1. **(a)**
2. , not applicable
3. , not applicable
4. The combination cost is negative, not applicable

**(b)** The recursion tree has layers. The ith layer has node. The combination cost at each layer is . So, the total complexity is .

**(c)** Assume

Then

So,

**2.**

# array is the input list

def Peak\_1D(array):

maximum = array[0]

for i in array[1:]:

if i > maximum:

maximum = i

return maximum

def Test():

array = [2, 3, 5, 4, 6, 7, 3]

print ('input example:')

print (array)

print ('result:')

print (Peak\_1D(array))

if \_\_name\_\_ == '\_\_main\_\_':

Test()

**3.**

import numpy as np

# array is the input array

def Peak\_2D(array):

return array.max()

def Test():

array = np.array([[1,2,3],[4,5,6],[7,8,9]])

print ('input example:')

print (array)

print ('result:')

print (Peak\_2D(array))

if \_\_name\_\_ == '\_\_main\_\_':

Test()

**4. (a)**

Assume that

For each part of , we need to multiply it with each part of . As both and have n parts, so the total number of multiplication is , the total number of summation is . The complexity of polynomial multiplication is .

**(b)** Divide the two polynomials into the following form:

Let

Then we have that

For n-order polynomial multiplication, we only need to calculate the three (n/2)-order polynomial multiplication, namely , , and . Some additional summation are required to calculate each part. By using Divide and Conquer method, we have that

According to master theorem, the final complexity is improved to

**5.**

# array is the input list

def QuickSort(array):

if len(array) <= 1:

return array

pivot = array[-1]

pivots = [i for i in array if i == pivot]

left = QuickSort([i for i in array if i < pivot])

right = QuickSort([i for i in array if i > pivot])

return left + pivots + right

# O(nlgn) algorithm

def Method\_1(array):

sorted\_array = QuickSort(array)

maximum = sorted\_array[-1]

minimum = sorted\_array[0]

for i in range(len(array)):

if (array[i] == maximum):

maximum\_id = i

elif (array[i] == minimum):

minimum\_id = i

return (minimum\_id, maximum\_id)

# O(n) algorithm

def Method\_2(array):

maximum = 0

minimum = 0

for i in range(1, len(array)):

if (array[i] < minimum):

minimum = i

elif (array[i] > maximum):

maximum = i

return (minimum, maximum)

def Test():

array = [1,2,5,8,6,14,-3]

print ('input example:')

print (array)

print ('O(nlgn) algorithm result:')

print (Method\_1(array))

print ('O(n) algorithm result:')

print (Method\_2(array))

if \_\_name\_\_ == '\_\_main\_\_':

Test()

**6.**

# pid is the input list

# O(n) algorithm

def Method\_1(pid):

k = int(len(pid) / 2)

for i in range(k):

if pid[2 \* i] != pid[2 \* i + 1]:

return pid[2 \* i]

return pid[-1]

# O(lg n) algorithm

def Method\_2(pid):

start = 0

finish = len(pid)

while (finish - start > 2):

length = finish - start

k = int((start + finish) / 2)

if (k % 2):

k -= 1

if (pid[k] == pid[k + 1]):

start = k + 2

else:

finish = k

return pid[start]

def Test():

pid = [5, 5, 12, 12, 18, 18, 19, 21, 21, 50, 50]

print ('O(n) algorithm result: %d' % Method\_1(pid))

print ('O(lg n) algorithm result: %d' % Method\_2(pid))

if \_\_name\_\_ == '\_\_main\_\_':

Test()