1. The choice of algorithm is mergesort because it runs O(nlogn) time and it is stable. The algorithm is stable because it runs the sort without changing the order of the elements of the array.

for i in range(0,n1):

while l>=0 and l<r:

arr[l] = arr[l+i]

for j in range(0,n2):

while r>=0 and r<m:

arr[r] = arr[m+1+j]

As you can see the order is not being changed.

2. The best case occurs if the array is already sorted. The best case time complexity of this algorithm is O(nlogn) because the sort is determined by a loop of which the run time is log n. The loop will be executed n times, so the best-case time complexity is O(nlogn).

for i in range(0,n1):

while l>=0 and l<r:

arr[l] = arr[l+i]

for j in range(0,n2):

while r>=0 and r<m:

arr[r] = arr[m+1+j]

The loop is executed n times.

3. The worst case time complexity would also be O(nlogn) because regardless of the array the sort is determined by a loop of which the run time is log n. The loop will be executed n times, so the worst-case time complexity is also O(nlogn).

for i in range(0,n1):

while l>=0 and l<r:

arr[l] = arr[l+i]

for j in range(0,n2):

while r>=0 and r<m:

arr[r] = arr[m+1+j]

The loop is executed n times as shown.