Comparison Of Quick Sort with different implementations

Using Concurrent Quicksort using multiple processes,i.e.,using fork().
Using Concurrent Quicksort using multiple threads,i.e.,using threads.
Using Quicksort normally.

Input:
Enter a number.
An array will be created consisting of numbers from n to 1.

Sorting will be done by all three methods and will be checked by sorting using inbuilt function to check the correctness of the code.

We will use appropriate time related algorithms to observe the running of each of the 3 implementations.

And will be displayed with comparison at the end.

```
struct arg{
    int l;
    int r;
    int* A;
};

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);
}
```

Shared memory for processes.

```
int B[1000000+1];




void swap(int *a,int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

```
int part(int A[],int l,int r)
{
    int x = A[r];
    int i = l-1;
    for(int j=l;j<=r;++j)
    {
        if(A[j]<x)
        {
            i++;
            swap(&A[i],&A[j]);
        }
    }
    i++;
    swap(&A[i],&A[r]);
    return i;
}
```

Partition function for find pivot for quicksort.

```
int compare(const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

void quickSort(int A[], int l, int r)
{
    if (l < r)
    {
        if(r-l+1<=5){            for(int i=l;i<=r;i++)   for(int j=i+1;j<=r;j++)        if(A[j]<A[i])
swap(&A[j],&A[i]);}
        else
        {
            int p = part(A, l, r);
            quickSort(A, l, p - 1);
            quickSort(A, p + 1, r);
        }
    }
}
```

Normal Quicksort

```
void doqsort(int *A, int l, int r)
{
    if(l < r)
    {
        if(r-l+1<=5){            for(int i=l;i<=r;i++)       for(int j=i+1;j<=r;j++)         if(A[j]<A[i])
swap(&A[j],&A[i]);}
        else
        {
            int status;

            int pid1,pid2;
            int p = part(A,l,r);
            pid1 = fork();
            if(pid1==0){
                doqsort(A, l, p-1);
                _exit(1);
            }
            else{
                pid2 = fork();
                if(pid2==0){
                    doqsort(A,p+1,r);
                    _exit(1);
                }
                else{
                    waitpid(pid1, &status, 0);
                    waitpid(pid2, &status, 0);
                }
            }
        }
    }
}
```
Concurrent Quicksort using multiple Processes

```
void *threaded_doqsort(void* a){
    struct arg *args = (struct arg*) a;

    int l = args->l;
    int r = args->r;
    int *A = args->A;
    if(l < r)
    {
        if(r-l+1<=5){    for(int i=l;i<=r;i++)       for(int j=i+1;j<=r;j++)         if(A[j]<A[i])
swap(&A[j],&A[i]);}
        else
        {
            pthread_t tid1,tid2;
            int p = part(A,l,r);
            struct arg a1;
            a1.l = l;
            a1.r = p-1;
            a1.A = A;
            pthread_create(&tid1, NULL, threaded_doqsort, &a1);

            struct arg a2;
            a2.l = p+1;
            a2.r = r;
            a2.A = A;
            pthread_create(&tid2, NULL, threaded_doqsort, &a2);

            pthread_join(tid1, NULL);
            pthread_join(tid2, NULL);
        }
    }
    return NULL;
}
```

Concurrent quicksort using threads

```
void runSorts(long long int n){

    int *A = shareMem(sizeof(int)*(n+1));
    int arr[n+1];
    for(int i=0;i<n;i++) A[i] = n - i;
    for(int i=0;i<n;i++) B[i] = A[i];
    for(int i=0;i<n;++i) arr[i] = A[i];
    qsort(arr, n, sizeof(int), compare);

    struct timespec ts;

    printf("Running concurrent_quicksort\n");
    clock_gettime(CLOCK_MONOTONIC, &ts);
    long double st = ts.tv_nsec/(1e9)+ts.tv_sec;
    long double t1,t2,t3;
    doqsort(A, 0, n-1);
    pthread_t tid;

    clock_gettime(CLOCK_MONOTONIC, &ts);
    long double en = ts.tv_nsec/(1e9)+ts.tv_sec;
    printf("time = %Lf\n", en - st);
    t1 = en-st;
    struct arg a;
    a.l = 0;
    a.r = n-1;
    a.A = B;
    printf("Running threaded_concurrent_quicksort\n");
    clock_gettime(CLOCK_MONOTONIC, &ts);
    st = ts.tv_nsec/(1e9)+ts.tv_sec;

    pthread_create(&tid, NULL, threaded_doqsort, &a);
    pthread_join(tid, NULL);

    clock_gettime(CLOCK_MONOTONIC, &ts);
    en = ts.tv_nsec/(1e9)+ts.tv_sec;
    printf("time = %Lf\n", en - st);
    t2 = en-st;

    printf("Running normal_quicksort\n");
    clock_gettime(CLOCK_MONOTONIC, &ts);
    st = ts.tv_nsec/(1e9)+ts.tv_sec;

    quickSort(B, 0, n-1);

    clock_gettime(CLOCK_MONOTONIC, &ts);
    en = ts.tv_nsec/(1e9)+ts.tv_sec;
    printf("time = %Lf\n", en - st);
    t3 = en - st;

    printf("normal_quicksort ran:\n\t[ %Lf ] times faster than concurrent_quicksort\n\t[ %Lf ] times
faster than threaded_concurrent_quicksort\n\n\n", t1/t3, t2/t3);

    int flag[4];
    flag[0]=flag[1]=flag[2]=flag[3]=0;
    for(int i=0;i<n;++i)    if(A[i] != arr[i]) flag[0] = 1;
    for(int i=0;i<n;++i)    if(B[i] != arr[i]) flag[1] = 1;
    for(int i=0;i<n;++i)    if(a.A[i] != arr[i]) flag[2] = 1;

    if(!flag[0] && !flag[1] && !flag[2])    printf("Sorting done right\n");
    else                                    printf("Sorting\n");
    shmdt(A);
```

```
}

int main(){

    long long int n;
    scanf("%lld", &n);
    runSorts(n);
    return 0;
}
```