We saw the following algorithm for building a heap. 

\[ \text{Build} - \text{Heap}(A) \]

1. for \( i = \lfloor \text{length}[A]/2 \rfloor \) downto 1 do 
2. \( \text{Heapify}(A, i) \)

The following algorithm was suggested instead of the algorithm above.

\[ \text{Alt} - \text{Build} - \text{Heap}(A) \]

1. for \( i = 1 \) to \( \lfloor \text{length}[A]/2 \rfloor \) do 
2. \( \text{Heapify}(A, i) \)

1. Show an example where \( \text{Alt} - \text{Build} - \text{Heap}(A) \) will not result in a heap.
2. Is it possible to use \( \text{Alt} - \text{Build} - \text{Heap}(A) \) to build a heap by calling it more than once? If your answer is yes show such an algorithm. If your answer is no explain it.

Answers:
1. 

```
    1
   /\  
  2 3  
 /   / 
4   4
```

2. 

```
1   for i = 1 to length[A] do 
2     Alt - Build - Heap(A) 
```

The first call to Alt-Build-Heap will heapify node \( n \), the second call to Alt-Build-Heap will heapify node \( n - 1 \), and so on. It takes at most \( n \) calls to heapify all the nodes.

Note: The height of the tree is also an acceptable number of calls. The answer has to be given in terms of Alt-Build-Heap.