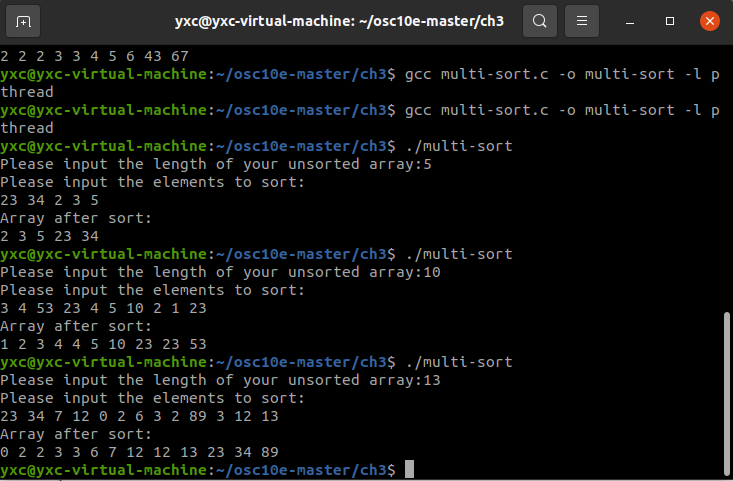
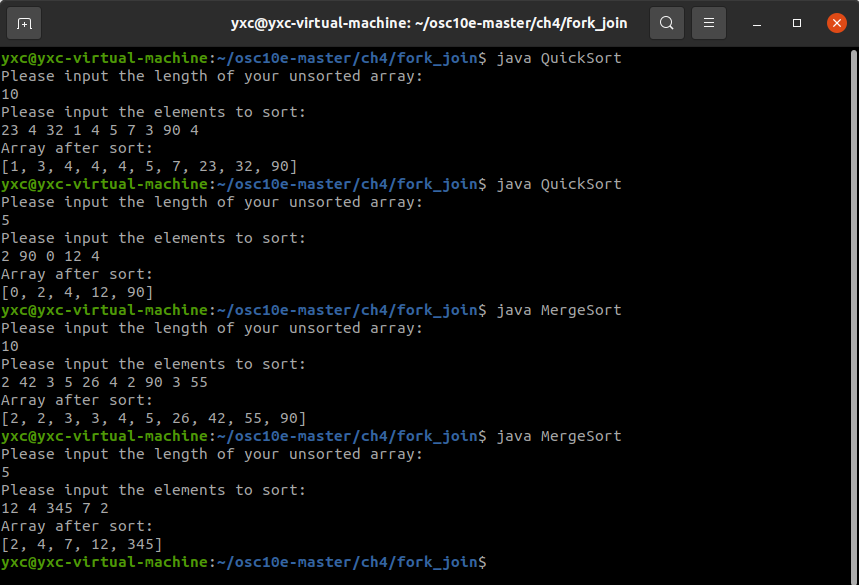
1. Multithreaded Sorting Application
2. 首先输入数组长短n和具体元素数值到array数组中。
3. 然后创建两个pthread\_t变量，即线程ID，用来表示两个排序线程。然后用pthread\_create()函数创建线程，并用pthread\_join()运行。需要注意的是，由于pthread\_create()只能向线程传递一个函数，且这个函数只能有一个参数，而排序部分需要首尾位置两个参数，故需要定义一个结构体range去包含排序部分的起始位置和结束位置，然后传递给pthread\_create()去调用排序函数。两个线程分别排序array数组的前半部分（0~n/2-1）与后半部分（n/2~n-1）。
4. 最后创建一个用于合并的线程，执行归并排序函数merge()，参数是起始位置为0，结束位置为n-1的range类型，将排序好的数组放在result数组中。
5. 根据各个线程pthread\_create()的返回值判断其报错，并根据错误输出相应的提示。
6. 输出result数组。
7. 值得注意的是，由于pthread.h文件比较特殊，在编译时，需在编译命令后加上“-l pthread”命令才能正确编译。

结果如图

1. Fork-Join Sorting Application
2. 根据不同的排序方式，创建QuickSort和MergeSort两个继承于RecursiveAction的class，每个类的元素包括需排序数组和起始位置。在类中定义不同的排序函数，对数组进行排序。
3. 为了体现Comparable接口，数组元素之间的比较用compareTo进行。
4. 在主函数main中，先将需排序数组的长度和初始元素序列从System.in中取出（用Scanner变量承接）并放入n和array数组中。
5. 再根据n和array创建QuickSort或MergeSort类元素，并将其invoke到ForkJoinPool元素pool中，执行线程，对数组进行排序。
6. 输出结果为排好序之后的array数组

结果如图所示：