HW

Verify the `max_heapify(int x[],int i,int h_size)` by using CBMC

- `x[]` is an array to contain a max-heap
- `i` is the index to the node that may violate the max-heap property
- `h_size` is a total number of nodes in the max-heap:
- `max_heapify` makes a subtree whose root is `x[i]` a max heap with the following assumptions.

Assumptions

1. A given tree satisfies the shape property

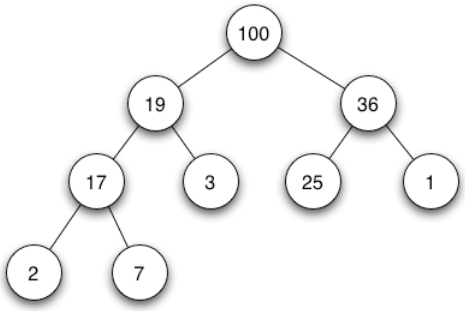
2. The right and left sub-trees of node i are max heaps, but that `x[i]` may be smaller than its children

3. The size of `x[]` is 8.

To do list:

- Describe your assertion check routine in detail
  - `max_heapify` should guarantee that the final subtree whose root node is x[i] should be a max heap.
  - Also, `max_heapify` should guarantee that the other part of a tree should not change.
- Describe your environment model in detail
- Describe run-time parameters of CBMC
- Report verification results (i.e., time, memory (you can use `top` utility), assert violation, size of generated SAT formula, etc)

- A *max heap* is a heap data structure created using a binary tree with two constraints:
  - *The shape property*: the tree is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.
  - *The max-heap property:* each node is greater than or equal to each of its children according to a comparison predicate defined for the data structure.



Max heap can be implemented using an array as follows (note that array index starts from 1):

Index	1	2	3	4	5	6	7	8	9
value	100	19	36	17	3	25	1	2	7

```c
/* Example code */
#include<stdio.h>
#define MAX 16
#define H_SIZE 10
#define parent(i)(i/2)
#define left(i) (2*i)
#define right(i)(2*i+1)

/* Ignore the first 0, since max heap
 contents start at index 1 */
int a[MAX] = {0,16,4,10,14,7,9,3,2,8,1,};

void  max_heapify(int x[],int i,int h_size){
  int largest, tmp;
  int l=left(i);
  int r=right(i);

  if (l<=h_size &&  x[l]>x[i]) largest=l;
  else  largest=i;
  if(r<=h_size && x[r]>x[largest]) largest=r;
  if (largest!=i)  {
    tmp=x[i];
    x[i]=x[largest];
    x[largest]=tmp;
    max_heapify(x,largest,h_size);
  }
}
```

```c
int main(){
  int i;
  max_heapify(a,2,H_SIZE);
  for (i=1;i<=H_SIZE;i++) printf("%d ",a[i]);
  return 0;
} /* Output: 16 14 10 8 7 9 3 2 4 1 */
```