Binomial Queues

CMSC420 0101

Spring 2019

Additional Priority Queue operations

- In PQs, we have talked about
  - Naïve implementation (list of lists, array of lists, single array that we splice in between whenever we add…)
  - Binary minheaps (solid implementation, with great cache locality)
    - $\mathcal{O}(\log_2 n)$ insertion, deleteMin()
    - $\mathcal{O}(1)$ getMin()

Additional Priority Queue operations

- In PQs, we have talked about
  - Naïve implementation (list of lists, array of lists, single array that we splice in between whenever we add…)
  - Binary minheaps (solid implementation, with great cache locality)
    - $\mathcal{O}(\log_2 n)$ insertion, deleteMin()
    - $\mathcal{O}(1)$ getMin()
- There are other crucial operations on PQs!
  - Merging PQs (binomial queues)
  - Decreasing a key (fibonacci queues)

Merging heaps

- Two ways to merge heap A with heap B (assume same size n):

Merging heaps

- Two ways to merge heap A with heap B (assume same size n):
  1. Call `deleteMin()` on A n times, and for every one of those times, insert into B. Complexity: $\mathcal{O}(n \cdot (\log_2 n)^2)$. Can drop to $\mathcal{O}(n \cdot \log_2 n)$ if instead of calling `deleteMin()` n times, we have a way of traversing the heap (BFS, DFS, Iterator…).

Merging heaps

- Two ways to merge heap A with heap B (assume same size n):
  1. Call `deleteMin()` on A n times, and for every one of those times, insert into B. Complexity: $\mathcal{O}(n \cdot (\log_2 n)^2)$. Can drop to $\mathcal{O}(n \cdot \log_2 n)$ if instead of calling `deleteMin()` n times, we have a way of traversing the heap (BFS, DFS, Iterator…).
  2. *(Only in array implementation):* Merge the two arrays into a new one, of size $2n$ ($\mathcal{O}(n)$) and then call `heapify()` on the new array $\left(\mathcal{O}(\log_2(2n))\right)$.
     - So, in total, the copying over "wins" and we have $\mathcal{O}(n)$ for merging two equi-sized heaps.

Merging heaps

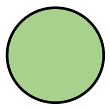
- Two ways to merge heap A with heap B (assume same size n):
  1. Call `deleteMin()` on A n times, and for every one of those times, insert into B. Complexity: $\mathcal{O}(n \cdot (\log_2 n)^2)$. Can drop to $\mathcal{O}(n \cdot \log_2 n)$ if instead of calling `deleteMin()` n times, we have a way of traversing the heap (BFS, DFS, Iterator…).
  2. *(Only in array implementation):* Merge the two arrays into a new one, of size $2n$ ($\mathcal{O}(n)$) and then call `heapify()` on the new array $\left(\mathcal{O}(\log_2(2n))\right)$.
     - So, in total, the copying over "wins" and we have $\mathcal{O}(n)$ for merging two equi-sized heaps.
- Binomial heaps can be merged in $\mathcal{O}(\log_2 n)$.

Binomial queue: basic structure

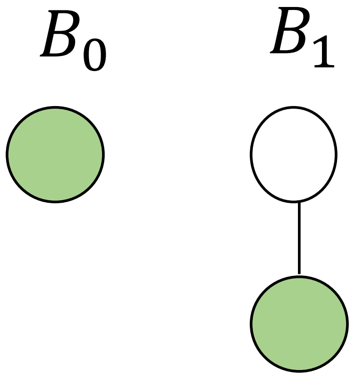
- Instead of a **single** binary tree, a binomial queue is a **forest** of k-ary trees B_i, themselves called *binomial trees*!
- Recursive definition of a binomial tree:
  - B_0 is a tree consisting of a single node.
  - B_k, for $k \geq 1$ is made up by a root node that contains k links to trees $B_0, B_1, \ldots, B_{k-1}$.

