Tutorial 1

Instructor: Meng-Fen Chiang

COMPSCI: WEEK 9.6

https://ankechiang.github.io/cs220_swu.html#week9

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

OUTLINE

- Question 1: Complexity
- Question 2: Complexity
- Question 3: Mergesort
- Question 4: Quicksort
- Question 5: Quickselect
- Question 6: Heapsort
- Question 7: Binary Search Tree
- Question 8: Heapsort (Proof)

Question 1

- Prove that $T(n) = 1000n^4 + 65n^3 + 3n^2 + 10n$ is both $\mathrm{O}(n^4)$ and $\mathrm{O}(n^5)$

$\lim_{n\to\infty} \frac{1000n^4+65n^3+3n^2+10n}{n^4} = 3$. This means $T(n)$ is $\Theta(n^4)$, and is also $O(n^4)$.

$\lim_{n\to\infty} \frac{1000n^4+65n^3+3n^2+10n}{n^5} = 0$. This means $T(n)$ is $O(n^5)$.

Question 1 (Contd.)

- Determine the asymptotic relationship between $f(n)$ and $g(n)$ using limit rule.

$$f(n) = n\ln(7n) \text{ and } g(n) = 10n^2$$

$\lim_{n\to\infty} \frac{n\ln(7n)}{10n^2} = 0$. This means $f \in O(g)$ and $g \in \Omega(f)$.

Question 2

- Let $T(n) = (n^{0.01} + \log_2 n)^2$ be processing time of an algorithm for input of size n. Which is the asymptotic time complexity of this algorithm, $\Theta(n^{0.02})$ or $\Theta\big((\log n)^2\big)$?

$T(n) = n^{0.02} + 2n^{0.01}\log_2 n + (\log_2 n)^2$. The dominant term is $n^{0.02}$, because $\log_2 n < n^k$ where $k > 0$. We can show this using limit rule and L'Hopital's rule:

$$\lim_{n\to\infty} \frac{\log_2 n}{n^k} = \lim_{n\to\infty} \frac{\ln n}{\ln 2 \cdot n^k} \qquad \text{(Change the base of logarithm)}$$

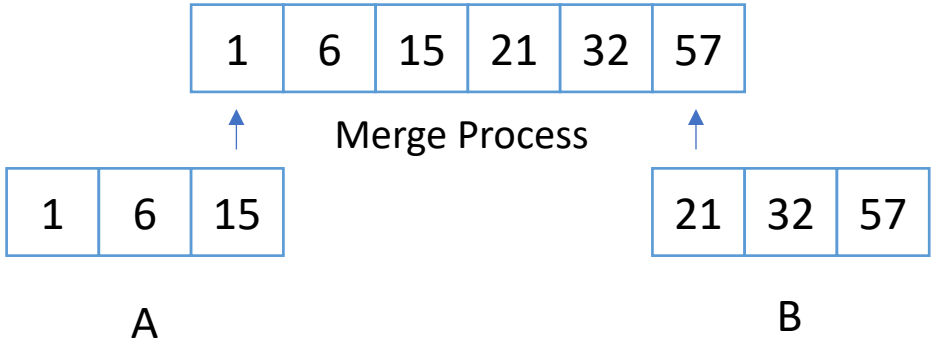
$$= \lim_{n\to\infty} \frac{1}{\ln 2 \cdot n \cdot kn^{k-1}} \qquad \text{(Apply L'Hopital's rule)}$$

$$= \lim_{n\to\infty} \frac{1}{\ln 2 \cdot kn^k} \qquad \text{(Because} k > 0, n^k \to \infty\text{)}$$

