The order of growth running time for priority queue with N items of a d-ary heap would be log base d of (N) for insert, d*log base d of (N) for delMax and 1 for max. Let us first understand heap sort time complexity.

Heap Sort has O(nlogn) time complexities for all the cases; best, average and worst. The height of a complete binary tree containing n elements is log(n) To fully heapify an element whose subtrees are already max-heaps, comparing the element with its left and right children needs to occur. Then it continues pushing that element downwards until it reaches a point where both its children are smaller than aforementioned element. So In worst case, we will need to move an element from the root to the leaf making multiple log(n) comparisons and swaps in order to accomplish our task. The worst case complexity of the build heap step is n/2*log(n) which is thus **nlogn**. For our case it would be **2*nlogn** with the leading coefficient being 2 attained from the given table.

```
void heapSort(int arr[], int n)
{
   for (int i = n / 2 - 1; i >= 0; i--)
      heapify(arr, n, i);

   for (int i=n-1; i>=0; i--)
   {
      swap(arr[0], arr[i]); <--Swaps mentioned in passage

      heapify(arr, i, 0);
 }
```

As for the delMax function we have a worst case time complexity of **d*log base d of (N)**. We can see so below since the complexity was obtained from the table given.

```
int delMax(int hole)
{
  int keyItem = array[hole];
  array[hole] = array[currentSize - 1];
  currentSize--;
  sink( hole );
  return keyItem;
}
```