Binomial queue example

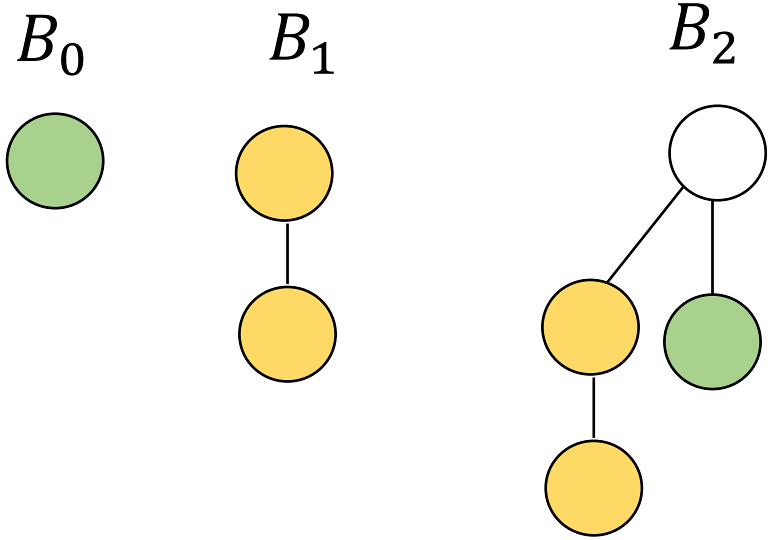
B_0



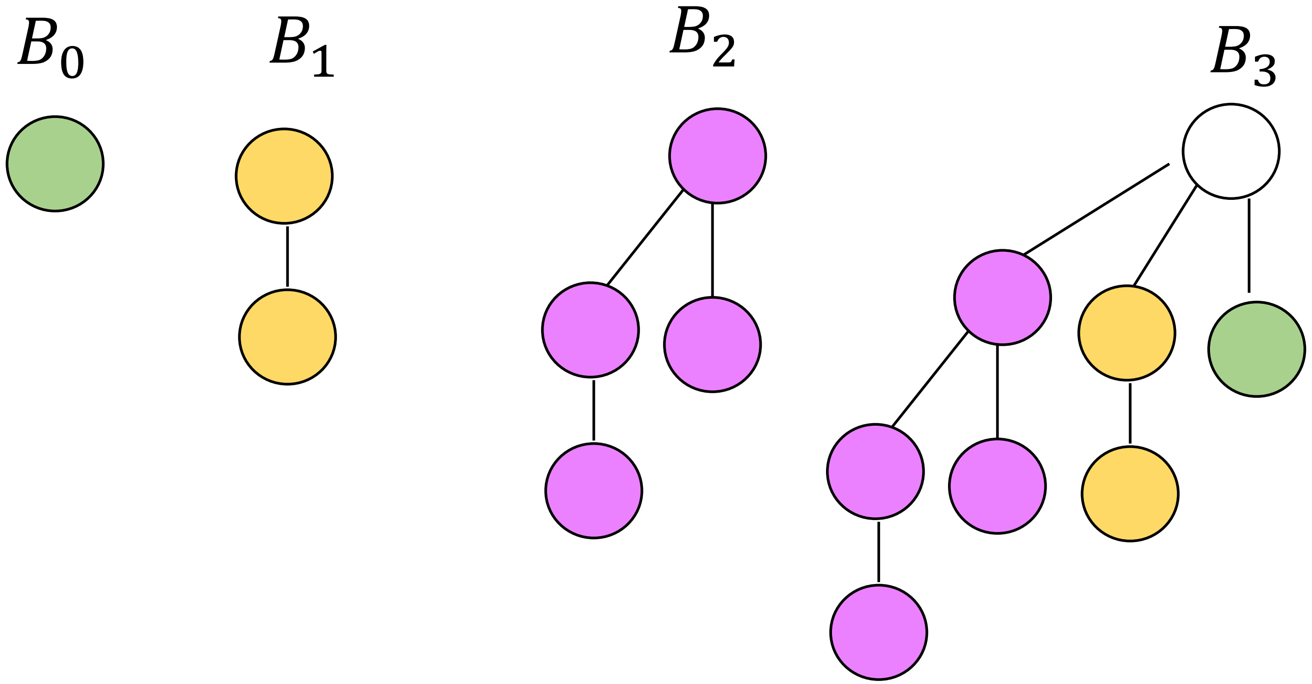
Binomial queue example



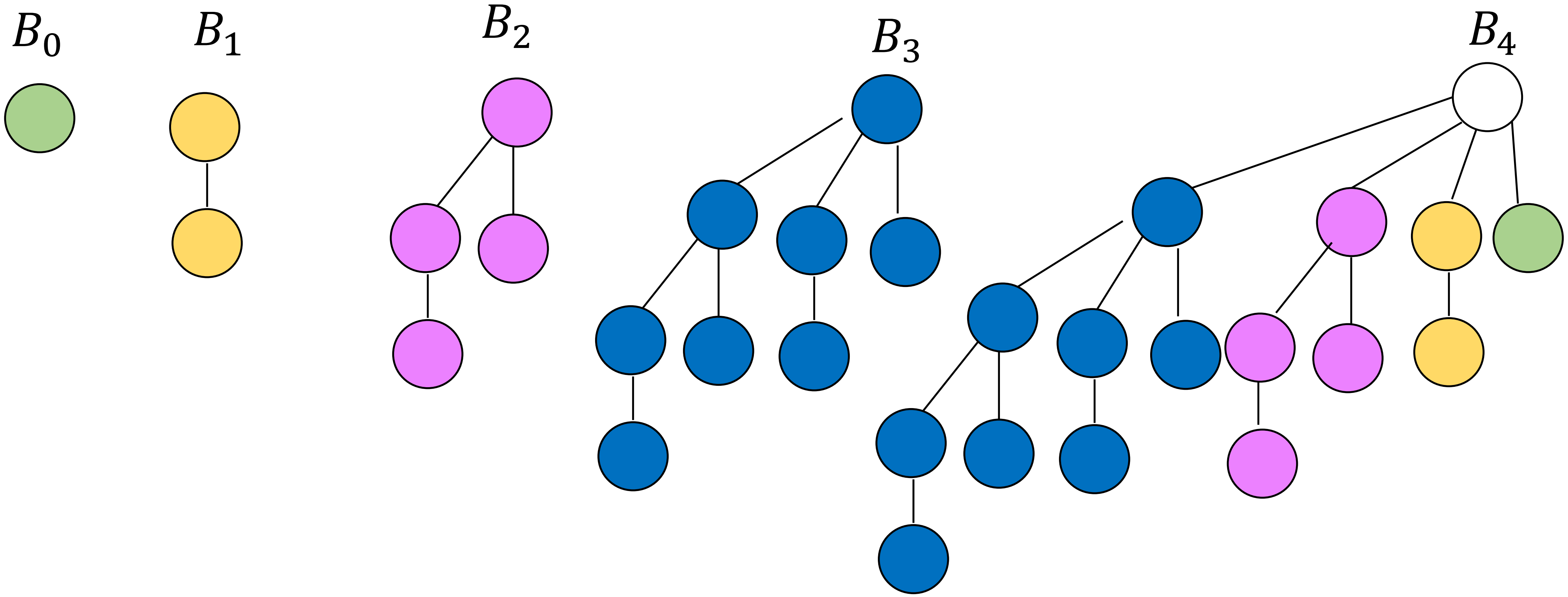
Binomial queue example



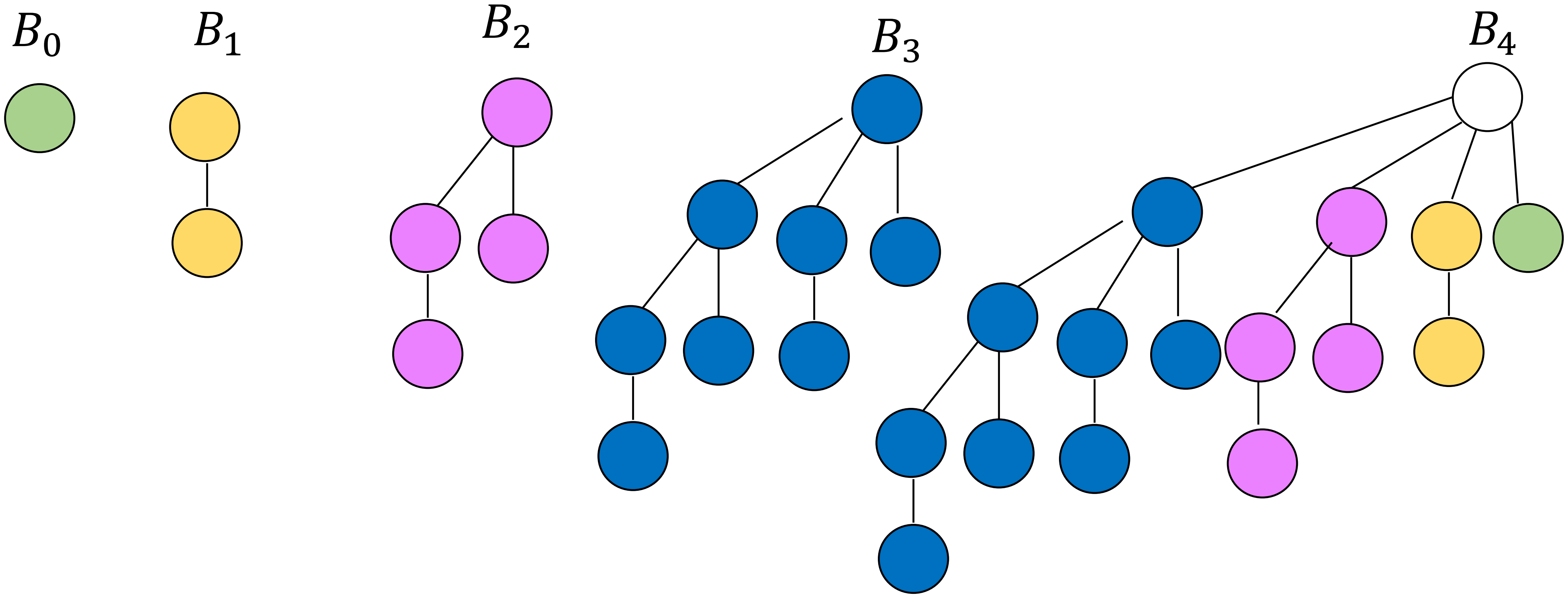
Binomial queue example



Binomial queue example

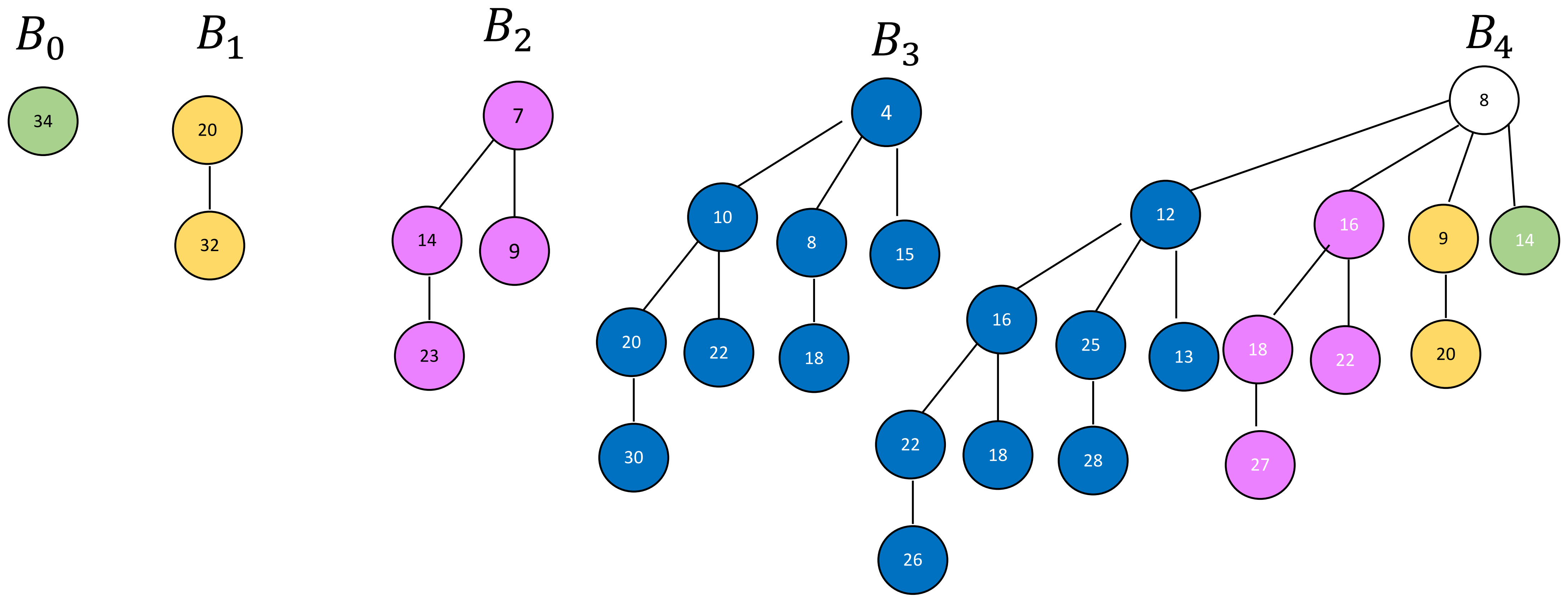


Binomial queue example



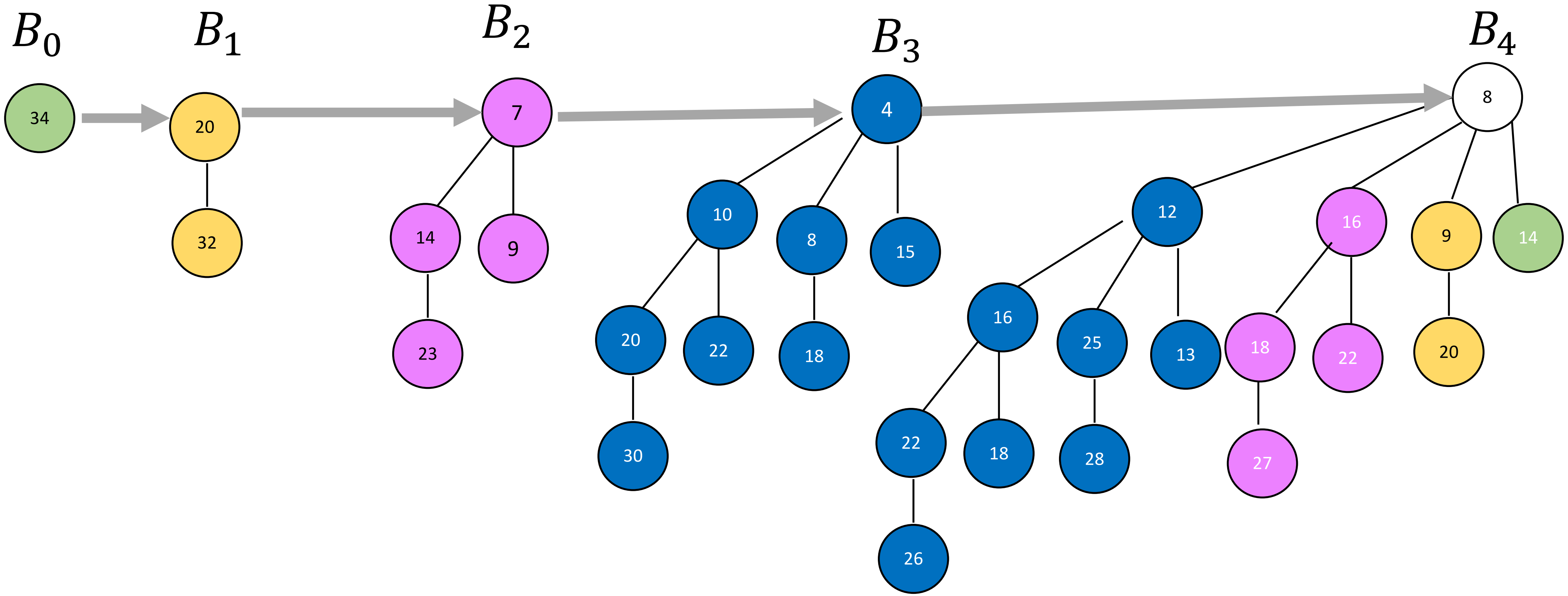
- All those heap-ordered trees together (a forest) make up the binomial queue.

Binomial queue example



- All those ***heap-ordered*** trees together (a forest) make up the binomial queue.

Binomial queue example



- All those ***heap-ordered*** trees together (a forest) make up the binomial queue.
- The roots are connected to each other through a list.

Some results

1. The tree B_k has 2^k nodes.
   - Starting with 1 node, we double the amount of nodes every time.

Some results

1. The tree B_k has 2^k nodes.
   - Starting with 1 node, we double the amount of nodes every time.
2. The tree B_k has k levels (so a height of $k - 1$).
   - Straightforward consequence of the fact that T_{k+1} adds a level to T_k, by construction

Some results

1. The tree B_k has 2^k nodes.
   - Starting with 1 node, we double the amount of nodes every time.
2. The tree B_k has k levels (so a height of $k - 1$).
   - Straightforward consequence of the fact that T_{k+1} adds a level to T_k, by construction
3. The i^{th} level of the tree B_k, where $i \in \{0, 1, 2, \dots\}$, has $\binom{k}{i}$ nodes!

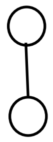
Binary representation of a binomial queue

- A binomial queue is a set of binomial trees with some maximum degree K for its trees.
- We define a length - K binary representation $B = B_0 B_1 \dots B_k$ of our queue using the following rule:

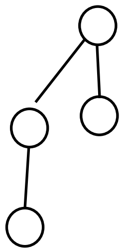
$$B_i = \begin{cases} 1, & \quad if\ T_i\ is\ part\ of\ the\ queue \\ 0, & \quad otherwise \end{cases}$$

