n the 1st question, we use graham scan to find our convex hull points.

n is the number of input points.

1. The first step (finding the bottom-most point) takes O(n) time. (since we are comparing each element with the 0th point to see which point has the minimum x coordinate and among that, which has the maximum y coordinate. This is the left-bottom most point. (point P)
2. The second step (sorting points) takes O(nlogn) time. Here, we are using quicksort algorithm. Take an arbitrary pivot. If the point in the array is to the left of the line segment formed by P and the pivot, then it comes to the right side in the partitioned array. Or else, it comes to the left.
3. The third step takes O(n) time. Once an element is popped, we realise that it never comes on the convex hull , its always inside it, so we never push it onto the array again. Therefore, step, every element is pushed and popped at most one time from the stack. Since each stack operation takes O(1) time, to be able to process all points would take O(n) time.

Overall complexity is O(n) + O(nlogn) + O(2n) which is O(nlogn).

#################################################################################

In the 2nd question, radix sort is used which is a non-comparative sorting algorithm. In a comparative sorting algorithm, the time complexity observed is O(nlogn).

But here, this non-comparative sorting algorithm can go even better which is upto O(n). (since its not lower bounded by decision- tree)

Average case complexity occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of Radix sort is theta(nd).

1.Let the number of digits in the maximum number be d. then we will be iterating for every position in all the numbers till we reach the dth position.

2. Since we are iterating through every number for a given position using counting sort (whose time complexity is O(n) where n are number of input samples), here the average time complexity will be theta (nd).

For negative numbers, I have used a modified version of the radix sort algorithm, which sorts from LSB to MSB in descending order. This again makes use of counting sort for each position which takes O(n) time.

In the 3rd question, the deterministic quick-select algorithm is used wherein the pivot is chosen deterministically by the median of medians algorithm.

The median of medians algorithm divides the input into groups of 5 and finds medians of those groups recursively. The worst-case time complexity of this part is O(n) because you have to recursively find medians of groups until you have a good pivot. n is number of elements.

1. The median of a group of 5 (or lesser elements incase of the last group) can be found in 7 comparisons. Since there are n/5 groups formed, it takes 7n/5 comparisons.
2. Now, we are having n/5 medians. We will find the median of these n/5 medians using the same Select algorithm. Therefore, the time taken is T(n/5).
3. Using the median of median which can act as a very good pivot ( as proved later), we compare every other element of the array and divide it into smaller and larger subgroups. This takes O(n) time as each element is compared only once.
4. Since, the pivot was median of medians, its bigger than half of the elements in the median array(which was consisting of n/5 medians). Therefore, it was bigger than n/10 medians and each median was further bigger than 2 elements in its respective subgroup of 5 elements. Therefore, overall its bigger than 3n/10 elements for sure. Its smaller than 3n/10 elements for sure.
5. So, if the element that we are looking for is bigger than this median of medians, we will be eliminating atleast 3n/10 elements for sure.
6. Like this, with each comparison, we will be reducing the size of the array by 30%. In order words, when we get into SELECT function again to find the kth element, we will be left with atmost 70% elements which makes our time taken= T(7n/10).

T(n) ≤ T(n/5) + T(7n/10) + cn + 7n/5

≤ T(n/5) + T(7n/10) + O(n)

Here, if we take T(n)= cn, we are able to get constant c and value n for which the above linear time complexity holds true.

SOURCES:

1. Class notes
2. CLRS- Introduction to Algorithm