$$= 0$$

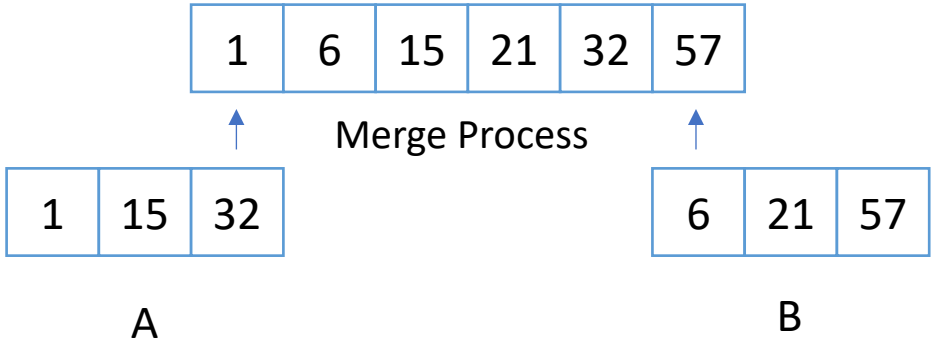
Question 3

- What is the **minimum** and maximum number of comparisons needed when merging two nonempty sorted lists of size n into a single list?
- **Minimum number of comparisons**: This happens when all elements in one list are smaller than all those in the other list. Denote the two lists as A and B and let's assume all elements in A are smaller than those in B. Then, we only need to compare all element in A with the first element in B. This requires n comparisons.



Question 3

- What is the minimum and **maximum** number of comparisons needed when merging two nonempty sorted lists of size n into a single list?
- **Maximum number of comparisons**: This happens when we need to zigzag between the two lists. The total number of comparisons is $2n - 1$.



Question 4

- Determine the order of the list after partitioning [41, 30, -1, 20, 15, 77, 10], assume the pivot is 20.

- Step1: First, we swap the pivot with the first element in the list.

[20, 30, -1, 41, 15, 77, 10]

- Step2: Next, we have the two pointers L and R starting on each end of the list and looks for elements bigger than the pivot and smaller than the pivot respectively. L pointer will find 30 and R pointer will find 10 for the first time. Swap 30 and 10.

[20, 10, -1, 41, 15, 77, 30]

Question 4 (Contd.)

- Step3: Continue to move L and R will lead to L finding 41 and R finding 15. Swap 41 and 15.

[20, 10, -1, 15, 41, 77, 30]

- Step4: Now, when the R moves to the left again, it will collide with L. This is when we swap this indexed element with the pivot.

[15, 10, -1, 20, 41, 77, 29]

At this point, all elements smaller than the pivot are on its left and all elements larger than the pivot are on its right. The first partition is done.

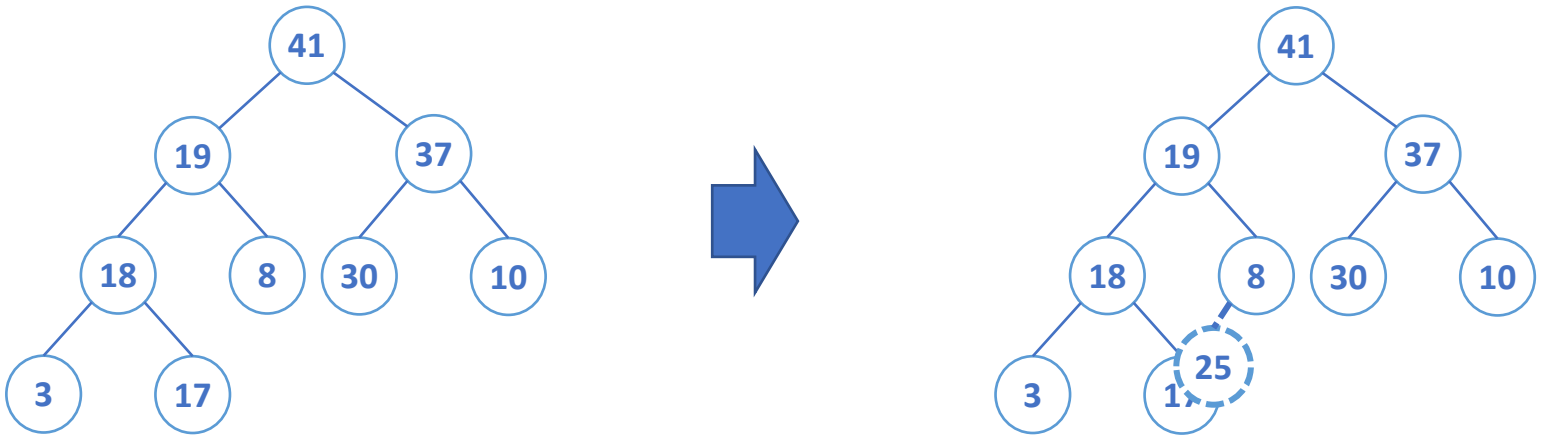
Question 5

- Use Quickselect to find the 3-rd largest element of the list [25, 65, 50, 21, 2, 67, 70, 31, 15, 8]. Show the state of the list after each step. Assume the first element is taken as the pivot.
- Step1: Select 25 as the pivot, partition into **[2, 8, 15, 21], 25, [67, 70, 31, 50, 65]**. We are looking for the 3rd biggest to recurse into the **right hand side**.
- Step2: Then, partition [67, 70, 31, 50, 65] with pivot 67 and get **[50, 65, 31], 67, [70]**. We recurse into the **left hand side** to find the 3rd largest (the largest within the left sublist).
- Step3: Pick 50 as the pivot and we partition this into **[31], 50, [65]**. So 65 is the largest element in this sublist, and thus the 3rd largest in the original list.

