EECS 4020 Algorithms

HW4

I. Amortized Analysis

Q1

Flipping-Push Stack

- Push: insert an item ➔ cost = 1
- Flip: when # items = 2 power, all items are flipped upside down ➔ cost = # items

Show that amortized cost of Push = O(1)

Q1 [solution: aggregate method]

- Consider m Push operations
- Total cost for Push (except Flip) = m
- Total cost for Flip

  $= 2 + 4 + 8 + \ldots + K < \ldots + m/4 + m/2 + m$

  $< 2m$ [K = largest 2 power not exceeding m]
- Total cost $< 3m$ ➔ amortized cost = $O(1)$

Q1 [solution: accounting method]

- For each Push, we pay \$3 to the inserted item
  - \$1 is used immediately
  - \$2 is saved for the next Flip
- Whenever Flip occurs :
  - Half of the items are inserted since the last Flip
  - Each has \$2 → Enough to pay for the current Flip

Q1 [solution: potential method]

- Define a potential function ϕ where

  $\phi(D) = 2 * (\#\text{ items inserted since last Flip })$

- If the current Push does not cause a Flip :

  amortized cost $= \Delta\phi + \text{actual cost}$

  $= 2 + 1 = 3$

Q1 [solution: potential method]

- Define a potential function ϕ where

$$\phi(D) = 2 * (\ \#\text{ items inserted since last Flip }\ )$$

- If the current Push causes a Flip :

$$\text{amortized cost} = \Delta\phi + \text{actual cost}$$
$$= (-|D|) + (|D| + 1) = 1$$

Q2

Show that for Min-Heap :

- Insert : O(log n) amortized cost
- Extract-Min : O(1) amortized cost

How to do so with potential method?

Q2 [solution: potential method]

- Define a potential function ϕ where

$$\phi(H) = \Sigma_v \text{ node-depth}(v)$$

- For instance, if the heap H has 6 nodes,

$$\phi(H) = 1 + 2 + 2 + 3 + 3 + 3$$

Q2 [solution: potential method]

- For Insert :

$$\begin{aligned}\text{amortized cost} &= \Delta\phi + \text{actual cost}\\ &= \log|H| + \log|H|\\ &= O(\log n)\end{aligned}$$

- For Extract-Min :

$$\begin{aligned}\text{amortized cost} &= \Delta\phi + \text{actual cost}\\ &= -\log|H| + \log|H|\\ &= O(1)\end{aligned}$$

Q3

- Maintain n numbers in sorted arrays
  - Each array has size = 2 power
  - No two arrays have same size
- Only need to support Insert (no Delete)

How to Insert with O(log n) amortized cost ?

Q3 [solution: aggregate method]

From the requirement, we see that :

when we have two arrays with the same size,

we need to combine them into one

How to do so?

➔ Since arrays are sorted, we can use merge

Q3 [solution: aggregate method]

Consider n Insert operations

- # times to merge two arrays of size 2^k $\leq$ $n / 2^k$
- cost to merge two arrays of size 2^k $=$ 2^{k+1}

