1.

(a)

1. Sort the array using merge sort
2. Set two pointers: left and right, left = 0, right = length of array – 1
3. While left < right:
   1. If(array[left] + array[right] == sum) then return left and right
   2. Else if(array[left] + array[right] > sum) then right = right – 1
   3. Else left = left + 1 (In this case array[left] + array[right] < sum)

Merge Sort(described in the lecture slides):

Merge\_Sort(A, p, r):  
 if(p < r):

Then q = floor( (p + r) / 2)

Merge\_Sort(A, p, q);

Merge\_Sort(A, q + 1, r)

Merge(A, p, q, r)

Analysis: The time complexity of merge sort is O(n log n), after sort, the while loop takes O(n),

So the total time complexity is O(n log n).

(b)

In order to accomplish the same task in O(n), we need auxiliary space.

We use a hash table to store the index information of the elements in array.

1. initialize a hash table, here using C++ map: Map<int, int> hash\_table

the key in hash table is the value of the element, the value in hash table is the index of the element

1. for i = 0 to length of array – 1 do:

calculate the the other addend, a1 = sum – array[i]

if hash\_table contains a1 then return a1 and array[i]

else update hash\_table: hash\_table[array[i]] = i

Analysis:

Time complexity: O(n), because we traverse the array only once. The look up operation of hash table takes O(1) time.

Space complexity: O (n), We need extra space for the hash table.