Set 2 Solution

1.

2 6 9 5 1 3 4 7 8
2 6 9 5 1 * 3 4 7 8
1 6 9 5 2 * 3 4 7 8
1 2 9 5 6 3 * 4 7 8
1 2 3 5 6 9 4 * 7 8
1 2 3 4 6 9 5 * 7 8
1 2 3 4 5 9 6 * 7 8
1 2 3 4 5 6 0 9 7 8
1 2 3 4 5 6 7 9
1 2 3 4 5 6 7 8 9

There are other possible valid orders. The important thing is that the Quicksort is maximized to 8 iterations. Each iteration should pick a pivot that is either the min or the max of the remaining elements, thus causing all elements to partition to one side of the pivot. Thus each partitioning reduces the partition size by only 1, resulting in a total of n-1 iterations. It is easiest to generate a solution by starting from the end and working backwards.

2.

The main idea is to note that the recursion steps when \( \frac{n}{2^i} = k \), that is \( i = \log_2 \frac{n}{k} \). The recursion takes in total \( O(n \cdot \lg \frac{n}{k}) \). The resulting array is composed of \( k \) subarrays of size \( \frac{n}{k} \), where the elements in each subarray are all less than all the subarrays following it. Running INSERTION-SORT on the entire array is thus equivalent to sorting each of the \( \frac{n}{k} \) subarrays of size \( k \), which takes on the average \( \frac{n}{k} \cdot O(k^2) = O(nk) \) (the expected running time of INSERTION-SORT is \( O(n^2) \)).

If \( k \) is chosen too big, then the \( O(nk) \) cost of insertion becomes bigger than \( \Theta(n \lg n) \). Therefore \( k \) must be \( O(\lg n) \). Furthermore it must be that \( O(nk + n \lg \frac{n}{k}) = O(n \lg n) \). If the constant factors in the big-oh notation are ignored, then it follows that \( k \) should be such that \( k < \lg k \) which is impossible (unless \( k = 1 \)) - the error comes from ignoring the constant factors. Let \( c_1 \) be the constant factor in quicksort, and \( c_2 \) be the constant factor in insertion sort. Then \( k \) must be chosen such that \( c_2 k + c_1 \lg \frac{n}{k} < c_1 \lg n \) which requires \( c_1 k < c_2 \lg k \). In practice these constants cannot be ignored (also there can be lower order terms in \( O(n \lg n) \)) and \( k \) should be chosen experimentally.

3.

First apply the k-max finding (median finding) algorithm to find the \( k^{th} \) largest element of that unsorted array. This will take \( O(n) \) time. Then, apply sorting algorithm to sort the first \( k \) elements. This will take \( O(k \log k) \) time. So, the total order is \( O(n + k \log k) \).
4. According to the definition, a number is major if it is repeated at least \( n/2 \) times. So, if there is any major number in the array, we can find the number using median finding algorithm and check if it occurs at least \( n/2 \) times. So, the order is \( O(n) \) which is linear.

5. If we group elements into groups of 3 each, the recurrence equation will be

\[
T(n) = cn + T(n/3) + T(2n/3).
\]

Let's assume \( T(n) = an \) where \( a \) is a constant. So,

\[
an = cn + an/3 + 2an/3
\]

? \( an = cn + an \)

? \( cn = 0 \)

? \( c = 0. \)

So, the median finding algorithm does not work in linear time. Now, let's assume \( T(n) = an \log n \).

\[
an \log n = cn + a(n/3) \log(n/3) + a(2n/3) \log(2n/3)
\]

? \( an \log n = cn + an \log n - a(n/3) \log 3 - a(2n/3) \log(3/2) \)

? \( cn = a(n/3) \log(3/2) + a(n/3) \log 2 + a(2n/3) \log(3/2) \)

? \( c = a(1/3 + \log(3/2)) \)

? \( c = a*\text{constant.} \)

So, for group size 3, the order becomes \( n \log n \), not linear anymore.

6. For group size 7, we get \( T(n) = cn + T(n/7) + T(5n/7) \). Let's assume, \( T(n) = an \). So,

\[
an = cn + an/7 + 5an/7
\]

? \( a = c + 6a/7 \)

? \( c = a/7 \)

? \( c = a*\text{constant.} \)

So, the order is linear in time. We can say that asymptotically, both of group size 5 and 7 is going to take same time. However, the constant factor may be different.

7. We find median of \( X \) say \( mx \) and median of \( Y \) say \( my \). If \( mx == my \), we are done, \( mx \) or \( my \) is the median of \( X \cup Y \). If that's not the case, for example, when \( mx < my \), we again start with the right half of \( X \) and left half of \( Y \). This reduces the problem space from \( n \) to \( n/2 \). In case of \( mx > my \), we would start with right half of \( X \) and left half of \( Y \). We stop, when \( mx == my \) or there are only one element left.
This takes $O(\log n)$ time asymptotically.