➔ Total cost to merge size 2^k arrays $\leq$ $2n$

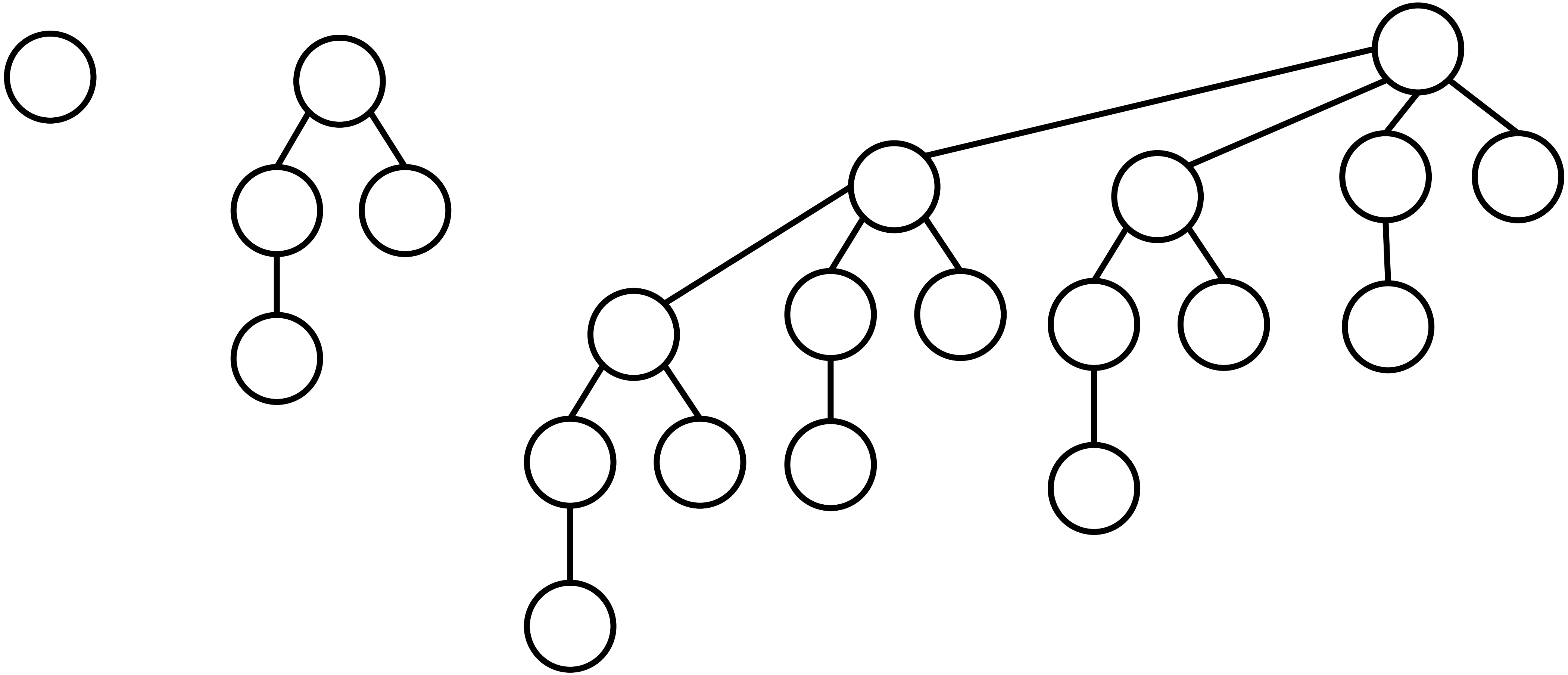
➔ Total cost for n Insert

$\leq$ $n + 2n \log n$ ➔ amortized cost = $O(\log n)$

II. Binomial Heap

Q1 [solution]

Draw a binomial heap with 21 nodes



Q2

Suppose we perform n Insert operations on a Binomial Heap (start from empty)

- Show that total time is O(n)

How to do the analysis ?

Q2 [solution: potential method]

- Define a potential function ϕ where

$$\phi(H) = \text{number of trees in } H$$

- For instance, if binomial heap H has 6 nodes,

$$\phi(H) = 2$$

Q2 [solution: potential method]

Consider an Insert operation:

- If no consolidation occurs, then

$$\text{amortized cost} = \Delta\phi + \text{actual cost}$$
$$= 1 + 1$$
$$= O(1)$$

Q2 [solution: potential method]

Consider an Insert operation:

- Else, suppose # trees changes from x to y:

$$
\begin{aligned}
\text{amortized cost} &= \Delta\phi + \text{actual cost} \\
&= (y - x) + (x - y + 1) \\
&= O(1)
\end{aligned}
$$

$\rightarrow$ Total cost for n operations $= O(n)$

Q3

Suppose we perform a sequence of n Insert or Union on a Binomial Heap (start from empty)

- Can we still show that total time is O(n) ?

Very tricky [Much harder than I thought …]

Q3 [solution]

- If we use the textbook definition of Binomial Heap, where after Union, we have to perform consolidation :

  ➔ Previous potential function does not work, as Union may take up to O(log n) time, and without changing # trees

Q3

- Yet, it does not mean that we cannot bound the total time to be O(n)
- We can separate the costs into two parts, and use different ways to bound :
  - Insert
  - Union

Q3 [solution]

- For Insert, we use the same potential method, so that total cost of this is $O(n)$
- For Union a heap of size x with a heap of size y :

$$\begin{aligned} \text{cost} &= \log x + \log y \\ &\le \text{cost to do } y \text{ increment to} \\ &\quad \text{a binary counter with value } x \end{aligned}$$

Q3 [solution]

- Thus, for Union,

  total cost $\leq$ cost to do n increment to a binary counter with value 0

➔ Total cost for either Insert or Union is O(n)

➔ Done!

