1f.

The big O notation for my delMax function is O(N)+O(N) +O(1) + O(1)= 2N in the worst case because it has a swap operation which includes a for loop in order to complete the necessary swap; the other linear complexity comes from the for loop this is used to output the array after the deletion and swap is completed. The two “O(1)”’s come from two iterations of a variable that exist in the function. For loops are always O(N) in time complexity, and since there are two of them, 2N would be the time complexity for the function. The pieces of the code who operate under constant time (the two iterations mentioned earlier) aren’t significant when we’re describing the worst-case scenarios for time complexity.

delMax Function:

```
void delMax(int d,int arr[],int sizeOfArray){

sizeOfArray++; ← this is the first constant operation mentioned in my answer O(1)

cout << "\n\nThe Largest Value in The Heap(which is going to be deleted) =\n" << arr[0] << endl;

swap(arr[0],arr[sizeOfArray-1]); ← swap is a for loop function who swaps two variables. This being a for loop is a giveaway that it a O(N)

sizeOfArray--; ← second constant operation in this function O(1)


cout << "\n\nArray After Deletion: ";

for (int i = 0; i < sizeOfArray;i++) ← second loop in the function O(N)

{

   cout << arr[i] << " ";

}

}
```

For my daryHeapSort function there are several things to consider in regards to its time complexity (although not everything is important for the overall time complexity). The time complexity for this function is: O(1) + O(1) +O(N) + O(N) + O(1) + O(1) + O(N) + O(1) + O(N) = 4N. The code snippet will highlight where these constant and linear time complexities stem from

daryHeapSort Function:

```
void daryHeapSort(int d,int arr[],int sizeOfArray){
int copyArray[sizeOfArray] = {0}; ← O(1)
signed int t = sizeOfArray - 1; ← O(1)
while (t != -1) ← O(N) [while loops are also linear functions in programming as for loops are]
{
swap(arr[0],arr[t]); ← O(N)
copyArray[t] = arr[t]; ←O(1)
int Max = 0; ← O(1)
   for (int w = 0; w < t; w++) ← O(N)
   {
      if (arr[Max] < arr[w])
      {
         Max = w;
      }
   }
   swap (arr[Max], arr[0]);
t--; ← O(1)
}
cout << "\n\nSorted Array = ";
for (int j = 0; j < sizeOfArray;j++)
{
   cout << copyArray[j] << " ";
}
cout << endl; }
```