Question 6.1

- Consider the following maximum heap: [41, 19, 37, 18, 8, 30, 10, 3, 17]. Insert 25 to the heap.
- Step1: We always insert new nodes into a heap at the next available index so the list will be as follows after the insertion.

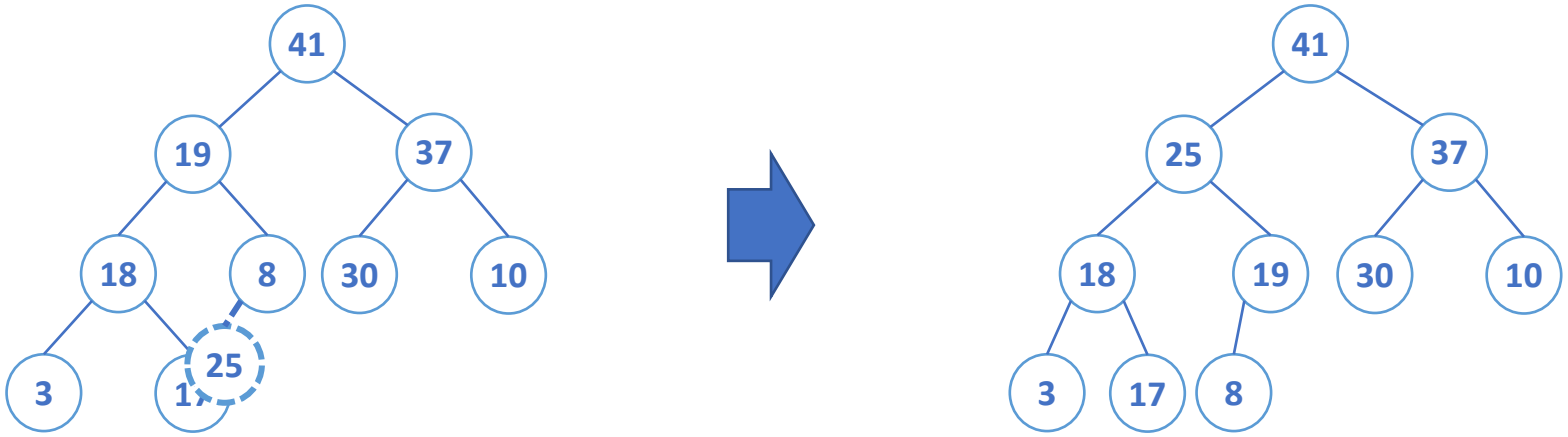
[41, 19, 37, 18, 8, 30, 10, 3, 17, 25]



Question 6.1 (Contd.)

- Consider the following maximum heap: [41, 19, 37, 18, 8, 30, 10, 3, 17]. Insert 25 to the heap.
- Step2: We then bubble the new node up to the correct place repeatedly comparing with its parent node. We obtain