Q3 [solution]

- Indeed, we may redefine a Binomial Heap, where after Union, we do not perform consolidation (just like Fibonacci Heap)
  - Do consolidation during Extract-Min

Q3 [solution]

- This idea is called Lazy Union
  - Extract-Min : O(log n) amortized cost
  - Others : O(1) amortized cost
- In this case, we can use the same potential function to bound both Insert and Union

III. Fibonacci Heap

Q1 [solution]

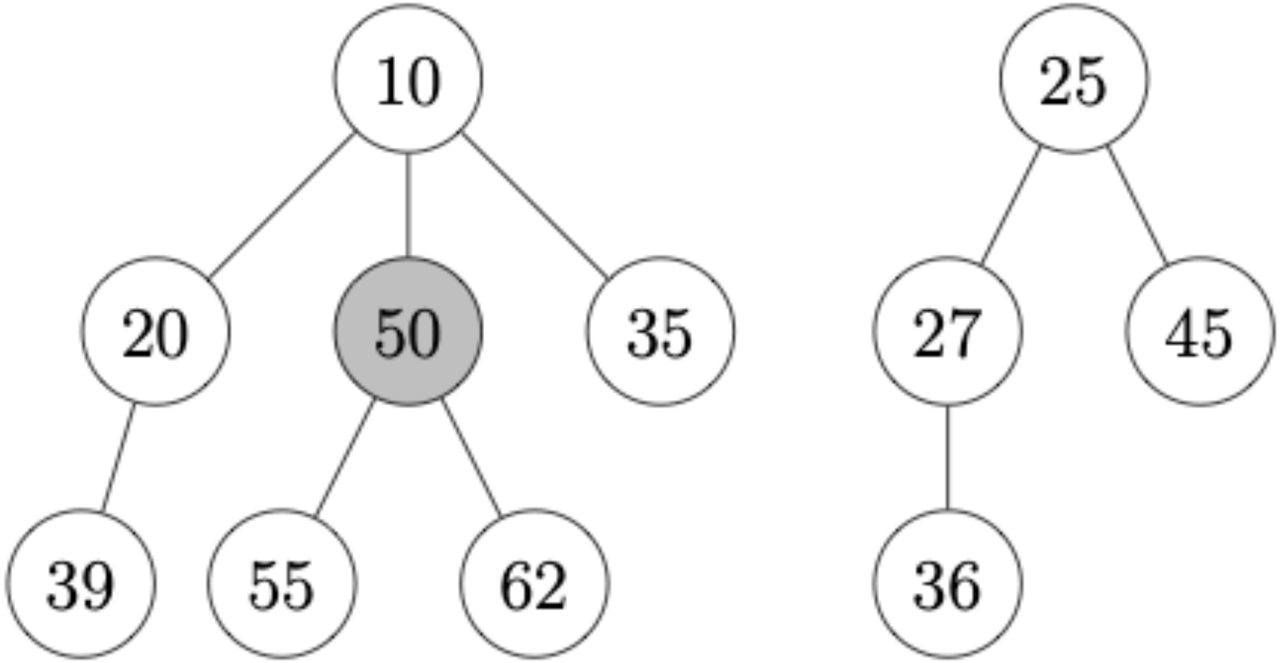
How can a root in a Fibonacci Heap be marked ?

