# QUICKSORT ALGORITHM

***PROBLEM STATEMENT* :**

Implement a Concurrent and Threaded version of Quicksort algorithm and compare the performance of Concurrent, Threaded and Normal Quicksort.

DESCRIPTION OF THE QUICKSORT ALGORITHM: Quicksort is a Divide and Conquer sorting algorithm. It is one of the most efficient sorting algorithms and is based on the idea of splitting an array into smaller ones. The algorithm first randomly picks an element called a pivot and partitions the array into a low subarray, elements smaller than the pivot and a high subarray, elements greater than the pivot. This step is called the partitioning step. Quicksort algorithm recursively applies the above step to the sub-arrays, resulting in sorting the array.

Given a number ‘n’ and n numbers to be sorted as the input, we have to sort the numbers using various Quicksort versions. The code has a function named:

- runSorts() - Runs all the sorts and stores the running time of each version of sort.

### IMPLEMENTING CONCURRENT QUICKSORT THROUGH PROCESSES

Function in the code : concurrent_quicksort

(i) Recursively make two child processes using fork system calls(*left_pid1* and *right_pid2*), one for the left subarray, one of the right subarray. If the number of elements in the array for a process is less than 5, we have to perform a Insertion Sort.

(ii) Shmget(for shared memory allocation) and Shmat(for shared memory operations) functions are for accessing the shared memory.

### IMPLEMENTING QUICKSORT THROUGH THREADS

Function in the code : threaded_quicksort

(i) Two threads are made recursively, one of which will sort the low subarray(*pthread_t tid1*) and the other will sort the high subarray(*pthread_t tid2*).

(ii) pthread_create and pthread_join functions are used.

## OBSERVATIONS

1. Similarly, concurrent sort takes more time than normal sort due to the creation of a lot of processes which increase the number of page faults, context switches and CPU migration.

2. Randomized Quicksort with threading takes much more time due to the creation of too many threads. The SegFault at large input therefore occurs due to the creation of too many threads (O(log(n)) number of threads).

3. Threaded Sort takes less time than Concurrent Sort as threads share memory and caches but for concurrent sort, we are creating two processes which require the os to make virtual memory and Shared Memory region and accessing data in that region takes time.