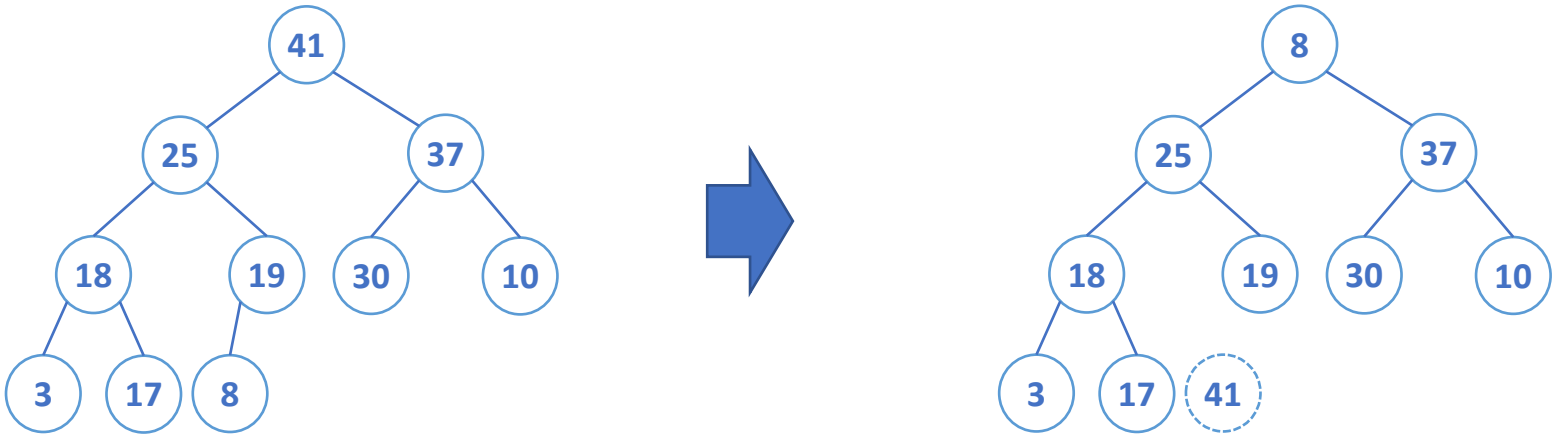
[41, 25, 37, 18, 19, 30, 10, 3, 17, 8]



Question 6.2

- Consider the following maximum heap: [41, 25, 37, 18, 19, 30, 10, 3, 17, 8]. Delete 41 from the heap.
- Step1: When removing a node, we always replace it with the last leaf. Thus we obtain the following list after removing 41.

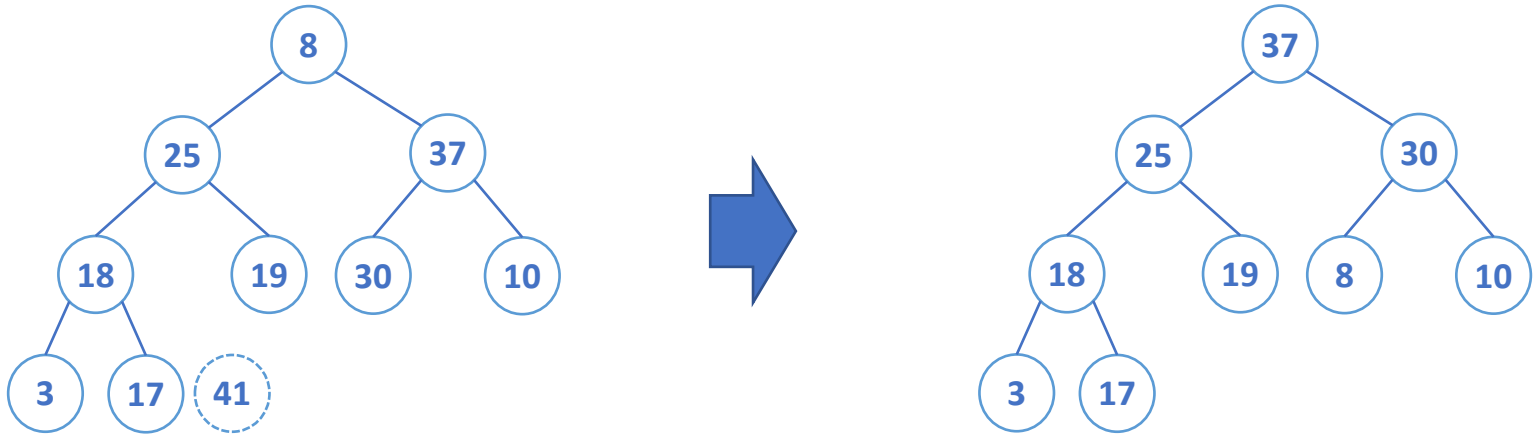
[8, 25, 37, 18, 19, 30, 10, 3, 17]



Question 6.2 (Contd.)