Binary representation example

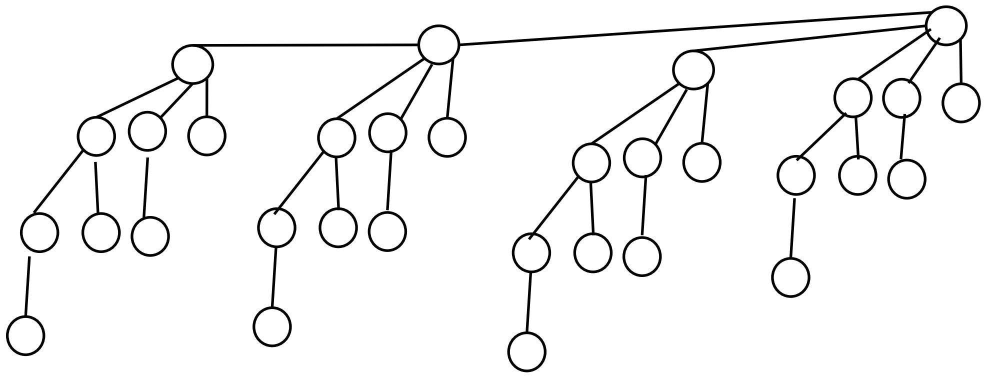
B_1



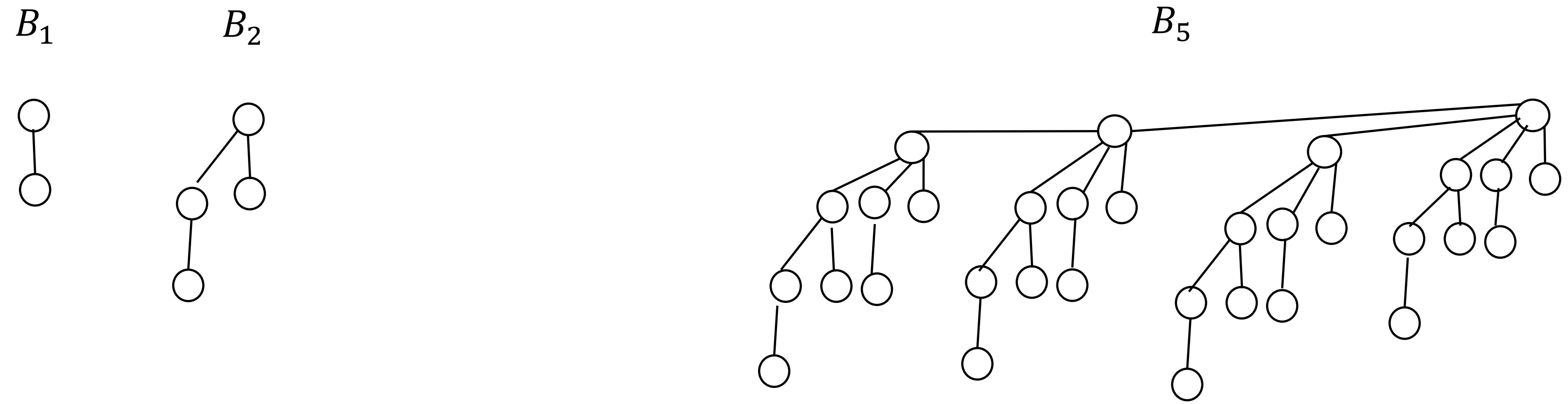
B_2



B_5

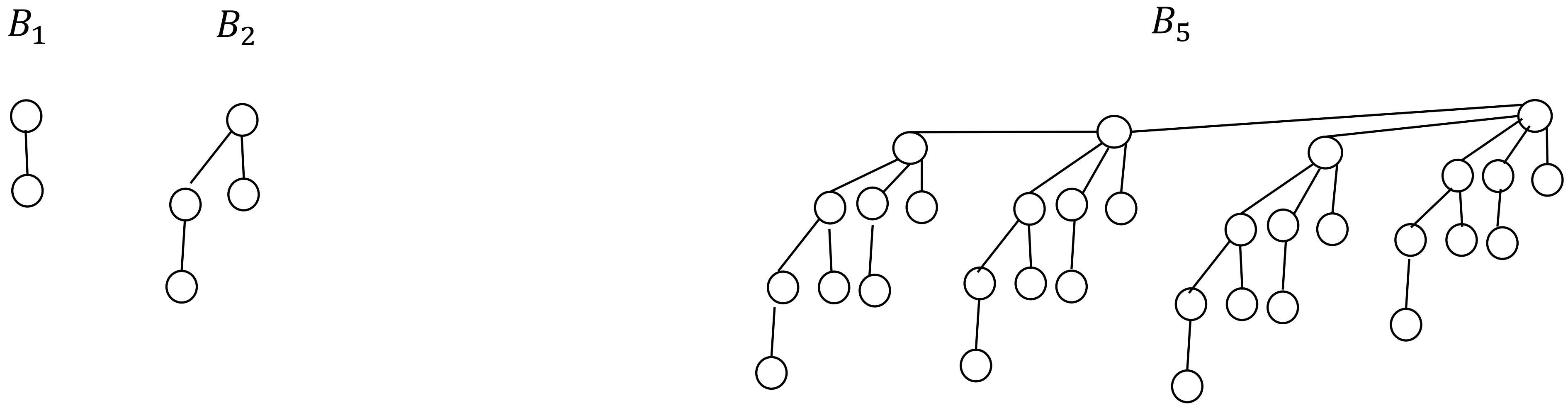


Binary representation example



k	5	4	3	2	1	0
B_k present?	T	F	F	T	T	F
$bits$	1	0	0	1	1	0

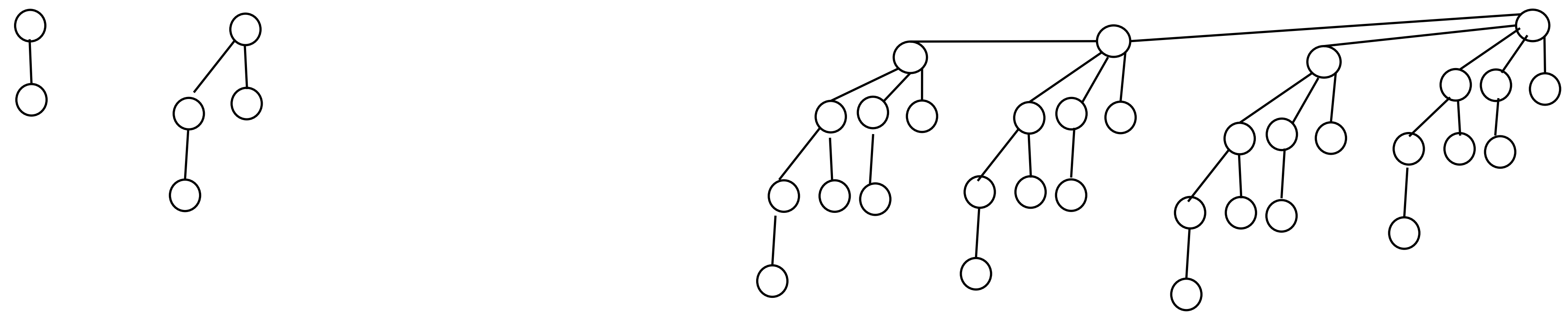
Binary representation example



k	5	4	3	2	1	0
B_k present?	T	F	F	T	T	F
$bits$	**1**	**0**	**0**	**1**	**1**	**0**

Binary representation example

B_1 B_2 B_5



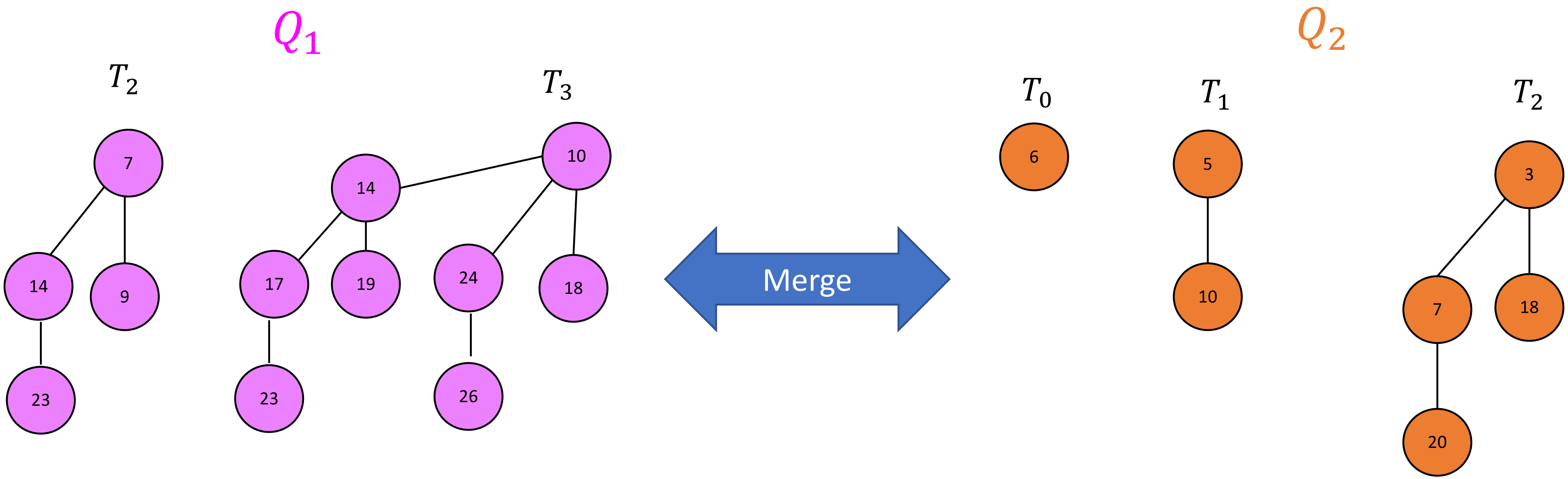
k	5	4	3	2	1	0
B_k present?	T	F	F	T	T	F
$bits$	**1**	**0**	**0**	**1**	**1**	**0**

Another result: A binomial queue of size n has $[\log_2 n]$ trees.

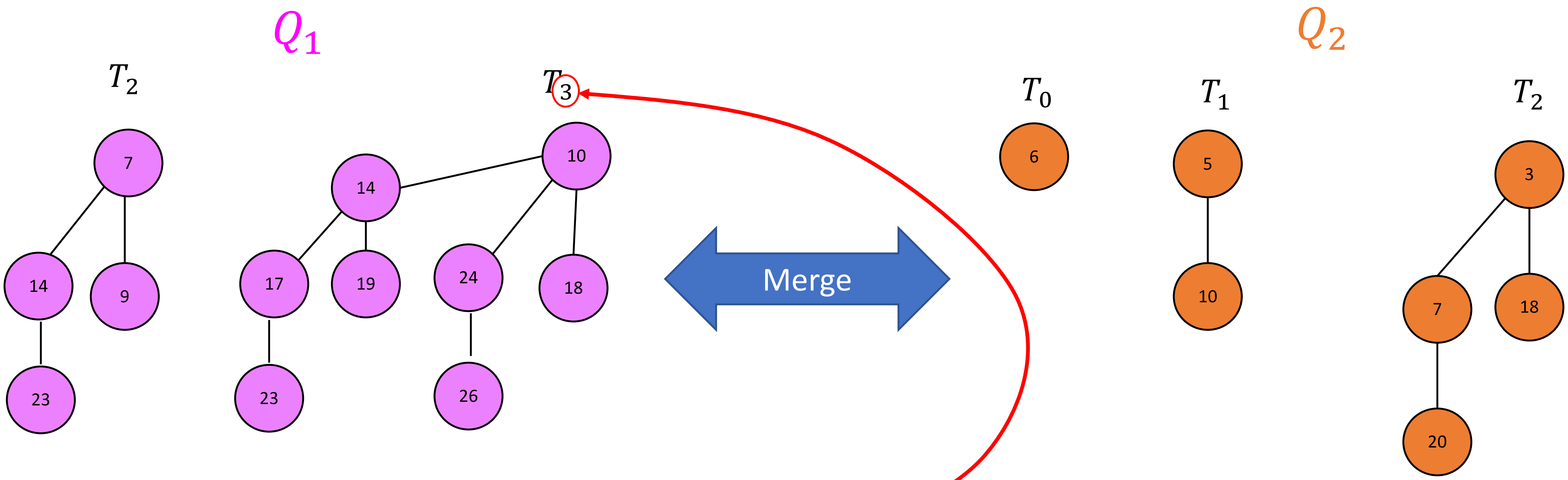
Merging binomial queues

- To merge two binomial queues, we simply **add the two binary representations together**!
- Whenever we encounter an addition of a 1 with a 1, we have a carry bit.
  - In Bin Queue terms, this means that we construct a tree of order B_{k+1} by comparing the roots of two B_ks (those made up the ‘1’s in the addition) and making the smaller one the root of the new tree.
  - So combining two binomial trees is ***constant time***: one comparison and one reference assignment, irrespective of n.

