Parent

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>


void sortArray(int arr[], int n) {

    int temp;

    for (int i = 0; i < n - 1; i++) {

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) {

                temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}


int main() {

    int n;

    printf("Enter the number of elements: ");

    scanf("%d", &n);


    int arr[n];

    printf("Enter %d elements:\n", n);

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }
```

```
pid_t pid = fork();


if (pid < 0) {
    perror("Fork failed");
    exit(1);
}
else if (pid == 0) {
    // Child process (becomes orphan if parent exits early)
    printf("\n[Child] Child process created with PID = %d\n", getpid());
    printf("[Child] Parent PID = %d\n", getppid());
    sleep(5); // sleep to demonstrate orphan state
    printf("[Child] After sleep, Parent PID = %d (if 1, then orphan)\n", getppid());
    printf("[Child] Child process finished execution.\n");
}
else {
    // Parent process
    printf("\n[Parent] Parent process PID = %d\n", getpid());
    printf("[Parent] Sorting the array...\n");
    sortArray(arr, n);


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


    // Prepare arguments for execve
    char *args[n + 2];
    args[0] = "./child"; // program name
    char numStr[10];
    for (int i = 0; i < n; i++) {
        args[i + 1] = malloc(10);
```

```
            sprintf(args[i + 1], "%d", arr[i]);
        }
        args[n + 1] = NULL;

        // Wait to demonstrate zombie state
        pid_t child_pid = fork();
        if (child_pid == 0) {
            // Execute child program with sorted array
            execve("./child", args, NULL);
            perror("execve failed");
            exit(1);
        } else {
            printf("[Parent] Waiting for child process to finish...\n");
            wait(NULL);
            printf("[Parent] Child process finished.\n");
        }
    }
    return 0;
}
```

Child

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

int main(int argc, char *argv[]) {
    printf("\n[Child Program] Received sorted array from parent:\n");

    int n = argc - 1;
    int arr[n];
    for (int i = 1; i < argc; i++) {
        arr[i - 1] = atoi(argv[i]);
        printf("%d ", arr[i - 1]);
    }

    printf("\n[Child Program] Reversed array:\n");
    for (int i = n - 1; i >= 0; i--) {
        printf("%d ", arr[i]);
    }

    printf("\n");
    return 0;
}
```

Output

```
gcc parent.c -o parent
gcc child.c -o child
./parent
Enter the number of elements: 5
Enter 5 elements:
9 3 7 1 4

[Parent] Parent process PID = 4321
[Parent] Sorting the array...
[Parent] Sorted array: 1 3 4 7 9
[Parent] Waiting for child process to finish...

[Child Program] Received sorted array from parent:
1 3 4 7 9
[Child Program] Reversed array:
9 7 4 3 1
```

```
[Parent] Child process finished.


[Child] Child process created with PID = 4322

[Child] Parent PID = 4321

[Child] After sleep, Parent PID = 1 (if 1, then orphan)

[Child] Child process finished execution.
```