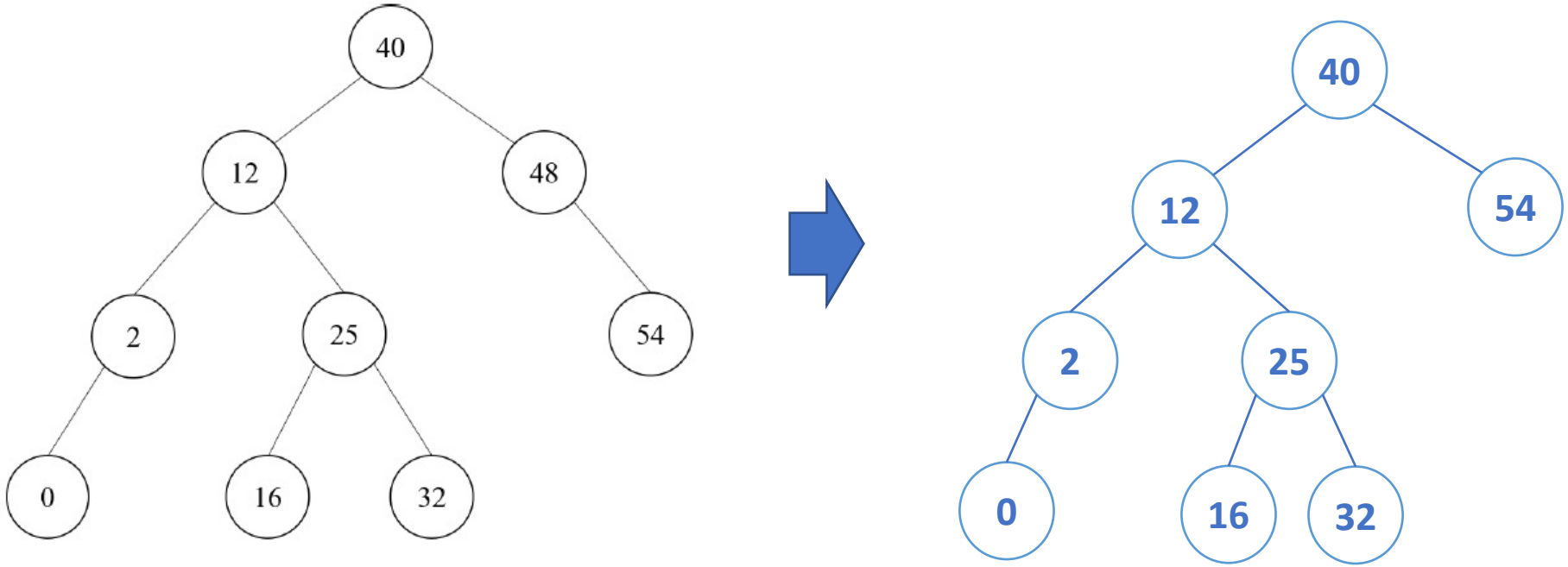
- Consider the following maximum heap: [41, 25, 37, 18, 19, 30, 10, 3, 17, 8]. Delete 41 from the heap.
- Step2: We then push the new root down to the correct place it should be by swapping it with its largest child until it is larger than any of its child node. We then obtain

[37, 25, 30, 18, 19, 8, 10, 3, 17]