Merging example

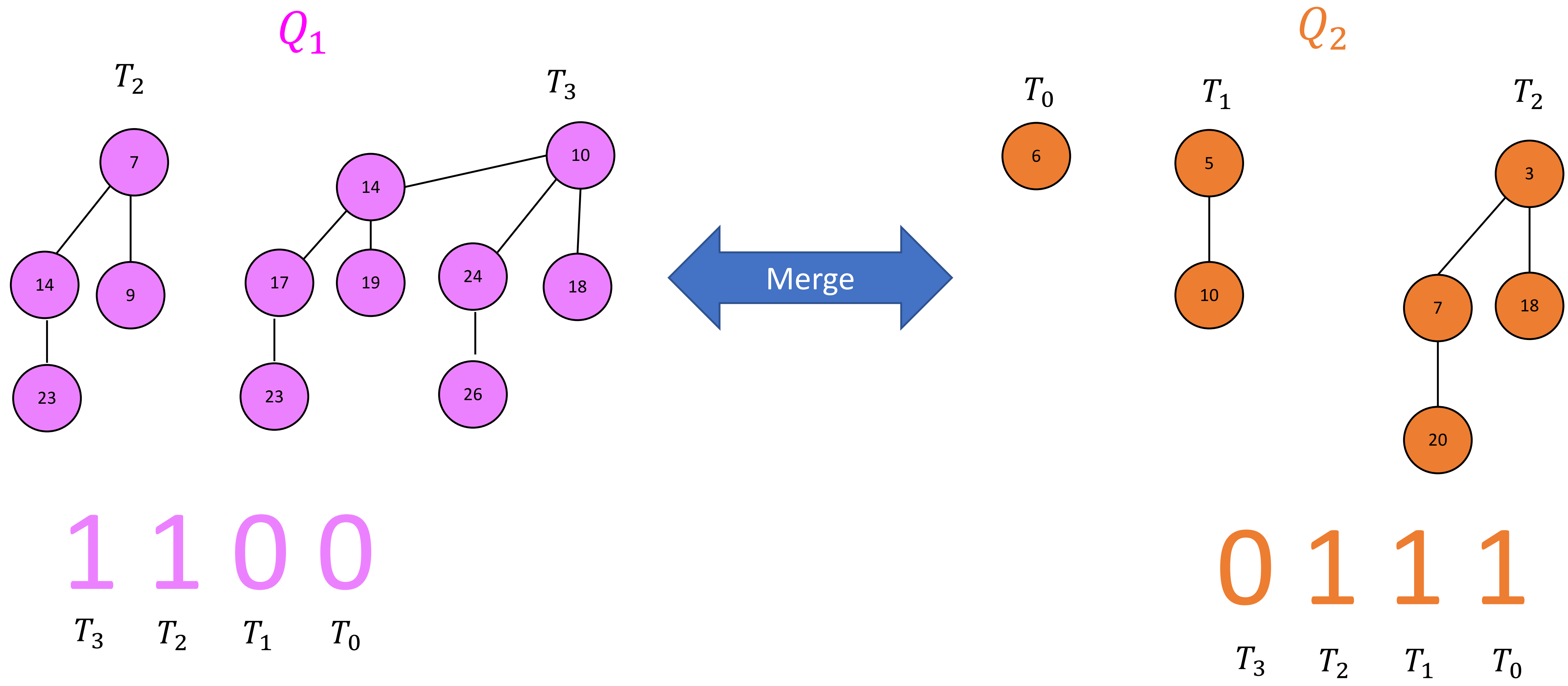


Merging example

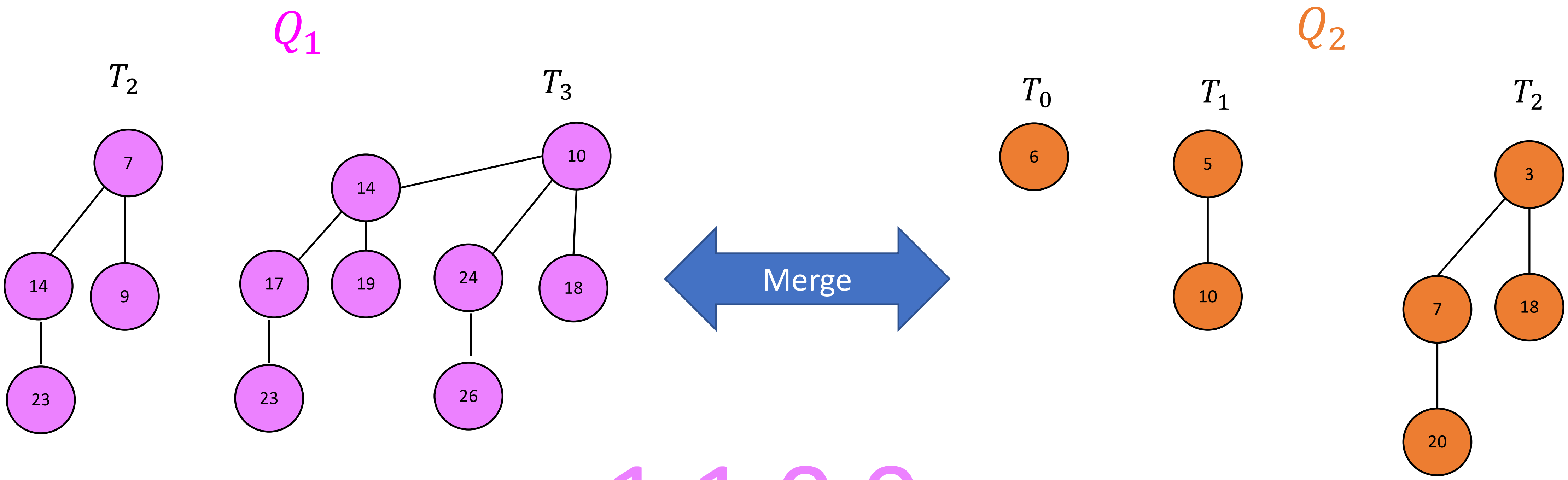


Since the highest rank tree is 3, we have two 4-bit representations to add.

Merging example

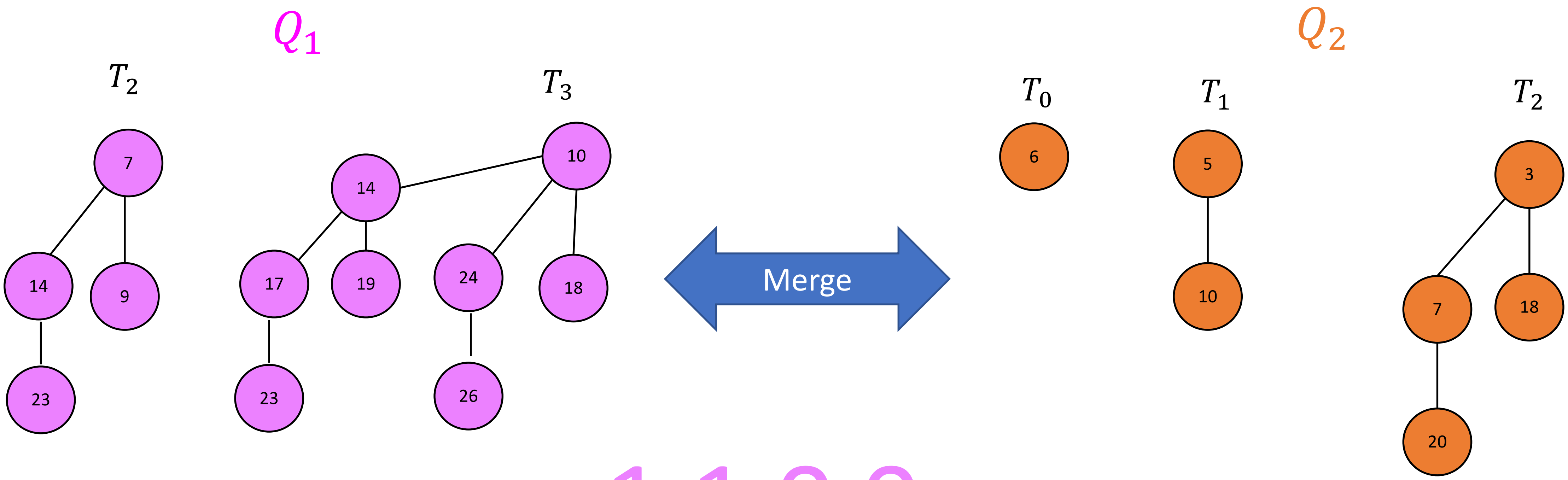


Merging example



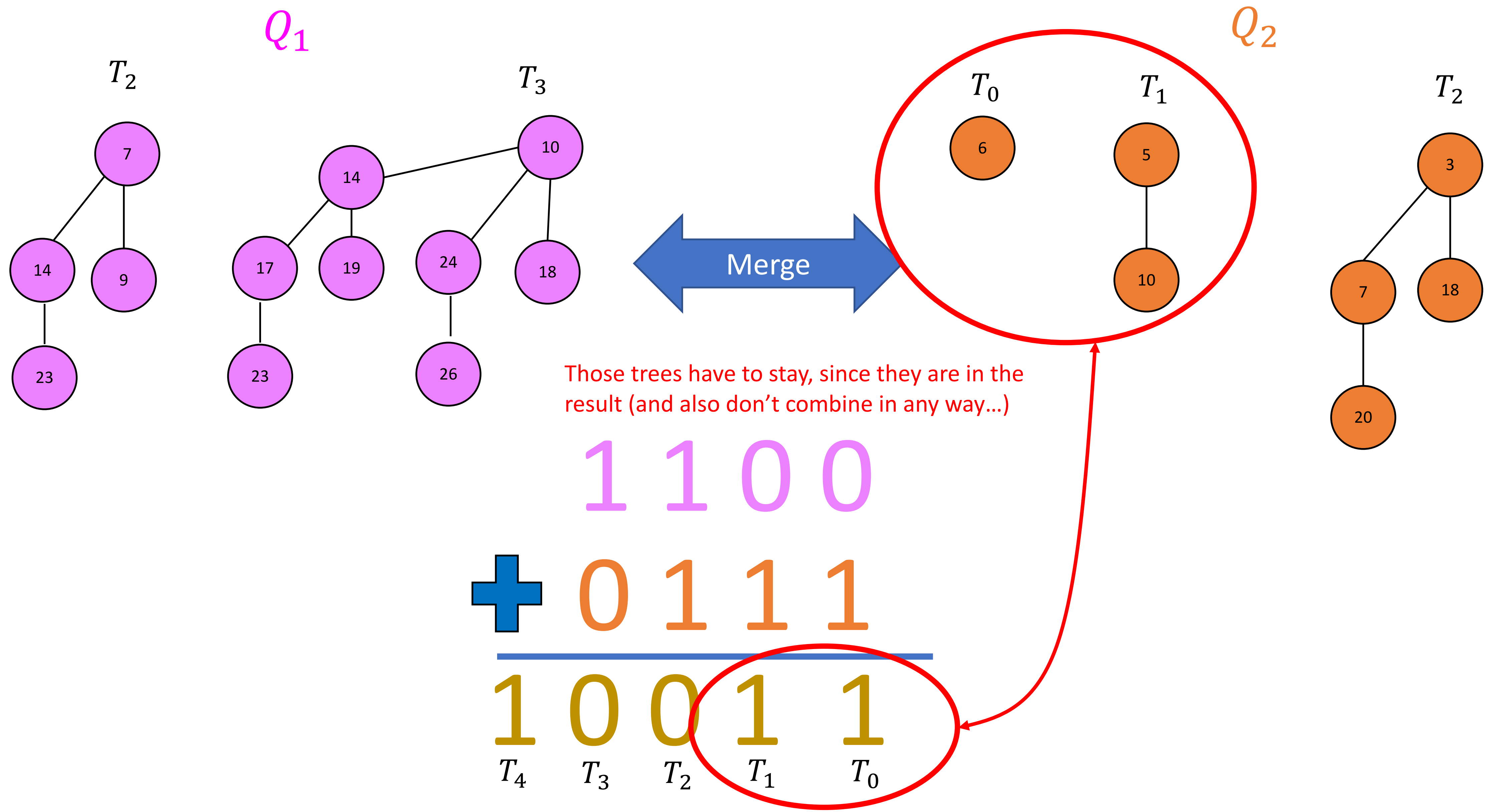
$$
\begin{array}{r}
1\,1\,0\,0 \\
+\ 0\,1\,1\,1 \\
\hline
1\,0\,0\,1\,1
\end{array}
$$

