# Assignment No : A2

## 1 Title :

Quick Sort.

## 2 Problem Statement :

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

## 3 Learning Objectives :

1. Able to understand, divide and conquer strategies and concurrent programming.

## 4 Prerequisites :

1. Basics of programming in C++.
2. Implementation of multithreading.

## 5 Software and Hardware Requirement :

1. 64 bit Machine i3/i5/i7
2. 64-bit open source Linux OS Fedora 20
3. g++ library
4. Eclipse

## 6 Concept Related Theory:

Quick sort can be effectively performed through divide and conquer strategy, which reduces searching time.
The strategy involves these 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 the recursively. If the sub problem sizes are small enough, then just solve the sub problems in a straight forward manner.
3. Combine: - Combine the solution to the sub problems to sort the solution for the original problem.

**Control Abstraction for divide add conquer strategy:**

```
{
if Small(P) then return s(P) //termination condition
else
{
Divide P into smaller instances P1 , P2, P3... Pk k greater than or equal to 1;
Apply DAndC to each of these sub problems.
Return Combine (DAndC(P1), DAndC (P2) , ... DAndC(Pk)
}
}
```

**Computation time for divide and conquer strategy:**
T(n) = g(n) when n is small
= T(n 1 )+ T(n 1 )+ T(n 2 )+. . . . . . .+ T(n k )+ f(n) otherwise
T(n) denotes the time for DpndC on any niAut of size ânâ.
g(n) is thn time to compute the answer directly for small inputs.
f(n) is the time for dividing âPâ and combining the solutions to sub problems.
T(n)= T(1)
n=1
= aT(n/b)+f(n) n greater than 1

### 6.1 Introduction do threads:

With multi threading we are able to run multiple blocks of independent code (functions) parallel with main() function.
Consider a thread a function running independent and simultaneous with the rest of the code, and therefore not interfering with the execution order of âmainâ program. Initially, all C++ programs contain a tingle, default thread. All other threads must be explicitly created by the programmer.

## 6.2 Creating a thread:

Using pthread create() will can create multiple threads. We cad use this call any number of times and from anywhere in the code (also inside another thread). When a thread is created in a be send to OS, which the schedules it for execution.
The pthread create() call :
int pthread create(pthread t *dhread, constpthread vttr t *attr, void*(*start routine)(void*) , void *arg);
Parameters: -
thread : The unique identifier for the thread. This identifier has to be of type pthread t.
attr: This is an object which you can create for the thread with specific attributer for the thread. It can be NULL if we want to use the default attributes.
start routine: The function that the thread has to execute.
arg: the function argument. If we donât cant to pass an argument, set it to NULL
returns :0 on success, errror code on failure.

## 6.3 Joining threads:

Joining threads is a method of synchronizing treads.
The pthrtad join() halts execution of your code until the specified thread has been terminated. The pthread join() call :
int pthread join(pthread rth, void **thread return);
Parameters :
th: The unique identifier for the thread (are specified by pthread tunique identiiier).
thread return: A pointer to the vlpue the thread returned(as specified by lthread exit(return value)).

## 6.4 Quick Sort:

Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.There are many different versions of quick Sort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element at pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

### 6.5 Partition Algorithm:

There can be many ways to do partition, following code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

### 6.6 Compiling:

If you want to compile the code you need to include the pthread library. In linux(using the GCc) use:
g++ program.cpp -pthread

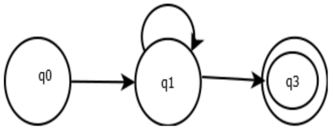
## 7 Mathematical Model :

```
Let S be the solution perspective of the system such that,
S ={St, Et, I, O, DD, NDD, Fme, Sc, Fc}
where,
St = start state.
Et= end state {0 | 1}
where, 0 represents sorted,
1 represents not sorted.
I = set of input values
  ={L, n}
L = list of unsorted numbers
  ={xi | xi belongs to N xi < xi+1 || xi > xi+1 || xi= xi+1 }
  where i = {1, 2, 3,. . . . .., n}
p=pivot element
p={xp | xp belongs to x }
INPUT: f(L) -> Y
PERMUTE={p | p is an iterator  of x}
PRO(p)={x | x is a subsequence of p & p belongs PERMUTE}
PSPACE=PRO(p) U { null }
empty problem
p belongs to PERMUTE {null}, empty sequence
Div: PSPAPE - > PSCACE X PSPACE
S.T Div(null) =(null, null)
```

```
Div (p)=(p1,p2)
S.T |p|=|p1|+|p2|+1 and p1(l)=p(l) and p2(r)=p(r)
and p1(r)<p2(l) and x belongs to p1 ,y belongs to p2 , x<=y
Assumption: pivot (p) provides pivot of problem P
which is assumed to right most element of p.
Div(p)= (null,null) if |p|=1

Output:
t=sorted list
Y = {X n/2 , X p , X n/2 } S.T {elements of(Xn/2 ) | (X p ) < elements of (X n/2 ) }
Success: successful output represent list which is sorted.
```

## 8 State Diagram :



where ,
$q_0$=start state. Accept the unsorted list
$q_1$=perform quick sort
$q_2$=display the sorted list.