Gray Houston

ghousto

Project 2

Analysis Questions

1. Both the best case and the worst case running time is O(nd). To start, the inner loop will execute in O(d) time, as the program loops over the next d elements to find the smallest element to insert. Next, the outer loop will iterate n times and therefore run in O(n) time. Because the inner loop will run proportionally to the outer loop, the overall running time is O(nd). Because the inner loop will always run in O(d) time (and can never run for shorter), both the best case and the worst case will be O(nd) time.
2. The best and worst case running time is O(nd logd). To start, the creation of each small min-heap will take log(d) because it uses the bottom-up heapify method and there will be d heaps created. Therefore, creating all of the heaps will run in O(d logd). Deleting each element from a heap will run in O(n) time. Therefore, the algorithm will run at O(nd logd) time. This implementation runs in this time because it is in-place.
3. If T(n) is the time that it takes for my implementation of Mergesort to run, then

T(n) = 2 \* T(n/2) + K where K is some constant locality value

= 22 \* T(n/22) + K(1 + 2)

= 23 \* T(n/23) + K(1 + 2 + 22)

…= 2K \* T(n/2K) + K(1 + 2 + 22 + … + 2K-1)

= 2K \* T(n/2K) + K \* n

= K \* n

Therefore, when d is set to a constant value, my Mergesort will have an asymptotic O(n) running time.

1. A
2. A
3. For an array of size n, when d=0, that means that each element is at most 0 indexes away from its final position. When there are 0 indices between an element and its final position, it means that the element must be in its final position. Therefore, d=0 for an array means that the array is already sorted.
4. Of all of the locality sorting algorithms implemented, the only stable sort was LMerge. First, selection sort’s and heapsort’s original algorithms were not stable, so their locality-based implementations would not have been. Mergesort remained stable because the only difference between the locality version and the normal version were the bounds used for the merge. Because changing the bounds of the merge doesn’t impact stability, the locality Mergesort is stable.
5. We were not asked to implement a locality version of Insertion Sort because it is already locality aware. As the element being sorted moves left into the sorted region, it moves closer and closer to its final destination. If there is some d such that d >= the maximum distance between an element and its final position, because Insertion Sort already moves elements to the left towards their final positions, Insertion Sort will never iterate a single element more than d times. Therefore, Insertion Sort already handles the case of locality.
6. Even if data with locality d significantly less than n was used with bubble sort, it would still run in O(n2) time. Because the array is already partially sorted, there would likely just be less exchanges. However, the number of comparisons will remain the same.
7. The method that could be used to generate data that isn't sorted is quicksort because of its partition. Each partition is a logical separation of the data. Furthermore, the final position of each element within the partition is within the partition. Therefore, the locality of a set of a data is the size of the partition. Using this information, you could take any array, even if it isn't sorted, and form a non-sorted array that has the locality condition.  
     
   To create the array with a specified locality, call quicksort as normal until a partition size is less than or equal to the locality parameter. Quicksort wouldn't be called on this partition (of size <= d). This would act as a base case in the recursion. Because the locality of an element in the quicksort algorithm is the size of its partition, stopping when partitions are less than d would create data that possesses the locality condition.