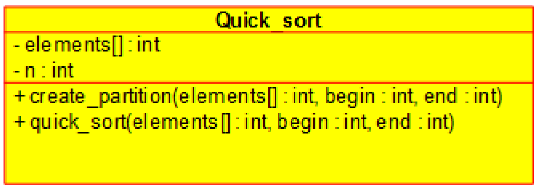

Quick_sort
- elements[] : int
- n : int
+ create_partition(elements[] : int, begin : int, end : int)
+ quick_sort(elements[] : int, begin : int, end : int)

Figure 3: Class Diagram

5.4 Use Case

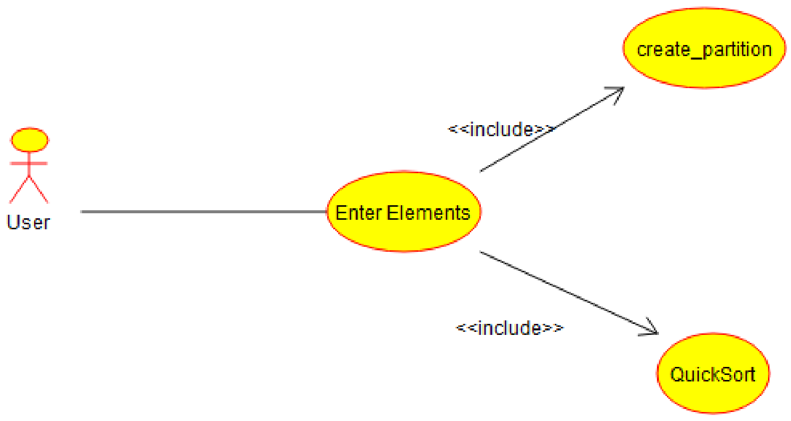


Figure 4: Use Case Diagram

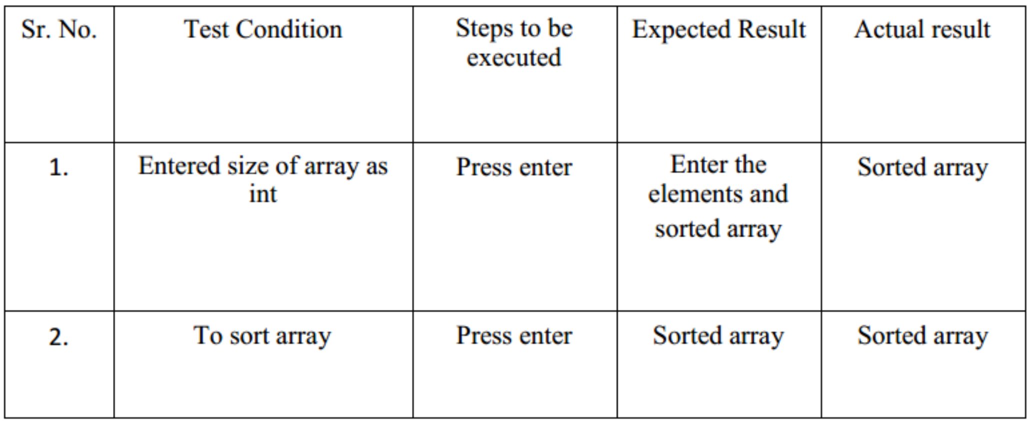
5.5 POSITIVE/NEGATIVE TESTING

Example:
Sample Input: For integer array
int A[] = f21; 33; 67; 72; 27; 10; 2; 66; 81; 1g; Function is passed following arguments:
quickSort(A; 0; 9);
Output obtained:
New partition: Low=0 High=9
Swapped 21 with Pivot 1

```
New partition: Low=1 High=9
Swapped 33 with 10
Swapped 67 with 2
Swapped 72 with Pivot 21
New partition: Low=1 High=2
Swapped 10 with Pivot 2
New partition: Low=4 High=9
Swapped 27 with 27
Swapped 33 with 33
Swapped 67 with 67
Swapped 66 with 66
Swapped 81 with Pivot 72
New partition: Low=4 High=7
Swapped 27 with 27
Swapped 33 with 33
Swapped 67 with Pivot 66
New partition: Low=4 High=5
Swapped 27 with 27
Swapped 33 with Pivot 33
1,2,10,21,27,33,66,67,72,81,
```

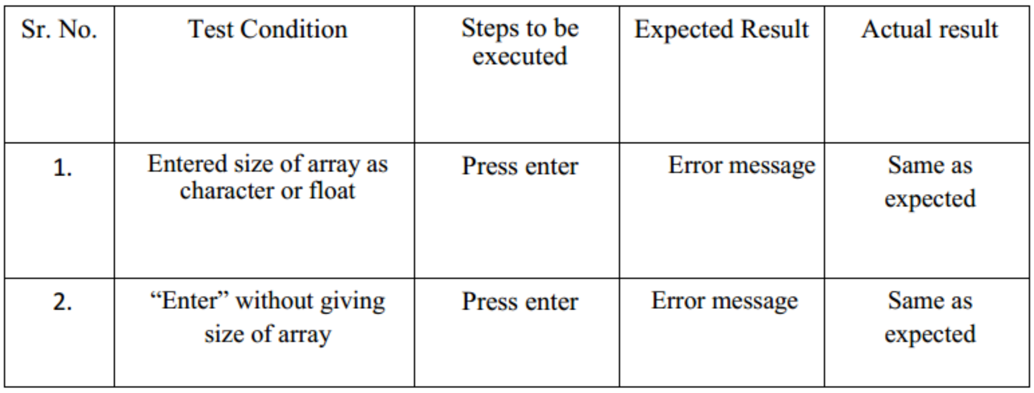
Note:
The underlined nodes are the ones being tested. The above output shows that every test region is covered for given input.

