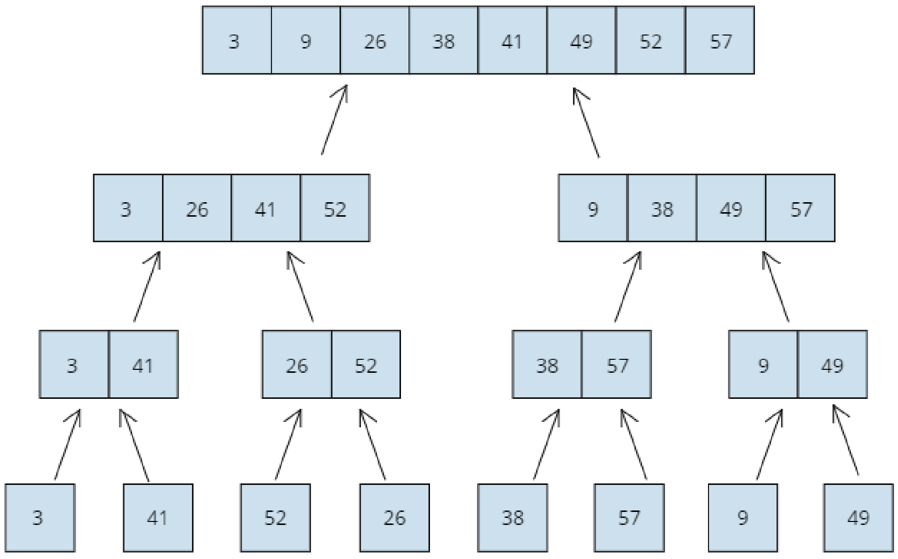
1 Using Figure 2.4 as a model, illustrate the operation of merge sort on the array A=(3,41,52,26,38,57,9,49).



2 Rewrite the Merge grocedure so that it does not use sentinels? instead stopping pnce either array L or Rhas had all elements copied back to A and then colying the remainder of the other array back into A

```
def _merge(A,p,r,q):
    L = A[p:r+1];
    R = A[r+1:q+1];
    index = p;
    while len(R)>0 and len(L)>0:
        if (L[0]<R[0]):
            A[index] = L.pop(0);
        else:
            A[index] = R.pop(0);
        index = index+1;
    for i in range(0, len(L),1):
        A[index] = L.pop(0);
        index = index + 1;
    for i in range(0, len(R), 1):
        A[index] = R.pop(0);
        index = index + 1;

def _sort(array, l, r):
    if r >l:
        _sort(array,l, int((l+r)/2));
        _sort(array, int((l+r)/2+1),r);
        _merge(array, l, int((l+r)/2),r)

def sort(array):
    _sort(array,0, len(array)-1);
    return array;

print(sort([1,2,3,4,5]));
print(sort([5,4,3,2,1]));
print(sort([5,4,1,2,3]));
print(sort([1]));
print(sort([]));
```

3 2.3-3

Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2, & \text{if } n = 2 \\ 2T(n/2) + n, & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = nlg(n)$

For $n = 2$: $T(2) = 2 = 2lg(2) = nlg(n)$

For $n = 4$: $T(4) = 2T(2) + 4 = 2 * 2lg(2) + 4 * 1 = 4 * lg(2) + 4 * lg(2) = 4 * lg(4) = nlg(n)$

Assume, that for all $n \leq k$ T(n)=nlg(n). For $n = k * 2$: $T(k * 2) = 2T(k * 2/2) + 2k = 2k * lg(k) + 2k = 2k * lg(k) + 2k * lg2 = 2k * (lgk + lg2) = 2k * lg(2k)$

4 2.3-4

We can express insertion sort as a recursive procedure as follows. In order to sort A[1..n], we recursively sort A[1..n-1] and than insert A[n] into the sorted array A[1..n-1]. Write a recurrence for the worst-case running time of this recursive version of insertion sort.

