**(a)**

When the size of an array grows to the max bounds, the double growth approach states that it should expand to double its original size.

So, if the (n-1) element, where n is the array size, is filled, construct a new array with twice the size of the original array and copy the data to it.

1. Creating a new array of double-sized take -> O(1) time
2. It takes O(n) time to copy data from one array to another.

Because copying an item of size n (from o to n-1) requires O(n) time.

1. total amount of time spent -> O(n) + O(1)

**🡪 O(n)**

**(b)**

Given,

B[i] = A [2i] + A [2i + 1] for i=0 to (n/2)-1

If B has size of 1 then, output 🡪 B(0)

Reduce size of array A of half until size reaches 1

Therefor,

While(s>1){

SizeB = size(A/2) 🡪 n/2

Array A[]

Array B[]

// Save the values of arrays A and B using the given algorithm, which loops from i=0 to (n/2)-1.

For(i=0 to (n/2)-1) {

B[i] = A[2i] + A[2i+1]

}

}

Therefore 🡪 basic structure describes

1. Data is copied from array A to array B in a linear loop for i=o to (n/2)-1 times, resulting in a total time of (n/2) time 🡪 O(n)
2. The main while loop continues until the size of array A reaches 1, at which point T(n) = T(n/2)

n/2k = 1

n = 2k  k = log2n

so total log2n comparison so = O(log2n)

= O(n) \* O(log2n)

= O(n \* log2n)