Positive Testing :



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 :



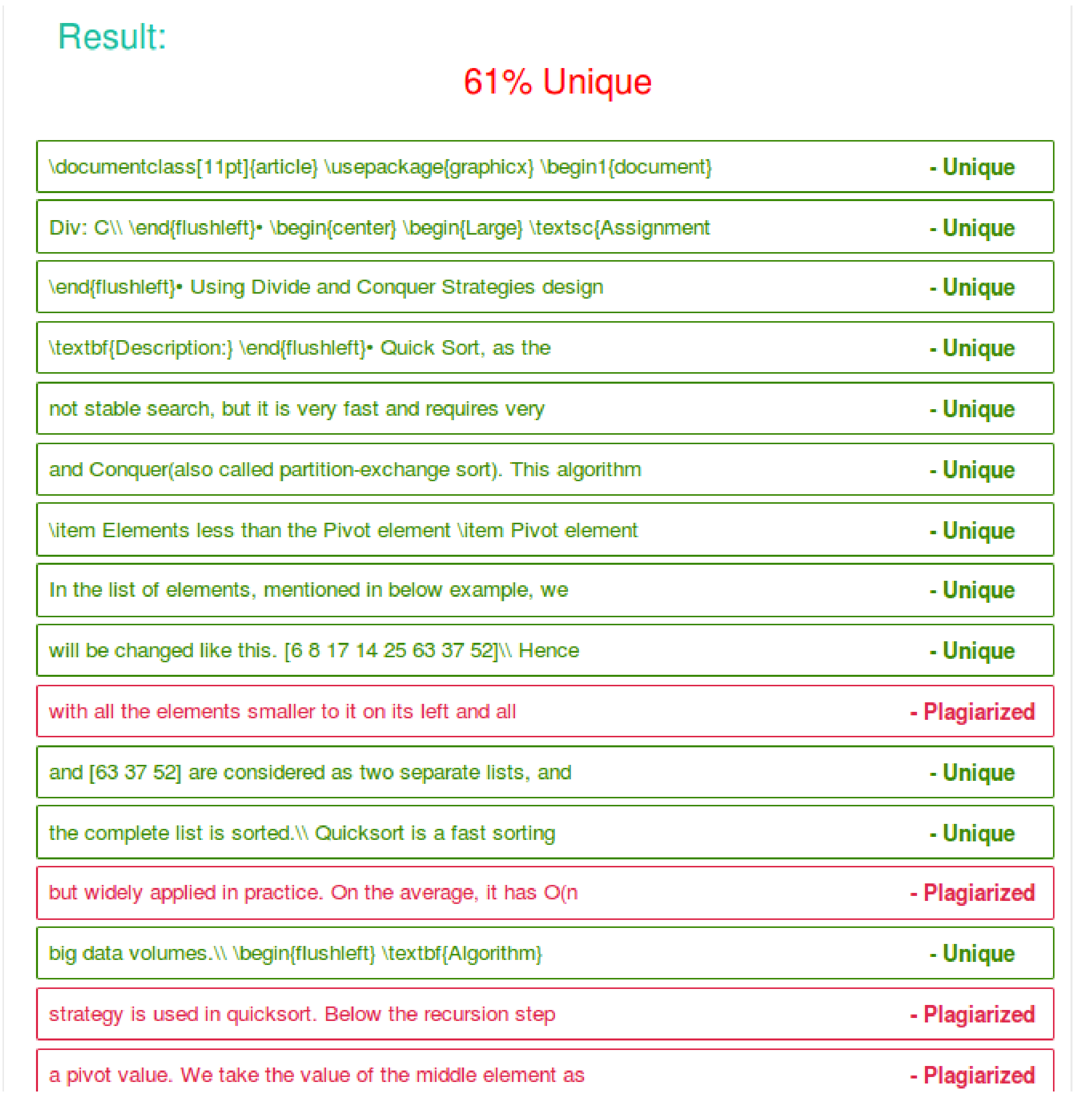
Sr. No.	Test Condition	Steps to be executed	Expected Result	Actual result
1.	Entered size of array as character or float	Press enter	Error message	Same as expected
2.	“Enter” without giving size of array	Press enter	Error message	Same as expected

6 Conclusion :

We have succesfully implemented quick sort concurrently, using divide and conquer strategy, we have used object oriented software design method and Modelio Tool

Roll No.	**Name of Student**	**Date of Performance**	**Date of Submission**
302	Abhinav Bakshi	21/12/15	04/01/16

7 Plagarism Report



Result:

61% Unique

\documentclass[11pt]{article} \usepackage{graphicx} \begin1{document}	- Unique
Div: C\\ \end{flushleft}• \begin{center} \begin{Large} \textsc{Assignment	- Unique
\end{flushleft}• Using Divide and Conquer Strategies design	- Unique
\textbf{Description:} \end{flushleft}• Quick Sort, as the	- Unique
not stable search, but it is very fast and requires very	- Unique
and Conquer(also called partition-exchange sort). This algorithm	- Unique
\item Elements less than the Pivot element \item Pivot element	- Unique
In the list of elements, mentioned in below example, we	- Unique
will be changed like this. [6 8 17 14 25 63 37 52]\\ Hence	- Unique
with all the elements smaller to it on its left and all	- Plagiarized
and [63 37 52] are considered as two separate lists, and	- Unique
the complete list is sorted.\\ Quicksort is a fast sorting	- Unique
but widely applied in practice. On the average, it has O(n	- Plagiarized
big data volumes.\\ \begin{flushleft} \textbf{Algorithm}	- Unique
strategy is used in quicksort. Below the recursion step	- Plagiarized
a pivot value. We take the value of the middle element as	- Plagiarized

Figure 5: Plagarism Report