

```
#include<stdio.h>
#include<sys/types.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<ctype.h>
```

```
int partition (int a[], int start, int end)
{
    int pivot = a[end]; // pivot element
    int i = (start - 1);

    for (int j = start; j <= end - 1; j++)
    {
        // If current element is smaller than the pivot
        if (a[j] < pivot)
        {
            i++; // increment index of smaller element
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}
```

```
/* function to implement quick sort */
```

```
void quick(int a[], int start, int end) /* a[] = array to be sorted, start = Starting index, end = Ending index */
```

```
{  
    if (start < end)  
    {  
        int p = partition(a, start, end); //p is the partitioning index  
        quick(a, start, p - 1);  
        quick(a, p + 1, end);  
    }  
}
```

```
void merge(int arr[], int l, int m, int r)
```

```
{  
    int i, j, k;  
    int n1 = m - l + 1;  
    int n2 = r - m;
```

```
/* create temp arrays */
```

```
int L[n1], R[n2];
```

```
/* Copy data to temp arrays L[] and R[] */
```

```
for (i = 0; i < n1; i++)  
    L[i] = arr[l + i];  
for (j = 0; j < n2; j++)  
    R[j] = arr[m + 1 + j];
```

```
/* Merge the temp arrays back into arr[l..r]*/
```

```
i = 0; // Initial index of first subarray  
j = 0; // Initial index of second subarray
```

```

k = l; // Initial index of merged subarray
while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    }
    else {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Copy the remaining elements of L[], if there
are any */
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of R[], if there
are any */
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
}

/* l is for left index and r is right index of the

```

```

sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

```

```

int main()
{
    pid_t p;
    int n;
    printf("Enter the number of elements");
    scanf("%d",&n);
    int a[n];

    for(int i=0;i<n;i++)
    {
        printf("Enter %d th element ",(i+1));
        scanf("%d",&a[i]);
    }

    p=fork();

```

```

quick(a,0,n-1);

if(p==0)
{

    printf("Process is child, ID is %d \n",getpid());
    printf(" Parent's process, ID is %d \n",getppid());
    quick(a,0,n-1);
    printf("After sorting elemets are ");
    for(int i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }

}

else{
    printf("Process is in Parent ,ID is %d \n",getpid());
    mergeSort(a,0,n-1);
    printf("After merge Sort elements are \n ");

    for(int i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}

return 0;
}

```