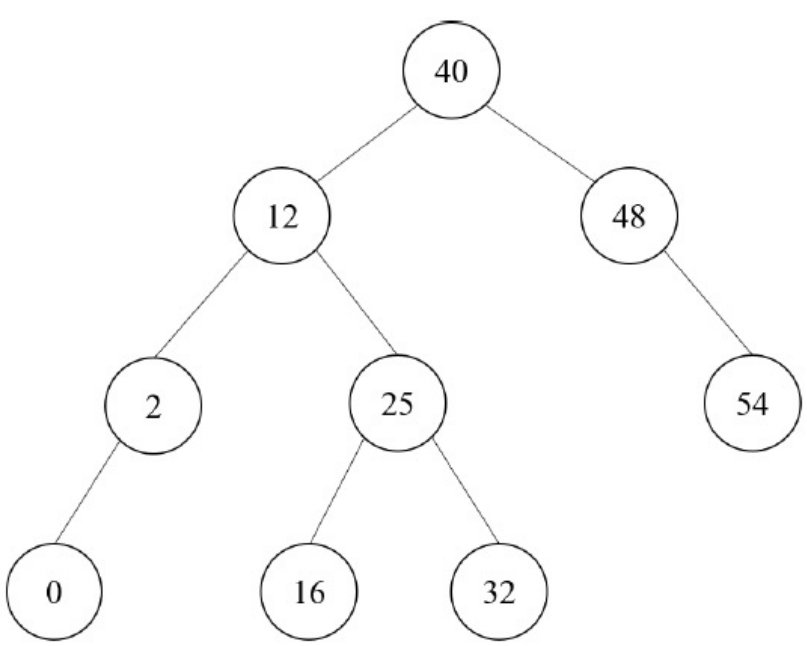
Question 7.1

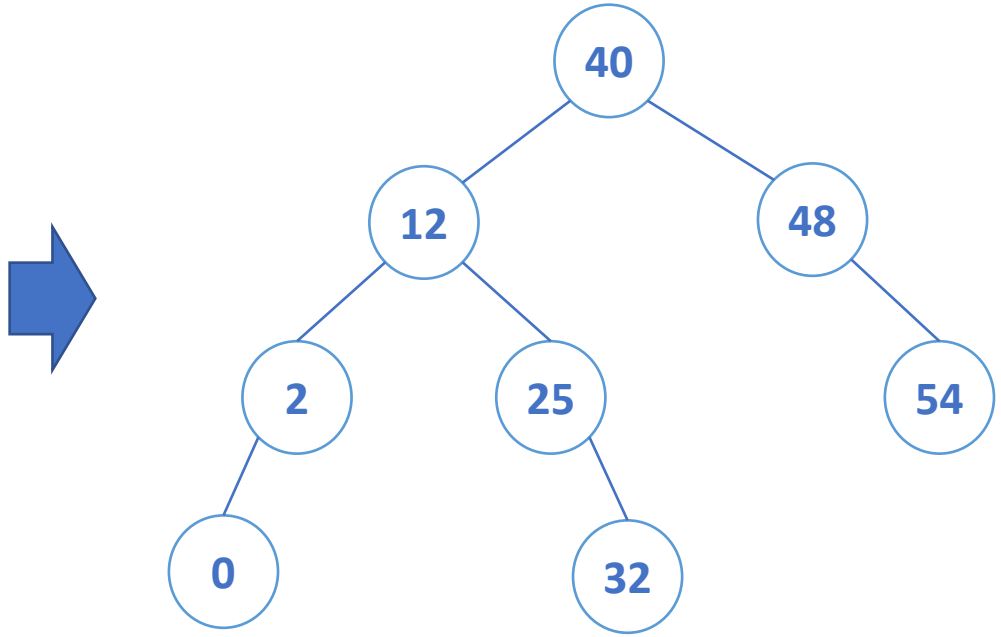
- Delete node 48 in the tree. Node 48 has only one child: delete the node, connect its child to its parent.



Question 7.2

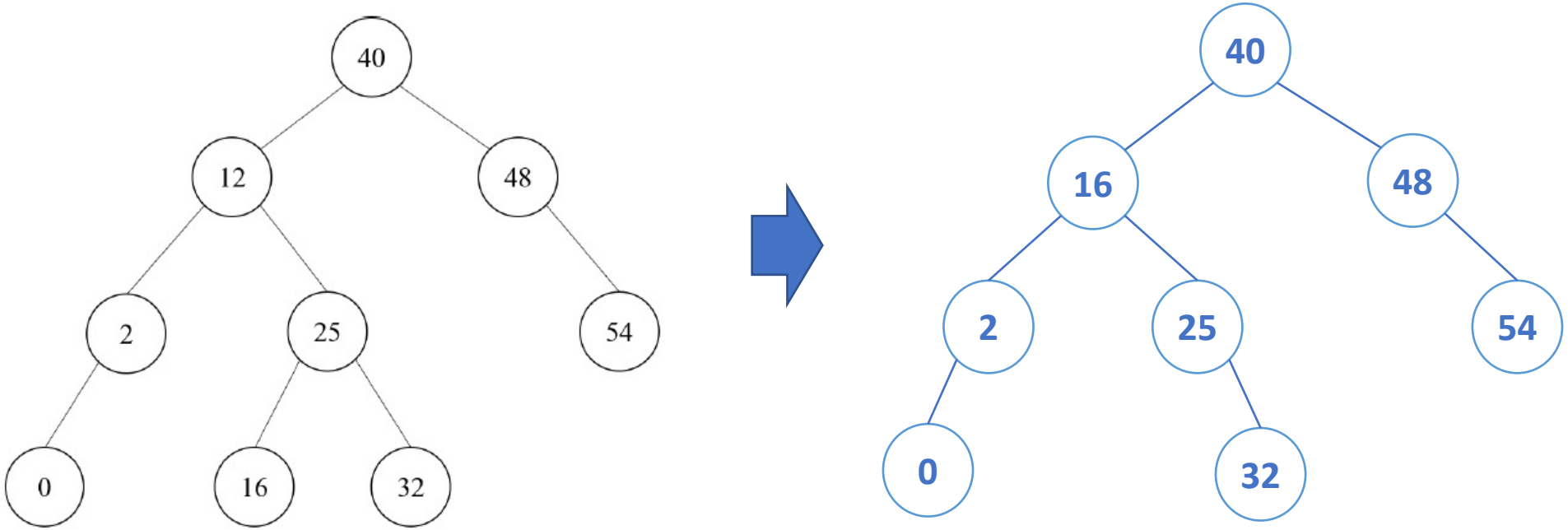
- Delete node 16 in the tree. Node 16 has no children: simply delete.





