Procedure *Heapify* as we have seen in class is shown on the other side of this page.

The procedure was applied to the following array $A$ using the call $\text{Heapify}(A, 1)$. Answer the following questions.

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
5 & 7 & 10 & 16 & 3 & 12 & 11 \\
\end{array}
\]

1. Show the binary tree corresponding to the array before applying $\text{Heapify}(A, 1)$.
2. Show the binary tree corresponding to the array after applying $\text{Heapify}(A, 1)$.
3. Did $\text{Heapify}(A, 1)$ achieve its purpose in the heapsort algorithm? Explain why.

Answers:

3. No. $\text{Heapify}(A, 1)$ should bring the largest element in $A$ to $A[1]$. This did not happen because the left and right subtrees of node 1 did not have the heap property.
Heapify(A, i)

1    l ← Left(i)
2    r ← Right(i)
4        then largest ← l
5        else largest ← i
6    if r ≤ heap − size[A] and A[r] > A[largest]
7        then largest ← r
8    if largest ≠ i
9        then exchange A[i] ↔ A[largest]
10   Heapify(A, largest)
ECE608, Spring 2015, Quiz 5

First Name: ___________________ Last Name: ____________________

I certify that I have neither given nor received unauthorized aid on this quiz.

Signed: ___________________

Use only the space provided on this page to answer the following question.
Do not write your answers on the other side of the page.

Procedure *Heapify* as we have seen in class is shown on the other side of this page.

The procedure was applied to the following array \( A \) using the call *Heapify*(\( A,1 \)). Answer the following questions.

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
4 & 8 & 12 & 18 & 11 & 2 & 15 \\
\end{array}
\]

1. Show the binary tree corresponding to the array before applying *Heapify*(\( A,1 \)).
2. Show the binary tree corresponding to the array after applying *Heapify*(\( A,1 \)).
3. Did *Heapify*(\( A,1 \)) achieve its purpose in the heapsort algorithm? Explain why.

Answers:

1. 

```
          4
         / \  
        12 8
       / / \ \\
      18 11 2
```

2. 

```
          12
         /  \\
        15 8
       / / \\
      11 18 2
```

3. No. *Heapify*(\( A,1 \)) should bring the largest element in \( A \) to \( A[1] \). This did not happen because the left and right subtrees of node 1 did not have the heap property.
Heapify(A, i)

1   l ← Left(i)
2   r ← Right(i)
4       then largest ← l
5       else largest ← i
6   if r ≤ heap − size[A] and A[r] > A[largest]
7       then largest ← r
8   if largest ≠ i
9       then exchange A[i] < − > A[largest]
10      Heapify(A, largest)