Merging example

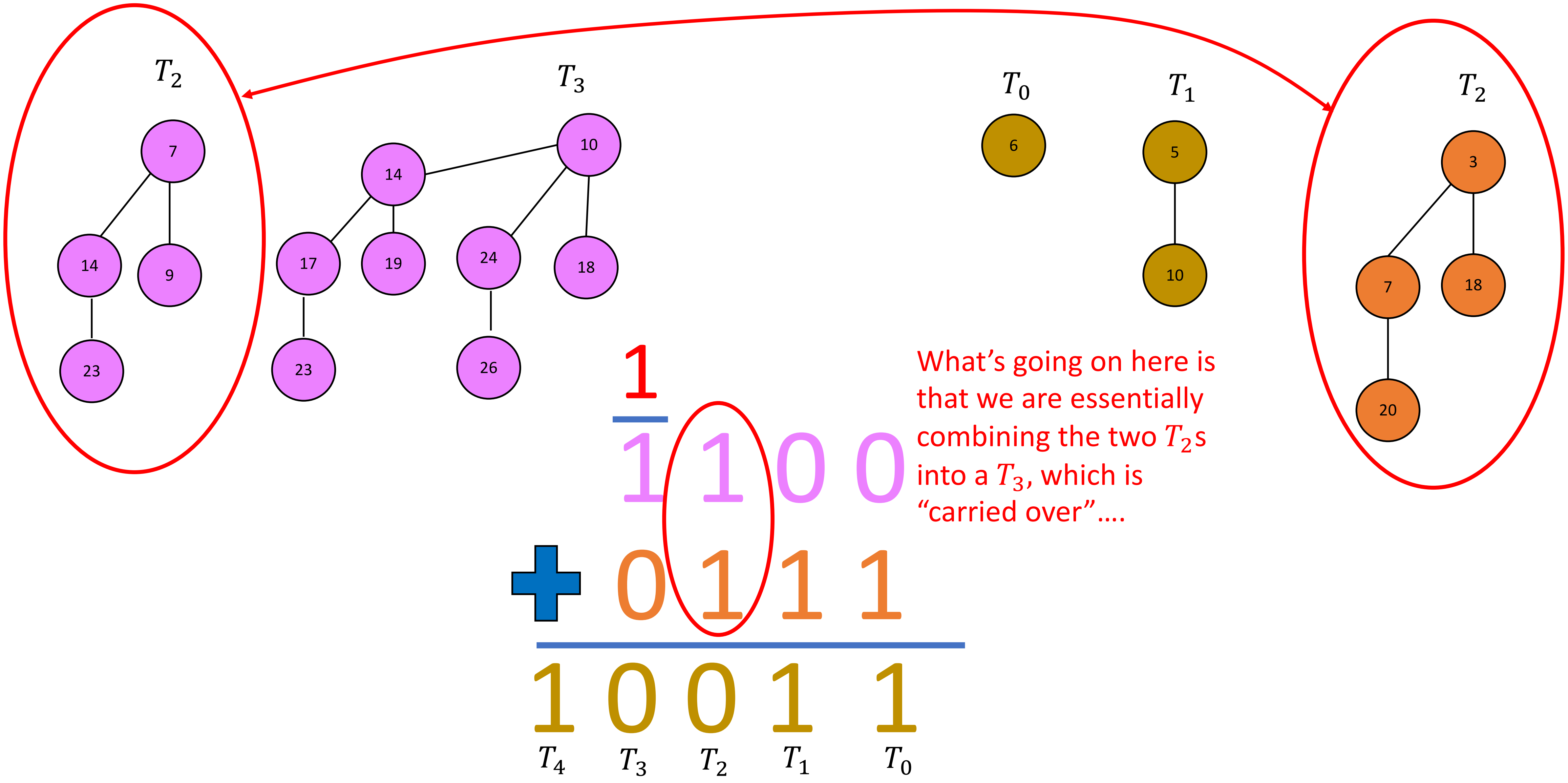


$$
\begin{array}{rccccc}
 & & 1 & 1 & 0 & 0 \\
+ & & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 1 & 1 \\
 & T_4 & T_3 & T_2 & T_1 & T_0
\end{array}
$$

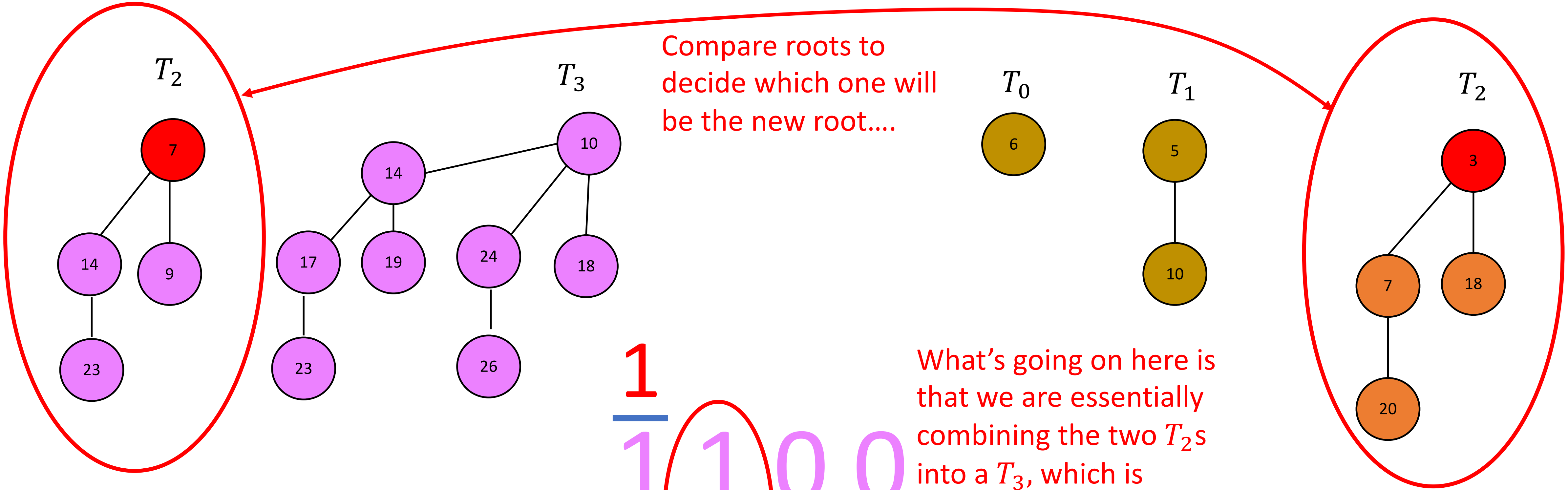
Merging example



Merging example



Merging example

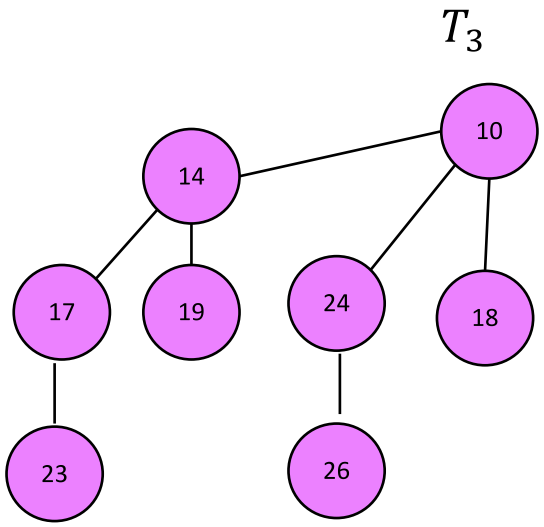


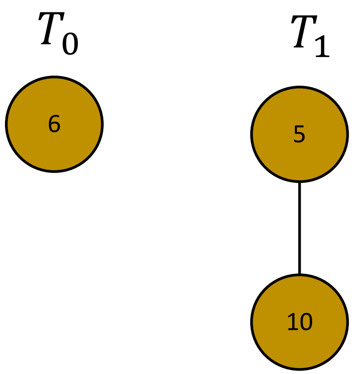
Compare roots to decide which one will be the new root….