Ans : When a marked node's parent = MIN, and is removed during Extract-Min

Ex : Insert 1, Insert 2, Insert 3, Insert 4, Delete 4, Extract-Min

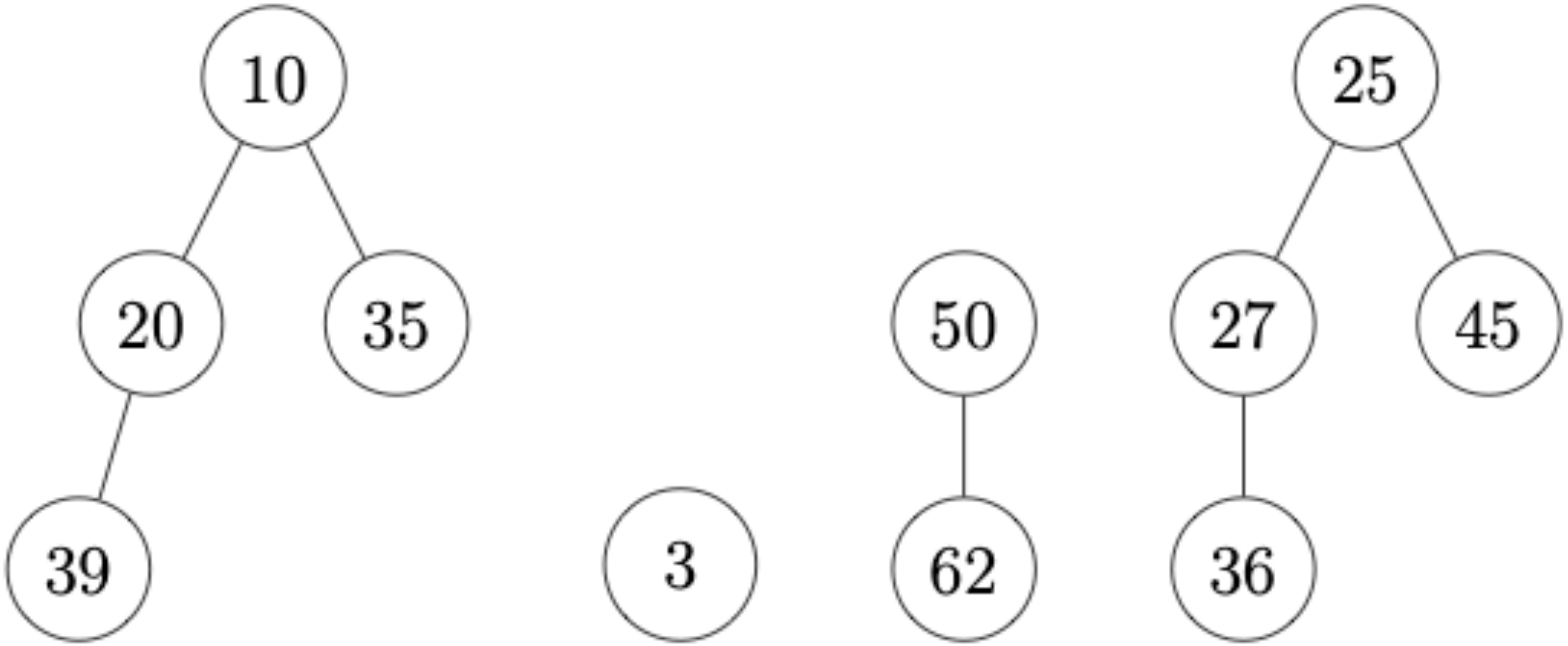
Q2

Consider the following Fibonacci Heap :



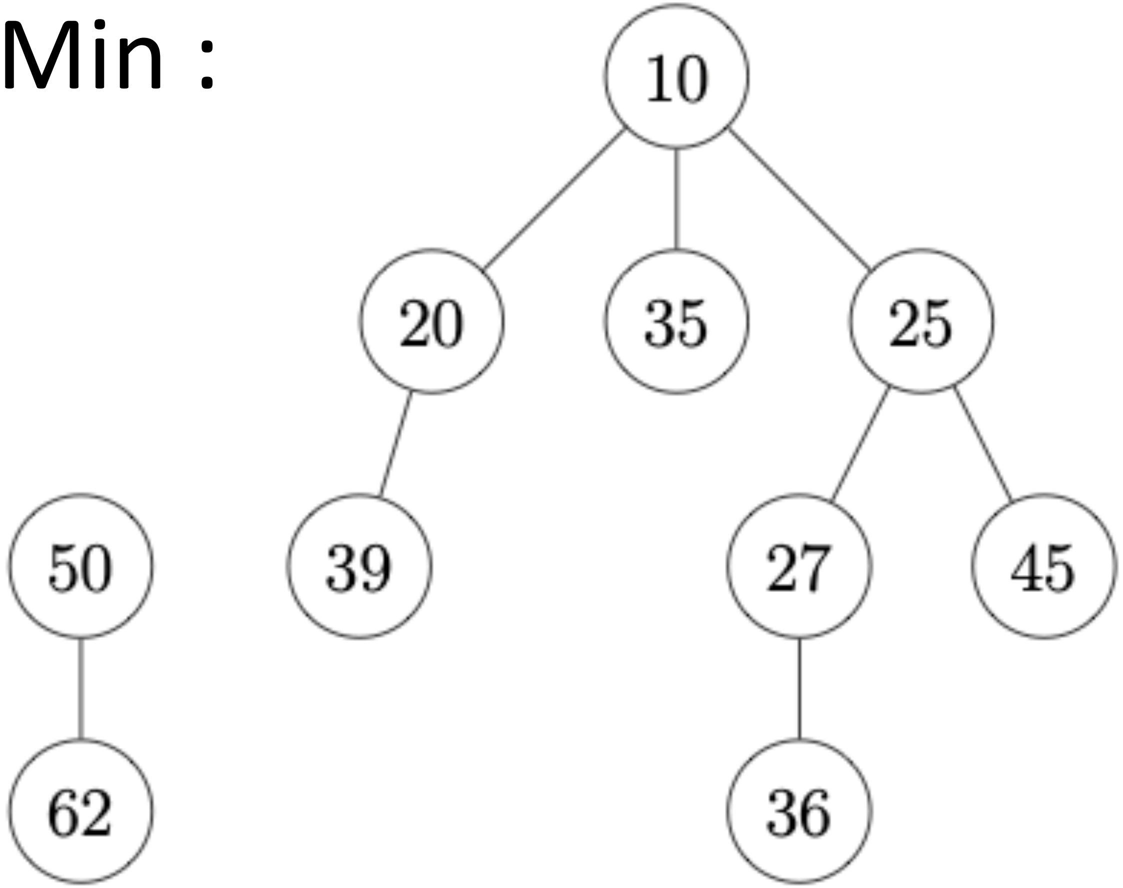
Q2(a) [solution]

