1a. what will happen to every element in the array is that it will switch with itself because 1 is not less than 1. The pivot will move all the way to the end of the array.

1b. every element will have to switch because the pivot will be 1. This is because the array is descending so the most right element will be the smallest. So nothing will happen to the array because an element will only be switched when the element is less than the pivot element which is 1. At the end the pivot element will be moved to the same spot.

2a.its the worst case because no matter what pivot is picked, quicksort will have to go through all the values in the array. Since all values are the same, each recursive call will lead to unbalanced partitioning.

2b. this is the worst case because the if the pivot is chosen as the first element of the array, then it will partition the array into two sub arrays one of size 1 and another of size n-1. This makes the total time n2.

3a. it would be the worst case because an increasing subarray, the median of the first, last and middle values will be the median of the entire subarray. Using the median value as a pivot will split the subarray in the middle. This will cause the total number of key comparisons be the smallest.

3b. it would be the worst case because decreasing subarray, the median of the first, last, and middle values will be the median of the entire subarray. Using it as a pivot will split the subarray in the middle, this will cause the total number of key comparisons to be the smallest, median of three method will slow down the process sorting and it is founded as very to calculate the middle value.

C4. Minimum number of elements in a heap of height h is 2hand maximum is 2h+1.

C5. Write n=2m−1+kn = 2^m − 1 + kn=2m−1+k where m is as large as possible. Then the heap consists of a complete binary tree of height m−1m − 1m−1, along with k additional leaves along the bottom. The height of the root is the length of the longest simple path to one of these k leaves, which must have length m. It is clear from the way we defined m that m=⌊lg n⌋.

C6. if the largest element in the subtree were somewhere other than the root, it has a parent that is in the subtree. So, it is larger than its parent, so, the heap property is violated at the parent of the maximum element in the subtree.

E10. There’s no effect. The comparisons are carried out, A[i] is found to be largest and the procedure just returns.

E11. There’s no effect. In that case, it is a leaf. Both left and right return values that fail the comparison with the heap size and i is stored in largest. Afterwards the procedure just returns.

E12.

⟨5,13,2,25,7,17,20,8,4⟩

⟨5,13,20,25,7,17,2,8,4⟩

⟨5,25,20,13,7,17,2,8,4⟩

⟨25,5,20,13,7,17,2,8,4⟩

⟨25,13,20,5,7,17,2,8,4⟩

⟨25,13,20,8,7,17,2,5,4⟩

⟨4,13,20,8,7,17,2,5,25⟩

⟨20,13,4,8,7,17,2,5,25⟩

⟨20,13,17,8,7,4,2,5,25⟩

⟨5,13,17,8,7,4,2,20,25⟩

⟨17,13,5,8,7,4,2,20,25⟩

⟨2,13,5,8,7,4,17,20,25⟩

⟨13,2,5,8,7,4,17,20,25⟩

⟨13,8,5,2,7,4,17,20,25⟩

⟨4,8,5,2,7,13,17,20,25⟩

⟨8,4,5,2,7,13,17,20,25⟩

⟨8,7,5,2,4,13,17,20,25⟩

⟨4,7,5,2,8,13,17,20,25⟩

⟨7,4,5,2,8,13,17,20,25⟩

⟨2,4,5,7,8,13,17,20,25⟩

⟨5,4,2,7,8,13,17,20,25⟩

⟨2,4,5,7,8,13,17,20,25⟩

⟨4,2,5,7,8,13,17,20,25⟩

⟨2,4,5,7,8,13,17,20,25⟩​

E13. Consider array 21 20a 20b 12 11 8 7 (already in max-heap format)

here 20a = 20b just to differentiate the order we represent them as 20a and 20b

While heapsort first 21 is removed and placed in the last index then 20a is removed and placed in last but one index and 20b in the last but two indexes so after heap sort the array looks like

7 8 11 12 20b 20a 21.

It does not preserve the order of elements and hence can't be stable