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ T(n-1) + n - 1, & \text{if } n > 1 \end{cases}$$

5 2.3-5

Reffering back to the searching problem (see Exercise 2.1-3), observe that if the sequence A is sorted, we can check the midpoint of the sequence against v and eliminate half of the sequence from futher consideration. The binary search algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive? for binary search. Argue that the worst-case running time of binary search is $\Theta(lg(n))$

```
def _search(array, l, r, value):
    if l==r:
        if array[l] == value:
            return l;
    else:
        middle = int((l+r)/2);
        if array[middle] >= value and array[l]<=value:
            return _search(array, l, middle, value);
        if array[middle] < value and array[r]>=value:
            return _search(array, middle+1, r, value);
        return None;

def search(array, value):
    if len(array) == 0:
        return None;
    return _search(array,0, len(array)-1, value);

print(search([1,2,3,4,5], 1));
print(search([1,2,3,4,5], 5));
print(search([1], 1));
print(search([1], 5));
print(search([], 5));
```

Recurrence: $T(n) = \begin{cases} 1, & \text{if } n = 1 \\ T(n/2) + 1, & \text{if } n > 1 \end{cases}$

let $n = 2^k$:

$n = 1$: $T(1) = 1 = 1 + lg(0) = \Theta(lg(n) + 1)$ $n = 2$: $T(2) = T(1) + 1 = lg(1) + 1 + 1 = lg(1) + lg(2) = lg(2) = \Theta(lg(n))$ $n = k * 2$: $T(2k) = T(k) + 1 = lg(k) + lg(2) = lg(2K) = \Theta(lg(n))$

6 2.3-6

Observe that the while loop of lines 5-7 of the Insertion sort procedure in Section2.1 uses a linear search to scan (backwards) through the sorted subarray A[1..j-1]. Can we use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time of insertion sort $\Theta(n * lg(n))$

```
def binary_search(array, l, r, value):
        middle = int((l+r)/2);
        if array[middle] >= value:
            if array[l]<=value:
                return _search(array, l, middle, value);
            else:
                return l;
        else:
            if array[r]>=value:
                return _search(array, middle+1, r, value);
            else:
                return r+1;

def sort(array):
    for i in range(1, len(array), 1):
        tempValue = array[i];
        index = binary_search(array, 0, i-1, tempValue);
        array.insert(index, tempValue);
        array.pop(i+1);
    return array;

print(sort([1,2,3,4,5]));
print(sort([5,4,3,2,1]));
print(sort([1]));
print(sort([]));
```

If we use List structure for a storage, insertion and deleting take $\Theta(1)$, so $T(n) = \Theta(n * lg(n))$ For Array structure there are still series of swaps required, so so $T(n) = \Theta(n^2)$

7 2.3-7

Describe a $\Theta(nlg(n))$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.

```
#################
# Merge Sort
#################

def _merge(A,p,r,q):
    L = A[p:r+1];
    R = A[r+1:q+1];
    index = p;
    while len(R)>0 and len(L)>0:
        if (L[0]<R[0]):
            A[index] = L.pop(0);
        else:
            A[index] = R.pop(0);
        index = index+1;
    for i in range(0, len(L),1):
        A[index] = L.pop(0);
        index = index + 1;
    for i in range(0, len(R), 1):
        A[index] = R.pop(0);
        index = index + 1;

def _sort(array, l, r):
    if r >l:
        _sort(array,l, int((l+r)/2));
        _sort(array, int((l+r)/2+1),r);
        _merge(array, l, int((l+r)/2),r)

def sort(array):
    _sort(array,0, len(array)-1);
    return array;

################
# Binary Search
################
def binarySearch(array, l, r, value):
    if l==r:
        if array[l] == value:
            return l;
    else:
        middle = int((l+r)/2);
        if array[middle] >= value and array[l]<=value:
            return binarySearch(array, l, middle, value);
        if array[middle] < value and array[r]>=value:
            return binarySearch(array, middle+1, r, value);
        return None;

def findIfSumExists(array, value):
    array = sort(array); #Theta(n*lg(n))
    for i in range(0, len(array)-1, 1): #Theta(n)
        nextElementValue = value - array[i];
        if (binarySearch(array, i+1, len(array)-1, nextElementValue)) != None: #
            return True;
    return False;

print(findIfSumExists([1,2,3,4,5], 6));
print(findIfSumExists([1,2,3,4,5], 1));
print(findIfSumExists([5,2,3,4,1], 5));
print(findIfSumExists([5], 5));
print(findIfSumExists([], 5));
```