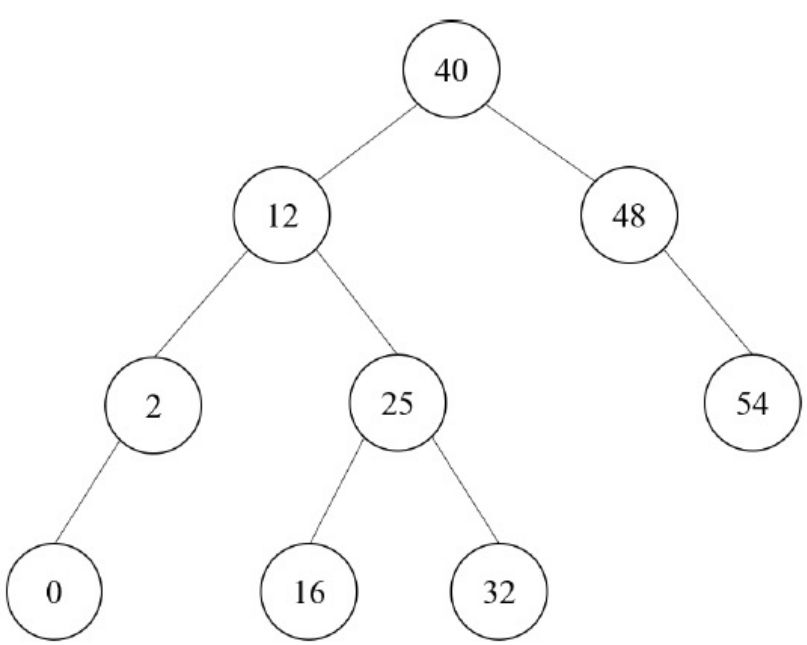
Question 7.3

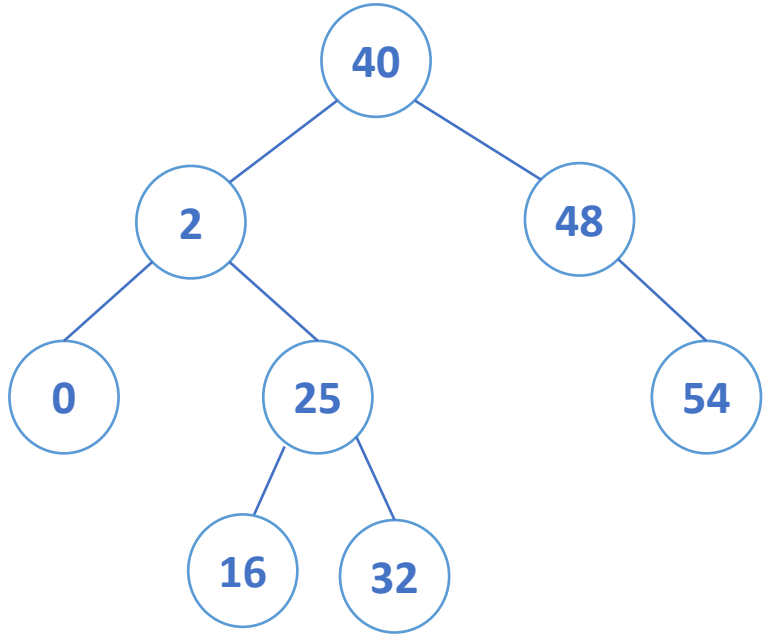
- Delete node 12 in the tree by using the **minimum** key in the right subtree. Node 12 has two children: find the **minimum** key K = 16 in the right subtree, delete that node, and replace the key of node 12 by K.



Question 7.4

- Delete node 12 in the tree by using the **maximum** key in the left subtree. Node 12 has two children: find the **maximum** key K = 2 in the left subtree, delete that node, and replace the key of node 12 by K.





