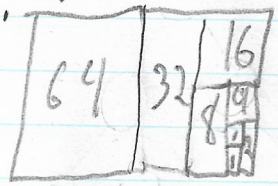
1. a)



When our array grows we need to make as many pushes as the size of our array each time it grows, but only need to grow when we run out of space

we can see that if $h = 64$, we need a further $32+16+8+4+2+1$ pushes, plus 64 actions to add the actual values

So, $h + 2^{K+1} - 1 = 3n - 1$

(h: from appending values)

$K = \log_2 h$, for pushes needed

$3n - 1 = O(h)$, so $T(h) = O(h)$

b) 0, 1, 2, 3, 4, 5, 6, 7

1_1, 5_2, 9_3, 13_4

6_5, 22_6

28_7

We know that summing them in order is running through n, and the other method takes just as many additions.

further, from the way I would program it seen left, it would have have a nested loop structure for it to be $> O(n)$

```
method(a){
  if a.size = 1
      return a[i]
  for(i=0; i<a.size; i++)
      b[i] = a[i*2] + a[i*2+1]
  return method(b)
```

thus it is $O(n)$