Parent

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

// ---------- QUICK SORT FUNCTION ----------
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

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

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
```

```
        quickSort(arr, pi + 1, high);
    }
}

// ---------- MAIN FUNCTION ----------
int main() {
    int n;
    printf("Enter number of integers: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d integers: ", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid == 0) {
        // CHILD PROCESS (to demonstrate orphan)
        printf("\n[Child] PID: %d | PPID: %d\n", getpid(), getppid());
        printf("[Child] Sleeping for 5 seconds to demonstrate orphan state...\n");
        sleep(5);
        printf("[Child] After sleep — my new PPID is %d (1 = Orphan)\n", getppid());
        printf("[Child] Now executing merge_sort program using execve...\n");

        // Convert array elements to strings for execve
```

```
    char *args[n + 2];
    args[0] = "./child"; // program name
    for (int i = 0; i < n; i++) {
        args[i + 1] = malloc(10);
        sprintf(args[i + 1], "%d", arr[i]);
    }
    args[n + 1] = NULL;

    // Execute new program
    execve("./child", args, NULL);
    perror("execve failed");
    exit(1);
} else {
    // PARENT PROCESS
    printf("\n[Parent] PID: %d created child PID: %d\n", getpid(), pid);
    printf("[Parent] Sorting array using Quick Sort...\n");
    quickSort(arr, 0, n - 1);

    printf("[Parent] Sorted array (Quick Sort): ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    printf("[Parent] Sleeping for 10 seconds to demonstrate zombie state...\n");
    sleep(10);
    printf("[Parent] Woke up, now calling wait() to remove zombie...\n");

    wait(NULL);
    printf("[Parent] Child process terminated. No zombie remains.\n");
}
return 0;
```

```
}
```

Child

```
#include <stdio.h>
#include <stdlib.h>

// ---------- MERGE SORT FUNCTION ----------
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

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

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
```

```
    }
}

// ---------- MAIN FUNCTION ----------
int main(int argc, char *argv[]) {
    int n = argc - 1;
    int arr[n];

    for (int i = 1; i < argc; i++)
        arr[i - 1] = atoi(argv[i]);

    printf("\n[Child - Merge Sort] Received array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    mergeSort(arr, 0, n - 1);

    printf("[Child - Merge Sort] Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

Output

```
gcc parent.c -o parent
gcc child.c -o child
./parent
```

Enter number of integers: 5

Enter 5 integers: 8 4 2 9 5

[Parent] PID: 4311 created child PID: 4312

[Parent] Sorting array using Quick Sort...

[Parent] Sorted array (Quick Sort): 2 4 5 8 9

[Parent] Sleeping for 10 seconds to demonstrate zombie state...

[Child] PID: 4312 | PPID: 4311

[Child] Sleeping for 5 seconds to demonstrate orphan state...

[Child] After sleep — my new PPID is 1 (1 = Orphan)

[Child] Now executing merge_sort program using execve...

[Child - Merge Sort] Received array: 8 4 2 9 5

[Child - Merge Sort] Sorted array: 2 4 5 8 9

[Parent] Woke up, now calling wait() to remove zombie...

[Parent] Child process terminated. No zombie remains.