Assignment No.2

1 Aim

To understand Divide and Conquer strategy for Quick Sort

2 Problem Definition

Using Divide and Conquer Strategies design a class for Concurrent Quick Sort using C++.

3 Learning Objective

1. Able to understand, divide and conquer strategies.
2. To learn concurrent programming using Open MP.

4 Learning Outcome

After successfully completing this assignment, we would be able to Understand Implement Concurrent Quick sort using Divide and Conquer strategy.

5 Theory

Divide and Conquer Strategy

Quick sort can be effectively performed through divide and conquer strategy, which reduces searching time. The strategy involves three steps at each level of recursion.

The name ”divide and conquer” is sometimes applied to algorithms that reduce each problem to only one sub-problem, such as the binary search algorithm for finding a record in a sorted list. These algorithms can be implemented more efficiently than general divide-and-conquer algorithms; in particular, if they use tail recursion, they can be converted into simple loops. Under this broad definition, however, every algorithm that uses recursion or loops could be regarded as a ”divide and conquer algorithm”. Therefore, some authors consider that the name divide and conquer” should be used only when each problem may generate two or more sub problems. The name decrease and conquer has been proposed instead for the single-sub problem class.

In computer science, divide and conquer 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 subproblems of the same 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. An important application of decrease and conquer is in optimization, where if the search space is reduced by a constant factor at each step, the overall algorithm has the same asymptotic complexity as the pruning step, with the constant depending on the pruning factor; this is known as prune and search.

Quick Sort Algorithm

Quick sort is an efficient sorting algorithm,serving as a systematic method for placing the elements of an array in order. Developed by Tony Hoare in 1959, with his work published in 1961, it is still a commonly used algorithm for sorting. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heap sort. Quick sort is a comparison sort, meaning that it can sort items of any type for which a ”less-than” relation is dened. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quick sort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.

Algorithm

The strategy involves three steps at each level of recursion.

1. Divide:- Divide the problem into a number of sub problems.
2. Conquer: - Conquer the sub problems by solving themrecursively. If the sub problem sizes are small enough, then just solve the sub problems in a straight forward manner.
3. Combine: - Combine the solutions to the sub problems toform the solution for the original problem.

5 Mathematical Model :

Consider S as a system S= {s, e, x, y, f, fc, Div, DD, NDD | φ}

s =start state={φ}

e = end state {0 | 1} *where*

0 *represents sorted,* 1 *represents not sorted.*

X = L, n
L = list of unsorted numbers
xi — xi N xi ¡ xi+1 —— xi ¿ xi+1 —— xi = xi+1 where i = 1, 2, 3,.., n

p=pivot element p=xp — xp x
INPUT: f(L) Y
PERMUTE=p — p is a permutation of x
PRO(p)=x — x is a subsequence of p p PERMUTE
PSPACE= ł PRO(p) ł empty problem p PERMUTE
empty sequence
Div: PSPACE–¿PSPACE X PSPACE

S.T Div() =(,) Div (p)=(p1,p2) S.T —p—=—p1—+—p2—+1 p1(l)=p(l) p2(r)=p(r) p1(r)¡p2(l) x p1 ,yp2 , x¡=y

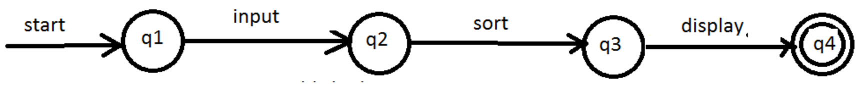
Assumption: pivot (p) provides pivot of problem P which is assumed to rightmost element of p. Div(p)= (,) if —p—=1

Output:
Y=sorted list

Y = X n/2, X p, X n/2 S.T elements of(X n/2)¡ (X p) ¡ elements of (X n/2)

Success:
Successful output represent list which is sorted.

6 State Diagram



q1 : start state
q2: accept input
q3: soring
q4: display and end state

7. Conclusion

Thus, we the help of divide and conquer strategy and OOPS concepts, concurrent Quick Sort is implemented in Cpp.