Report of Question 1:

Concurrent Quicksort using Processes and Threads:

Creating shared memory of size ‘size’ using function shareMem:

```
int *shareMem(size_t size)
{
    key_t mem_key=IPC_PRIVATE;
    int shm_id=shmget(mem_key,size,IPC_CREAT | 0666);
    return (int*)shmat(shm_id,NULL,0);
}
```

Partition Function:

Takes the element at position = pivot and partitions the array such that all elements less than or equal to arr[pivot] are to the left of the pivot element and all elements greater than pivot are to the right:

```
int partition(int *arr,int l,int r,int pivot)
{
    int i=l-1;
    swap(&arr[pivot],&arr[r]);
    pivot=arr[r];
    for(int j=l;j<r;j++)
    {
        if(arr[j]<=pivot)
        {
            i++;
            swap(&arr[i],&arr[j]);
        }
    }
    i++;
    swap(&arr[i],&arr[r]);
    return i;
}
```

Quicksort using Processes:

In normal quicksort we call the function recurrsively for the left and right halves and wait for them. In Multiprocess quicksort we create 2 new processes using the fork() function and sort the left and right halves using each of these processes.

```
int pid1=fork(),pid2;
if(pid1<0)
{
    printf("Forking Error\n");
    return NULL;
}
if(pid1==0)
{
    quicksort_processes(args1);
    _exit(1);
}
else
{
    pid2=fork();
    if(pid2<0)
    {
        printf("Forking Error\n");
        return NULL;
    }
    if(pid2==0)
    {
        quicksort_processes(args2);
        _exit(1);
    }
    else
    {
        int status;
        waitpid(pid1,&status,0);
        waitpid(pid2,&status,0);
    }
}
```

Quicksort using Threads:

We create two new threads in each step of the quicksort algorithm and let them sort the left and right halves simultaneously.

```
int pos=partition(arr,l,r,pivot);
struct data *args1=malloc(sizeof(struct data)),*args2=malloc(sizeof(struct data));
pthread_t p1,p2;
args1->arr=arr;
args1->l=pos+1;
args1->r=r;
args1->pivot=randompos(pos+1,r);
pthread_create(&p1,NULL,quicksort_threads,args1);
args2->arr=arr;
args2->l=l;
args2->r=pos-1;
args2->pivot=randompos(l,pos-1);
pthread_create(&p2,NULL,quicksort_threads,args2);
pthread_join(p1,NULL);
pthread_join(p2,NULL);
```

Output:

Prints the time taken by Multithreaded quicksort , Multiprocess quicksort , and Normal quicksort respectively.

```
10
9 4 6 7 8 5 4 3 1 2
Running threaded_concurrent_quicksort
time = 0.001195
Running process_concurrent_quicksort
time = 0.002046
Running normal_concurrent_quicksort
time = 0.000009
```

Time Comparison:

N	Normal Quicksort	Multiprocess Quicksort	Multithreaded Quicksort
10	0.000008	0.002245	0.001129
100	0.000043	0.021464	0.006803
1000	0.000179	0.145930	0.046581
10000	0.002136	2.406056	0.117228
100000	0.019219	Forking Error	Segmentation Fault

For N = 100000 , it was not possible to create so many O(log(n)) threads and processes.