Question 8.1

- Perform the linear time algorithm to construct a maximum heap on the following array. Please show the heap structure after each iteration.

A = [0, 54, 93, 2, 12, 32, 40, 16, 25, 51]

- A is not a max-heap, we need restore the max-heap property in the following
- There are 10 items in the heap so we start at index $i = \left\lfloor \frac{9-0}{2} \right\rfloor = 4$.
- Then, we bubble down each node. We will show the heap after bubbled down index i.

Question 8.1 (Contd.)

A = [0, 54, 93, 2, 12, 32, 40, 16, 25, 51]

- $i = 4$, A = [0, 54, 93, 2, 51, 32, 40, 16, 25, 12] as a result of bubbling down 12
- $i = 3$, A = [0, 54, 93, 25, 51, 32, 40, 16, 2, 12] as a result of bubbling down 2
- $i = 2$, A = [0, 54, 93, 25, 51, 32, 40, 16, 2, 12] as a result of bubbling down 93 (no change)
- $i = 1$, A = [0, 54, 93, 25, 51, 32, 40, 16, 2, 12] as a result of bubbling down 54 (no change)
- $i = 0$, A = [93, 54, 40, 25, 51, 32, 0, 16, 2, 12] as a result of bubbling down 0

Question 8.2

- Is A a minimum heap?

$$A = [0, 54, 93, 2, 12, 32, 40, 16, 25, 51]$$

- A is not a min-heap since the subtree rooted at node 54 contains value 2 and 12; and the subtree rooted at node 93 contains value 32 and 40.
- This violates the min-heap property where each subtree can only contain values bigger than or equal to the root of the subtree.

Question 8.2 (Contd.)

A = [0, 54, 93, 2, 12, 32, 40, 16, 25, 51]

- $i = 4$, A' = [0, 54, 93, 2, 12, 32, 40, 16, 25, 51] as a result of bubbling down 12 (no change)
- $i = 3$, A' = [0, 54, 93, 2, 12, 32, 40, 16, 25, 51] as a result of bubbling down 2 (no change)
- $i = 2$, A' = [0, 54, 32, 2, 12, 93, 40, 16, 25, 51] as a result of bubbling down 93
- $i = 1$, A' = [0, 2, 32, 16, 12, 93, 40, 54, 25, 51] as a result of bubbling down 54
- $i = 0$, A' = [0, 2, 32, 16, 12, 93, 40, 54, 25, 51] as a result of bubbling down 0 (no change)

Question 8.3

- Given the array A’ with min-heap property, delete the min value of A’.
- Please describe the heapification process and the outcome of the deletion operations on the min-heap constructed using A’

- Step 1: A’ = [51, 2, 32, 16, 12, 93, 40, 54, 25] after removing the root and replaced it with the last node
- Step 2: A' = [2, 12, 32, 16, 51, 93, 40, 54, 25] after bubbling down the new root 51 to the correct place

Question 8.4

- Prove that the height of a complete binary tree with n nodes is exactly $\lceil \log(n+1) \rceil - 1$ using mathematical induction.

- Base Step: A complete binary tree with 1 node is essentially the root. The height of the root $\lceil \log(n+1) \rceil - 1 = 0$
- Induction Step:
  - Suppose a complete binary tree with k nodes has height $h = \lceil \log(k+1) \rceil - 1$.
  - Then, the tree has at least $2^0 + 2^1 + 2^2 \ldots + 2^{h-1} + 1 = 2^h$ nodes (one node at last level); and at most $2^0 + 2^1 + 2^2 \ldots + 2^h + 1 = 2^{h+1}$ nodes (full nodes at the last level).

Question 8.4 (Contd.)

- Prove that the height of a complete binary tree with n nodes is exactly $\lceil \log(n+1) \rceil - 1$ using mathematical induction.
- To prove that the statement is true with $k+1$ nodes, we consider the tree with $k+1$ nodes by two cases (i) and (ii).
  i. In the case when the last level h is fully occupied, the tree already have $k = 2^{h+1} - 1$ nodes.
     Thus, the $k+1$-th node has to have depth $h+1 = \lceil \log(k+1) + 1 \rceil - 1$
  i. In the case when the last level h is not fully occupied, the new node will be on depth h.
     Thus, the $k+2$-th node will be any value within the lower/upper bound $2^h + 2 \leq k+1 \leq 2^{h+1} - 1$ with $h = \lceil \log(k+2) \rceil - 1$.
- Thus, the statement is true in both cases.