Problem 1

- A heap of size n has at most $\lceil n/2^{h+1} \rceil$ nodes with height h. **Key Observation**: For any $n > 0$, the number of leaves of nearly complete binary tree is $\lceil n/2 \rceil$. *Proof by induction* **Base case**: Show that it's true for $h = 0$. This is the direct result from above observation. **Inductive step**: Suppose it's true for $h - 1$. Let N_h be the number of nodes at height h in the n-node tree T . Consider the tree T' formed by removing the leaves of T . It has $n' = n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$ nodes. Note that the nodes at height h in T would be at height $h - 1$ in tree T'. Let N'_{h-1} denote the number of nodes at height $h - 1$ in T', we have $N_h = N'_{h-1}$. By induction, we have $N_h = N'_{h-1} = \lceil n'/2^h \rceil = \lceil \lfloor n/2 \rfloor /2^h \rceil \leq \lceil (n/2)/2^h \rceil = \lceil n/2^{h+1} \rceil$.

  Remark: Initially, I give following proof, which is flawed. The mistake is made in the claim "The remaining nodes have height strictly more than h. To connect all subtrees rooted at node in S_h, there must be exactly $N_h - 1$ such nodes." To see why it fails, here is a counterexample. Consider $h = 2$. The black two nodes has height 2, and $N_h = N_2 = 2$. The red node, among "The remaining nodes", has height 1, which is less than 2. Also, the number of nodes (blue nodes) connecting two black nodes is 2, instead of $N_2 - 1 = 1$.

  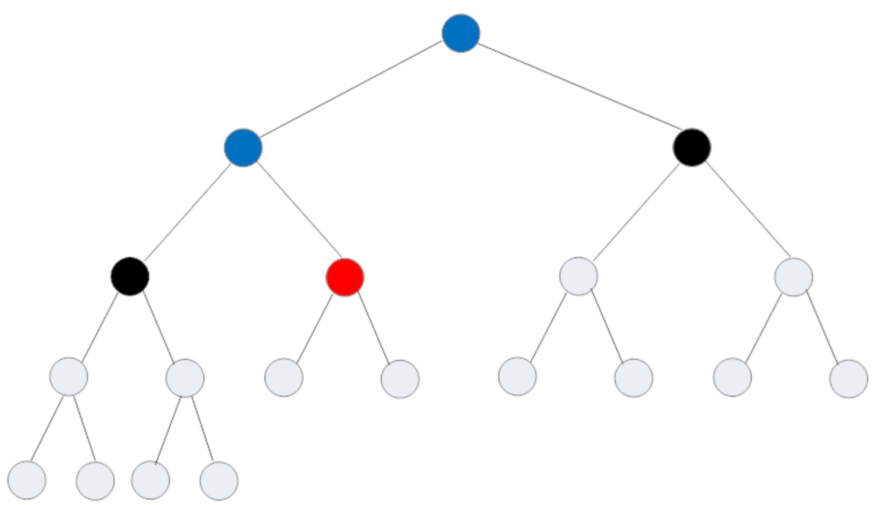

  Figure 1: Counterexample

  Flawed Proof: **Property 1:** Let S_h be the set of nodes of height h, subtrees rooted at nodes in S_h are disjoint. In other words, we cannot have two nodes of height h with one being an ancestor of the other. **Property 2** All subtrees are complete binary trees except for one subtree. Now we derive the bounds of n by N_h given these two properties. Let N_h be the number of nodes of height h. Since $N_h - 1$ of these subtrees are full, each subtree of them contains exactly $2^{h+1} - 1$ nodes. One of the height h subtrees may be not full, but contain at least 1 node at its lower level and has at least 2^h nodes. The remaining nodes have height strictly more than h. To connect all subtrees rooted at node in S_h, there must be exactly $N_h - 1$ such nodes (Flawed here!). The total of nodes is at least $(N_h - 1)(2^{h+1} - 1) + 2^h + N_h - 1$ while at most $N_h 2^{h+1} - 1$, So

$$(N_h - 1)(2^{h+1} - 1) + 2^h + (N_h - 1) \leq n \leq N_h(2^{h+1} - 1) + N_h - 1 \tag{1}$$
$$\Rightarrow -2^h \leq n - N_h 2^{h+1} \leq -1 \tag{2}$$
$$\Rightarrow \text{The fraction part of } n/2^{h+1} \text{ is larger than or equal to } 1/2 \tag{3}$$
$$\Rightarrow N_h \leq \lceil n/2^{h+1} \rceil \tag{4}$$

- A heap with n elements has a height of $\Theta(\log n)$. ($\Theta(n)$ is a typo in problem sheet).

Problem 2

- min-heap, if elements are sorted by ascending order; max-heap, if elements are sorted in descending order.

- Show that, with the array representation for storing an n-element heap, the leaves are the nodes indexed by $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

  Proof. **Basis:** The claim is trivially true for $n = 1$. **Inductive step:** Suppose the claim is true for $n = k (k \geq 1)$. That is, the leaves are the nodes indexed by $\lfloor k/2 \rfloor + 1, \lfloor k/2 \rfloor + 2, \ldots, k$. If k is even,then its parent $\lfloor k/2 \rfloor$ has only one child. In this case, when $n = k + 1$, $\lfloor k/2 \rfloor$ will have two nodes, while others remain unchanged. Since $\lfloor k/2 \rfloor = \lfloor (k + 1)/2 \rfloor$ when k is even, the claim is true for $n = k + 1$ when k is even. If k is odd, when $k \to k + 1$, the new node will be appended to the tree as a child of node $\lfloor k/2 \rfloor + 1$, while others remain unchanged. So the leaves are indexed by $\lfloor k/2 \rfloor + 2, \ldots, k + 1$. Because $\lfloor k/2 \rfloor + 2 = \lfloor (k + 1)/2 \rfloor + 1$ when k is odd, the claim is true for $n = k + 1$ given k is odd. By mathematical induction, the claim is true for all $n \geq 1$. □

Problem 3 See Figure below.