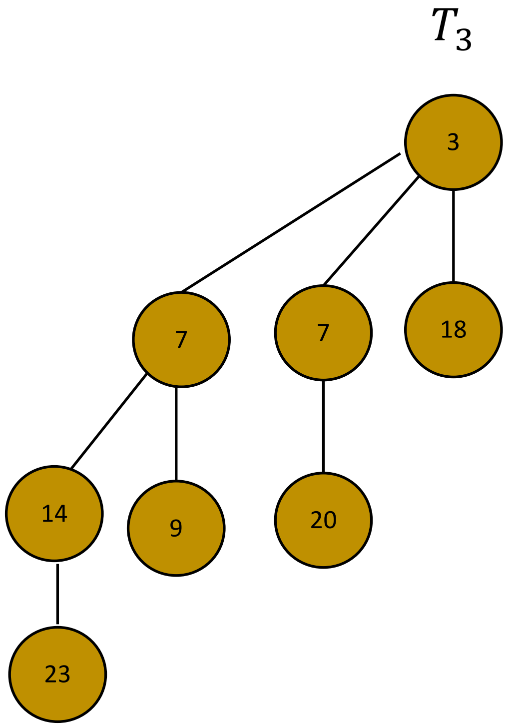
$$
\begin{array}{cccccc}
 & & 1 & & & \\
 & & 1 & 1 & 0 & 0 \\
+ & & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 1 & 1 \\
 & T_4 & T_3 & T_2 & T_1 & T_0
\end{array}
$$

What's going on here is that we are essentially combining the two T_2s into a T_3, which is "carried over"….

Merging example

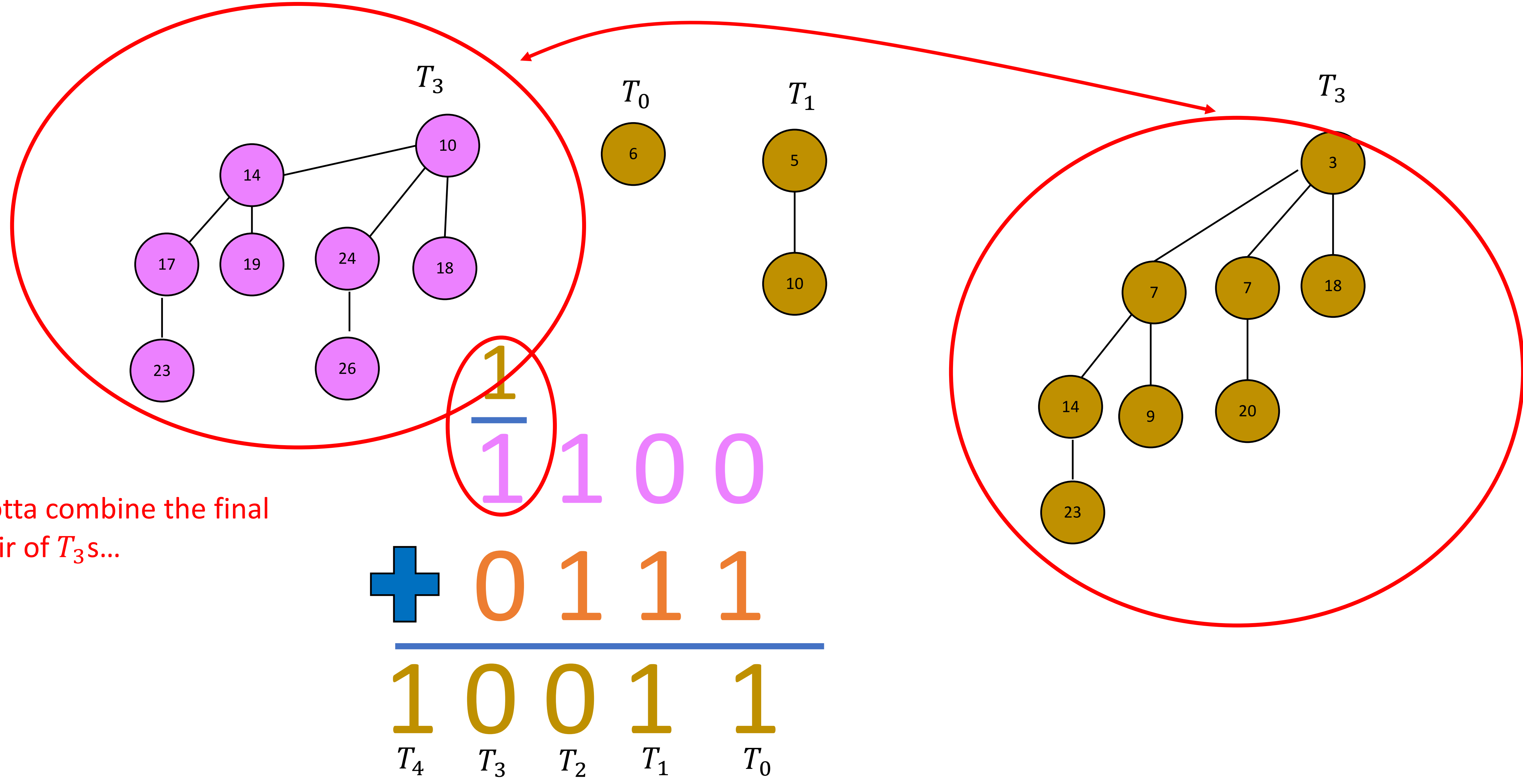






$$
\begin{array}{rccccc}
 & & 1 & & & \\
\hline
 & & 1 & 1 & 0 & 0 \\
+ & & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 1 & 1 \\
 & T_4 & T_3 & T_2 & T_1 & T_0
\end{array}
$$

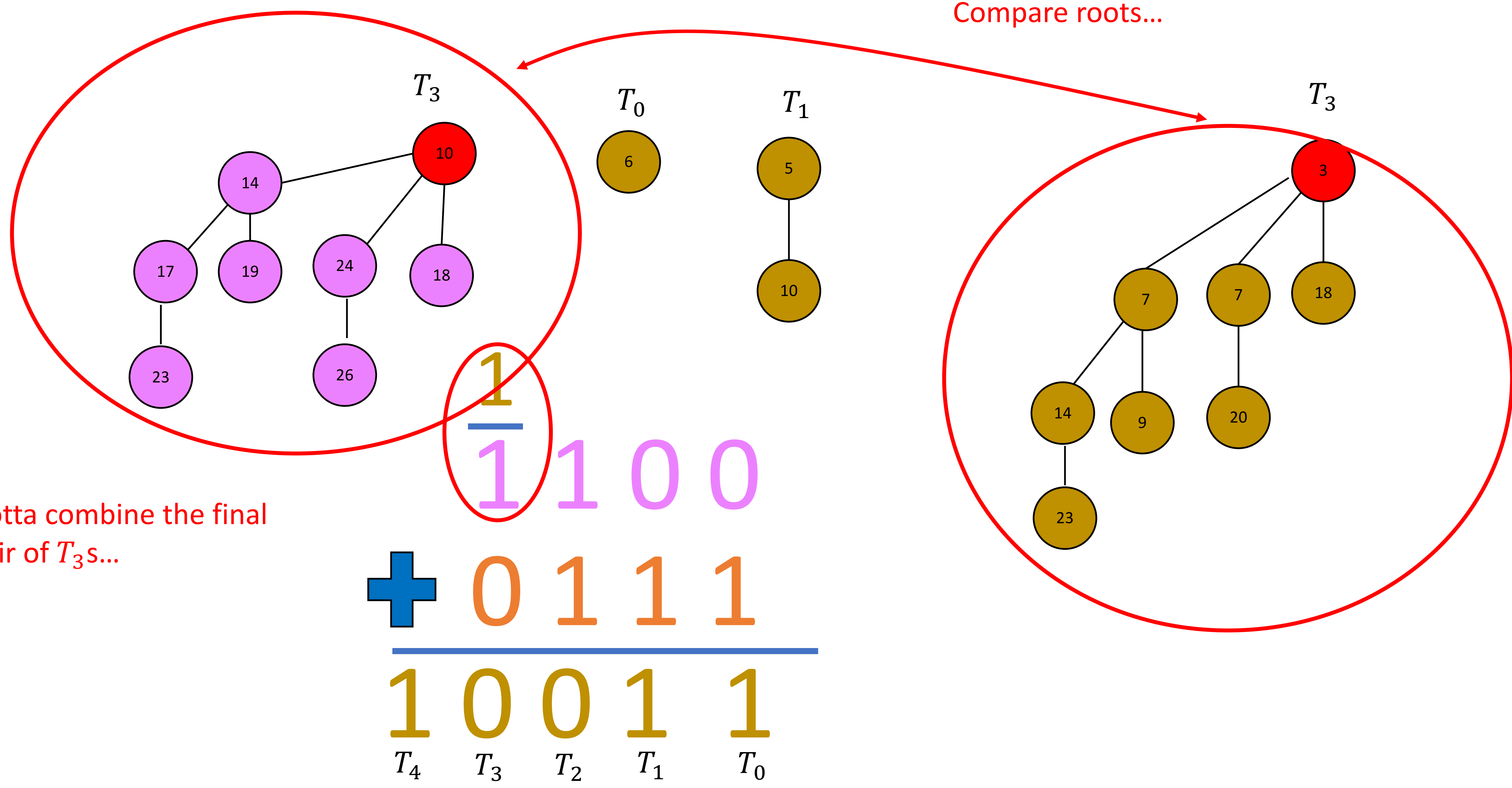
Merging example



Gotta combine the final pair of T_3s...