After Decrease-Key 55 to 3 :



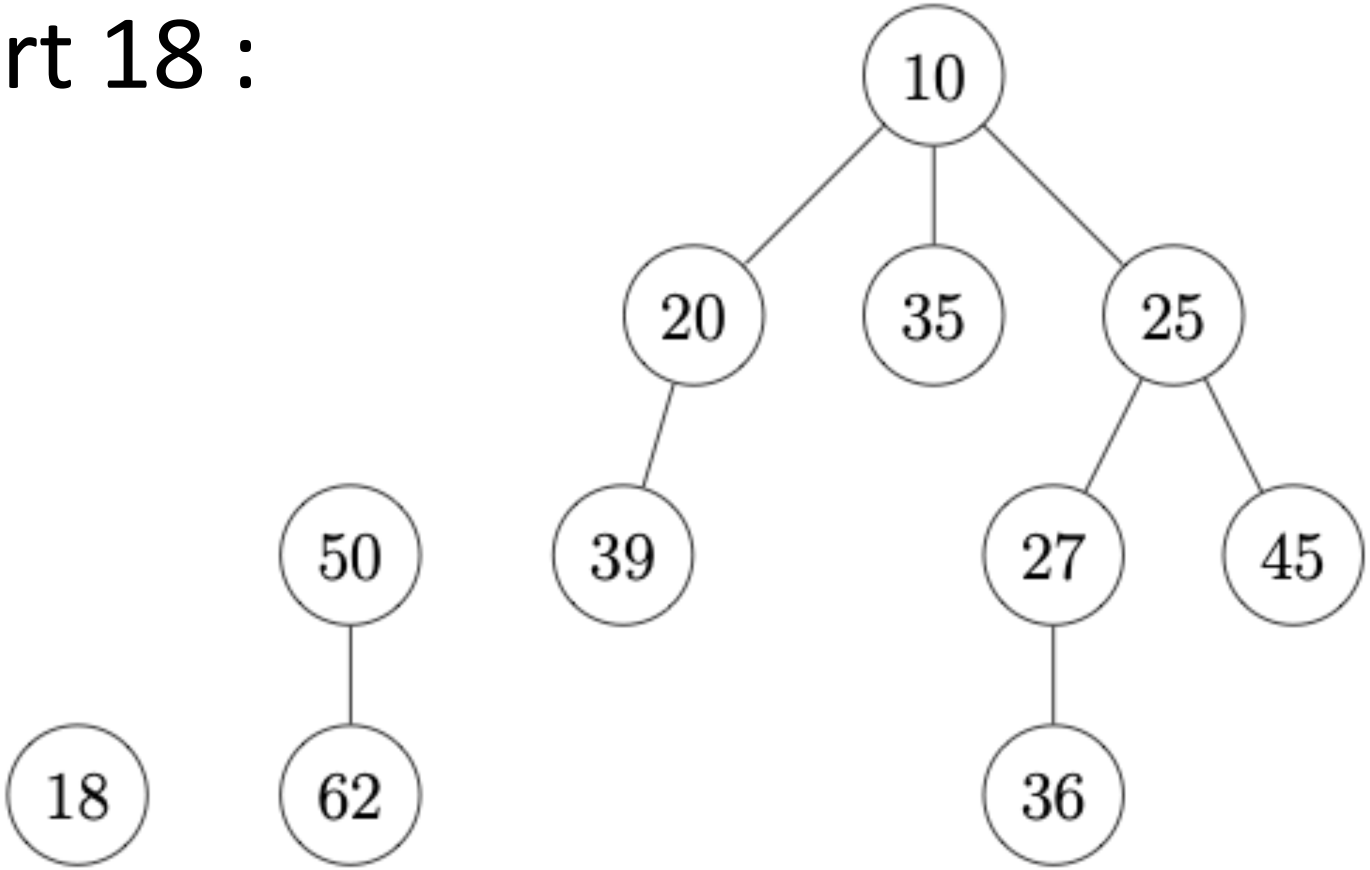
Q2(b) [solution]