Problem 4 Suppose the input stored in variables A, B, C, D, E.

Algorithm 1 Sort five elements within seven comparisons

```
if A > B ( 1st comparison) then
  swap A and B so that A < B
end if
if C > D (2nd comparison) then
  swap C and D so that C < D
end if
if A > C (3rd comparison) then
  swap C and A so that A < C ≤ B and A ≤ D
  swap B and D so that A ≤ C ≤ D and A ≤ B
end if{ So far, we have A ≤ C ≤ D and A ≤ B}
if  E < C (4th comparison)  then
  if  E > A (5th comparison) then
    F ← E
    E ← D
    D ← C
    C ← F
  else
    F ← E
    E ← D
    D ← C
    C ← A
    A ← F
  end if{ note that we still have A ≤ B}
else
  if  E < D (5th comparison) then
    swap E and D so that A ≤ C ≤ D ≤ E
  end if
end if
if  B < D (6th comparison) then
  if  B < C (7th comparison) then
    return  A, B, C, D, E
  else
    return  A, C, B, D, E
  end if
else
  if  B < E (7th comparison) then
    return  A, C, D, B, E
  else
    return  A, C, D, E, B
  end if
end if
```

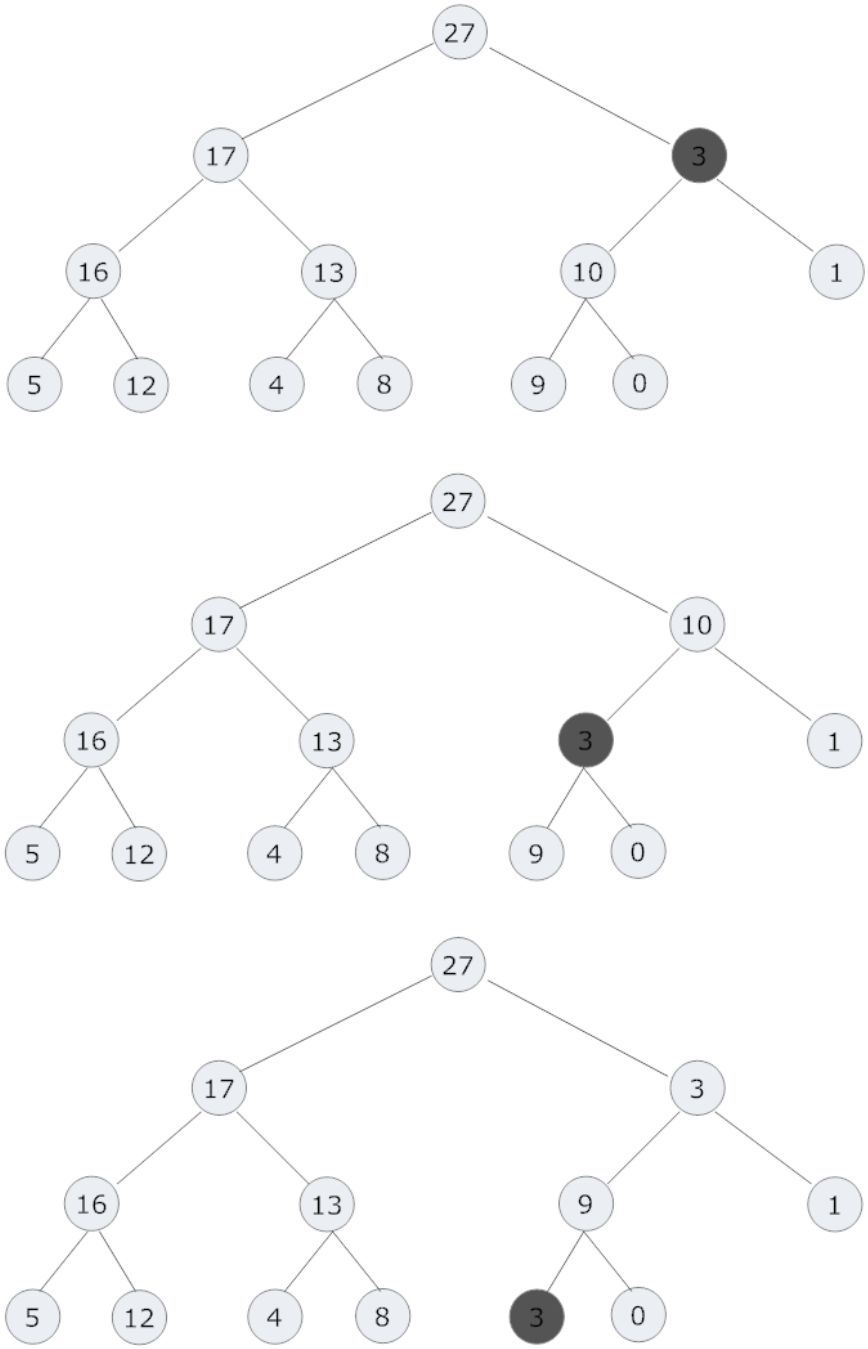

Figure 2: Solution to Problem 3