1. **Node deletion: 40 points**

```
typedef struct n {
            int       value;
            struct n *prev;
            struct n *next;
        } node;

node  *head, *x;
```

(a) (20 points:) Given the declarations above, what are the **two** statements needed to delete a node pointed to by `x` if it is a node in the middle of a doubly linked list pointed to by `head`.

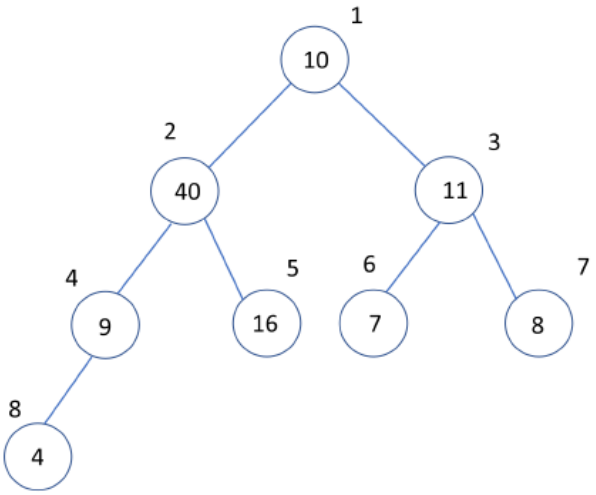
```
%   x->prev->next = x->next;

%   x->next->prev = x->prev;
```

(b) (20 points:) Complete the following function to delete the node pointed to by `x` from the list pointed to by `head`. You can assume that `x` points to a valid node in the list. You do need to check if `x` is the first or last node or the only node of the list. And do free up the deleted node.

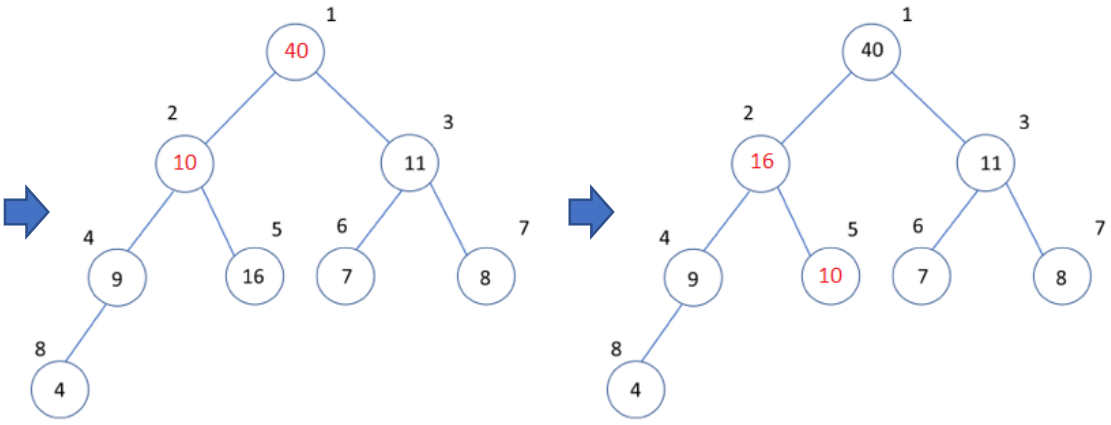
```
deleteNode( node **head, node *x )
{
%   if (*head == x)
%     *head = x->next;
%   else {
%      x->prev->next = x->next;
%      if (x->next != NULL)
%         x->next->prev = x->prev;
%   }
%   free(x);
```

2. **Max Heap: 30 points**

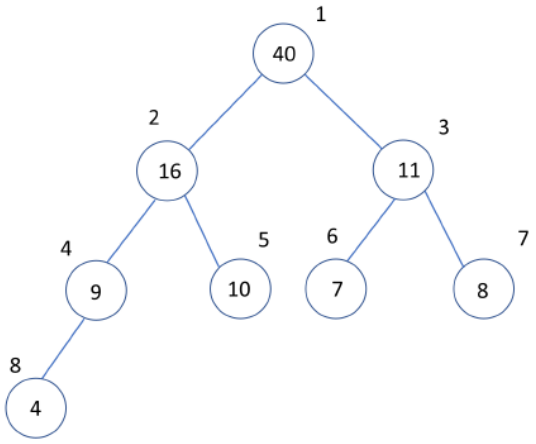
The `max_heapify` procedure from the book ensures that the tree rooted at index i (i.e. A[i]) has the heap property by recursively checking that the values of the children of node i is less than that of node i. If at a particular instant, you are given the following tree configuration:



sketch out the **sequence of changes** when a call to `max_heapify(A, 1)` is made. Redraw the tree for each change.



3. **Priority Queue: 30 points**



(a) (10 points): Given the priority queue above, what does `extract_max ( A )` return?

% 40

(b) (20 points): Sketch the new configuration of the priority tree after `extract_max ( A )` is completed. Also sketch the intermediate configuration(s) as necessary.