Then, Extract-Min :



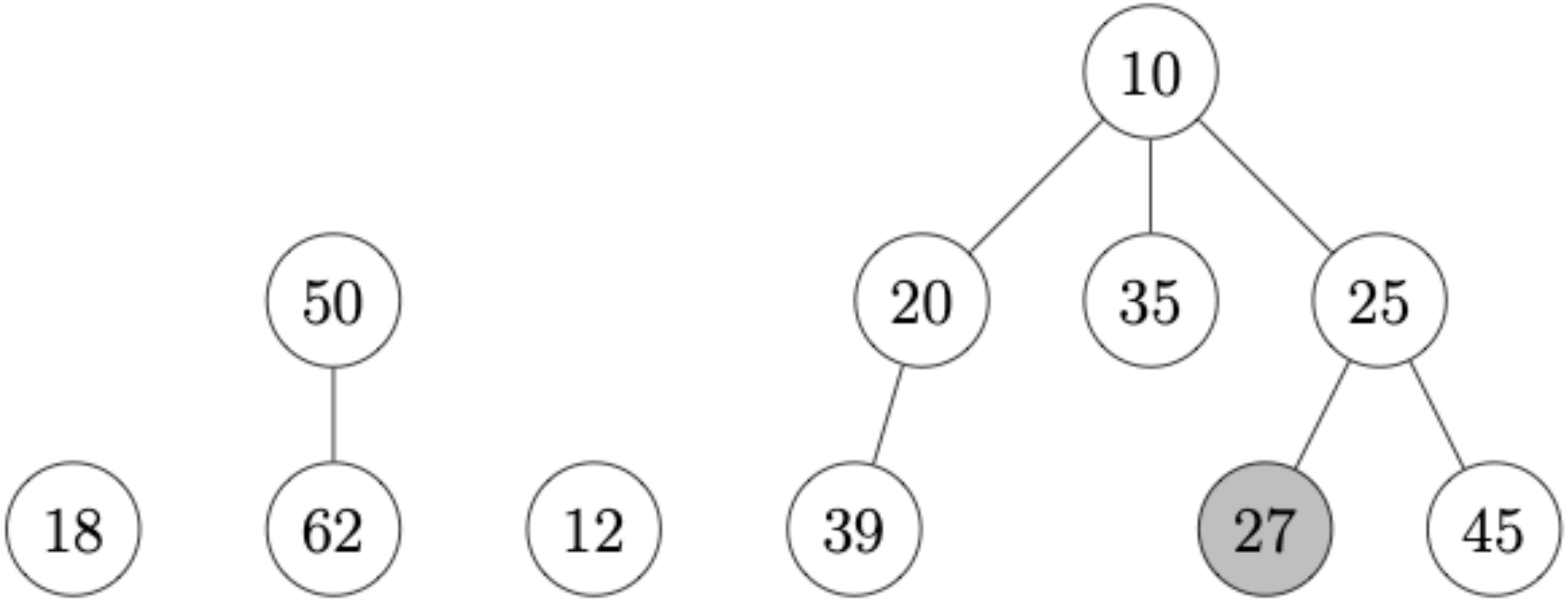
Q2(c) [solution]

Then, Insert 18 :



Q2(d) [solution]

Then, Decrease-Key 36 to 12 :



Q2(e) [solution (not unique)]

Then, Extract-Min :