1.
(a) We only check d positions right of the current index to search for the next minimum. Make a loop that loops through the entire array linearly. Have a nested loop inside this that loops from the current index i till the index i+d. This loop finds the minimum number within these indexes and then once the nested loop is done, it swaps the minimum number with the current index i. Now the next time the nested loop runs it will look at number from index i+1 till i+d+1 and so on. At the end of the outer loop, the array will be sorted. The nested loop only checks i+d position if it is within bounds and if it is not then it checks up till the last index.

(b) The outer loop runs O(N) times. The nested loop runs O(d) times. Thus, the time complexity will be O(N) * O(d) which is O(dN).

(c) `[3, 2, 0, 4, 1, 8, 5, 7, 9, 6]`

First loop for the nested loop goes from index 0 till index 3 and finds 0:
`[0, 2, 3, 4, 1, 8, 5, 7, 9, 6]`

Next loop for the nested loop goes from index 1 till index 4 and finds 1:
`[0, 1, 3, 4, 2, 8, 5, 7, 9, 6]`

Next loop for the nested loop goes from index 2 till index 5 and finds 2:
`[0, 1, 2, 4, 3, 8, 5, 7, 9, 6]`

Next loop for the nested loop goes from index 3 till index 6 and finds 3:
`[0, 1, 2, 3, 4, 8, 5, 7, 9, 6]`

Next loop for the nested loop goes from index 4 till index 7 and finds 4:
`[0, 1, 2, 3, 4, 5, 8, 7, 9, 6]`

Next loop for the nested loop goes from index 5 till index 8 and finds 5:
`[0, 1, 2, 3, 4, 5, 8, 7, 9, 6]`

Next loop for the nested loop goes from index 6 till index 9 and finds 6:
`[0, 1, 2, 3, 4, 5, 6, 7, 9, 8]`

Next loop for the nested loop goes from index 7 till index 9 and finds 7:
`[0, 1, 2, 3, 4, 5, 6, 7, 9, 8]`

Next loop for the nested loop goes from index 8 till index 9 and finds 8:
`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

Next loop for the nested loop goes from index 9 till index 9 and finds 9:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
(d) A: Array of size N
for i from 1 to N-1:
    j=i
    while j>i-d and j>0 and A[j] <A[j-1]:
        swap A[j] and A[j-1]
    end while
end for
```

In this algorithm, it does insertion sort, but it does not swap more than d times in any case. The outer loop loops through the entire array linearly. The inner loop checks if the current index is less than the index before that then swap it till it's not and do not swap more than d times.

(e) The outer loop runs O(N) times. The nested loop runs O(d) times in the worst case. Thus, the time complexity will be O(N) * O(d) which is O(dN).

(f) `[3, 2, 0, 4, 1, 8, 5, 7, 9, 6]`

`[2, 3, 0, 4, 1, 8, 5, 7, 9, 6]`

`[0, 2, 3, 4, 1, 8, 5, 7, 9, 6]`

`[0, 1, 2, 3, 4, 8, 5, 7, 9, 6]`

`[0, 1, 2, 3, 4, 5, 8, 7, 9, 6]`

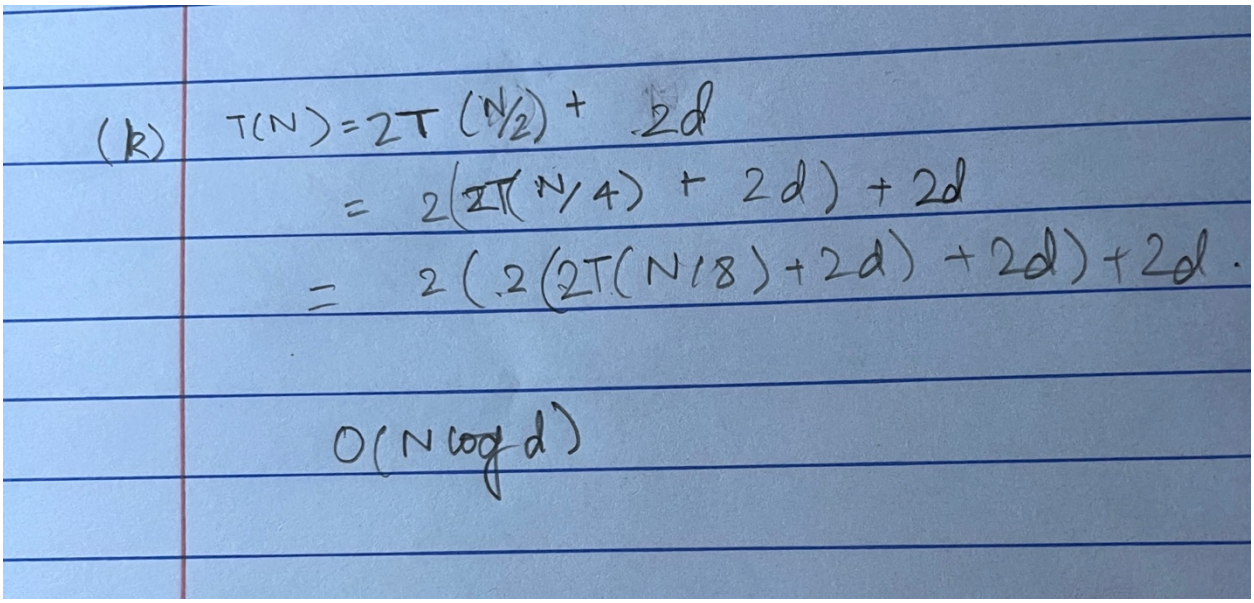
`[0, 1, 2, 3, 4, 5, 7, 8, 9, 6]`

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

(j)Like regular merge sort, divide the array into 2 halves and call the function again for the two halves. Merge the two halves by sorting them. This is a recursive algorithm which first divides the array into smaller arrays until we have 1-element

arrays and merge these arrays. While merging we only look at the last d elements of the first list and the first d elements of the second list to sort the arrays and merge them. Thus, only looking at 2d elements when merging and sorting.

(k)



(l)