Merging example

Compare roots…



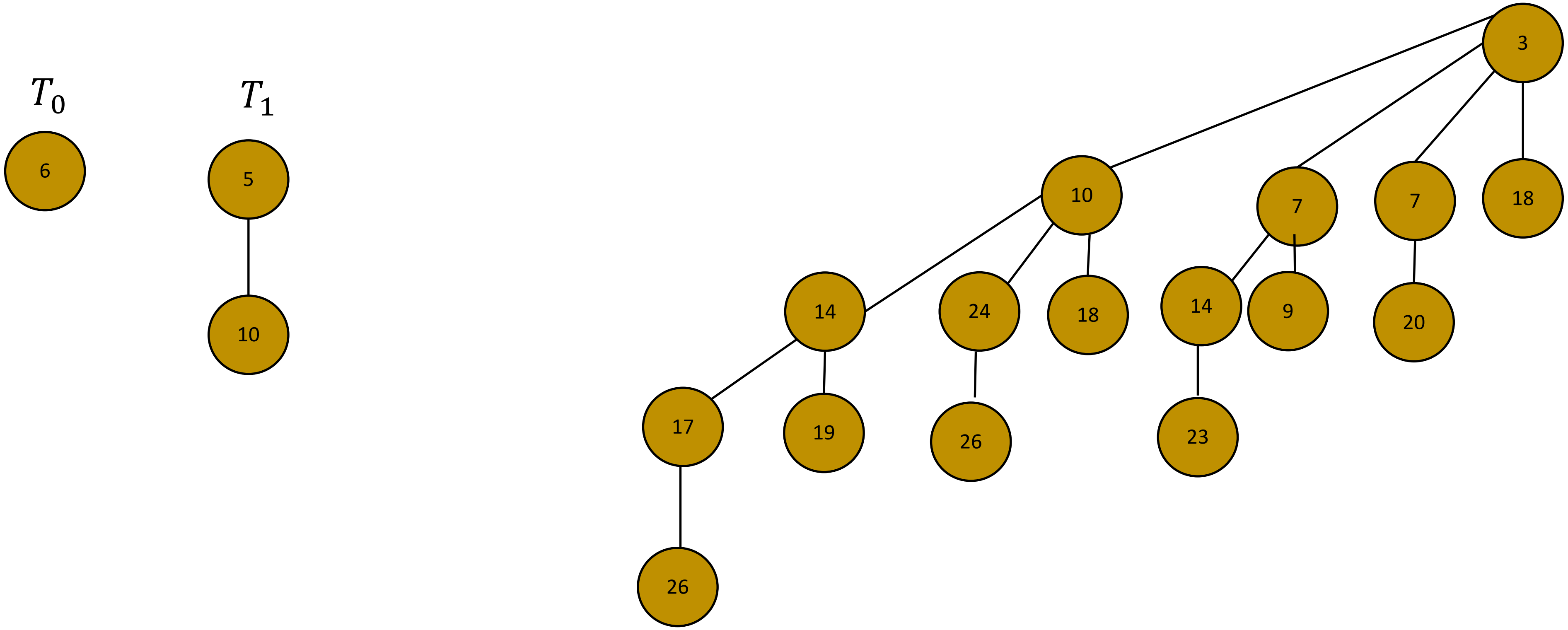
Gotta combine the final pair of T_3s…

Merging example

And done!

T_4

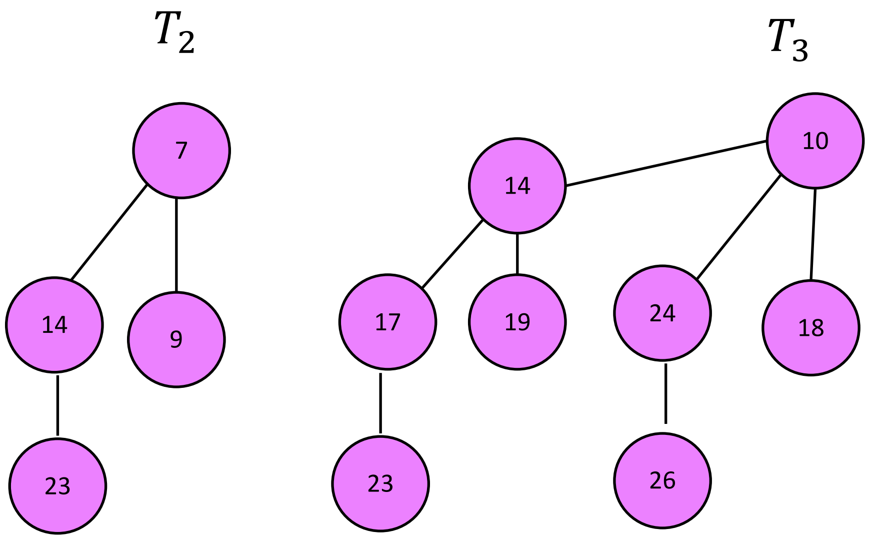
T_0 T_1

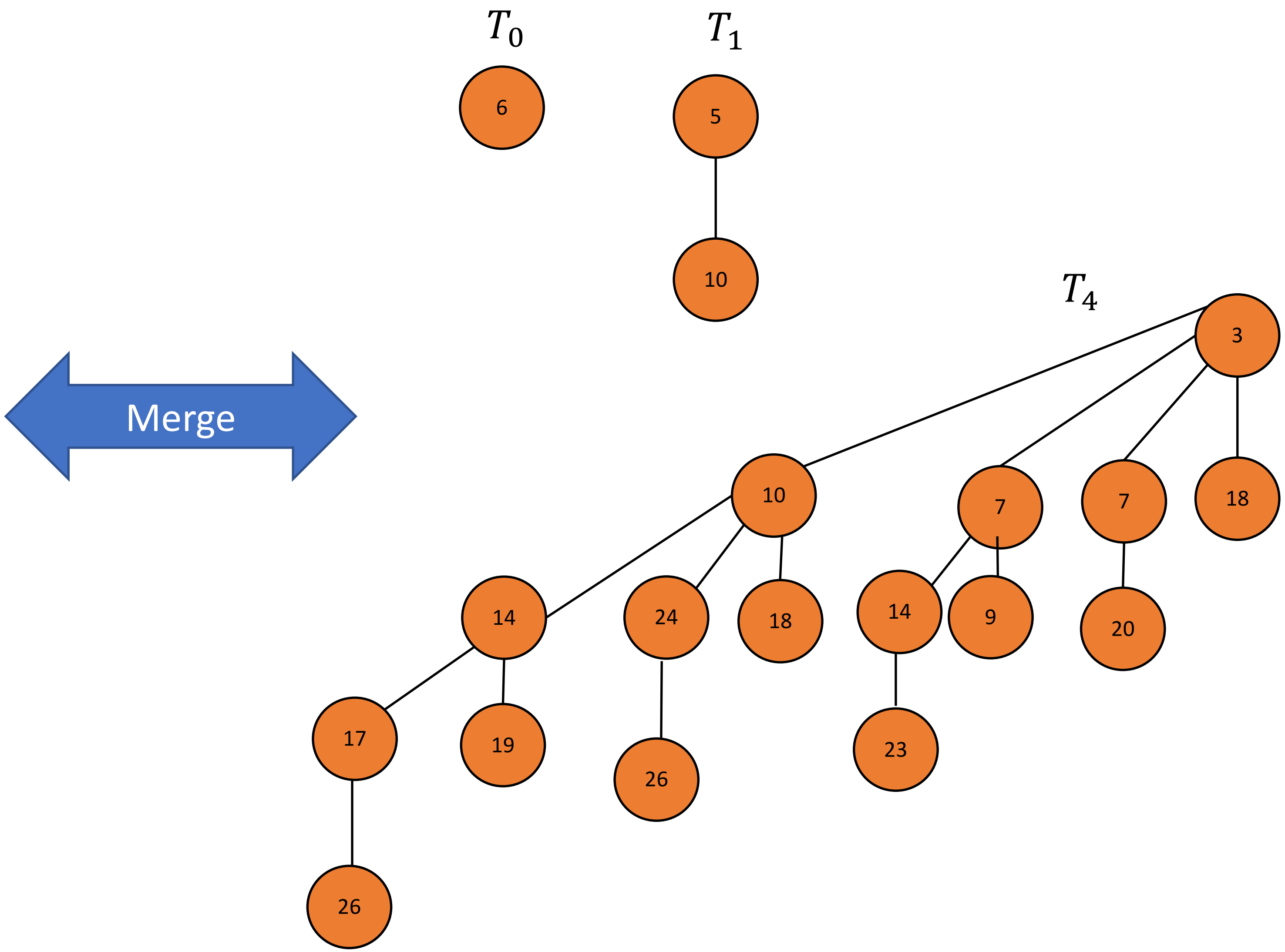


1 0 0 1 1

T_4 T_3 T_2 T_1 T_0

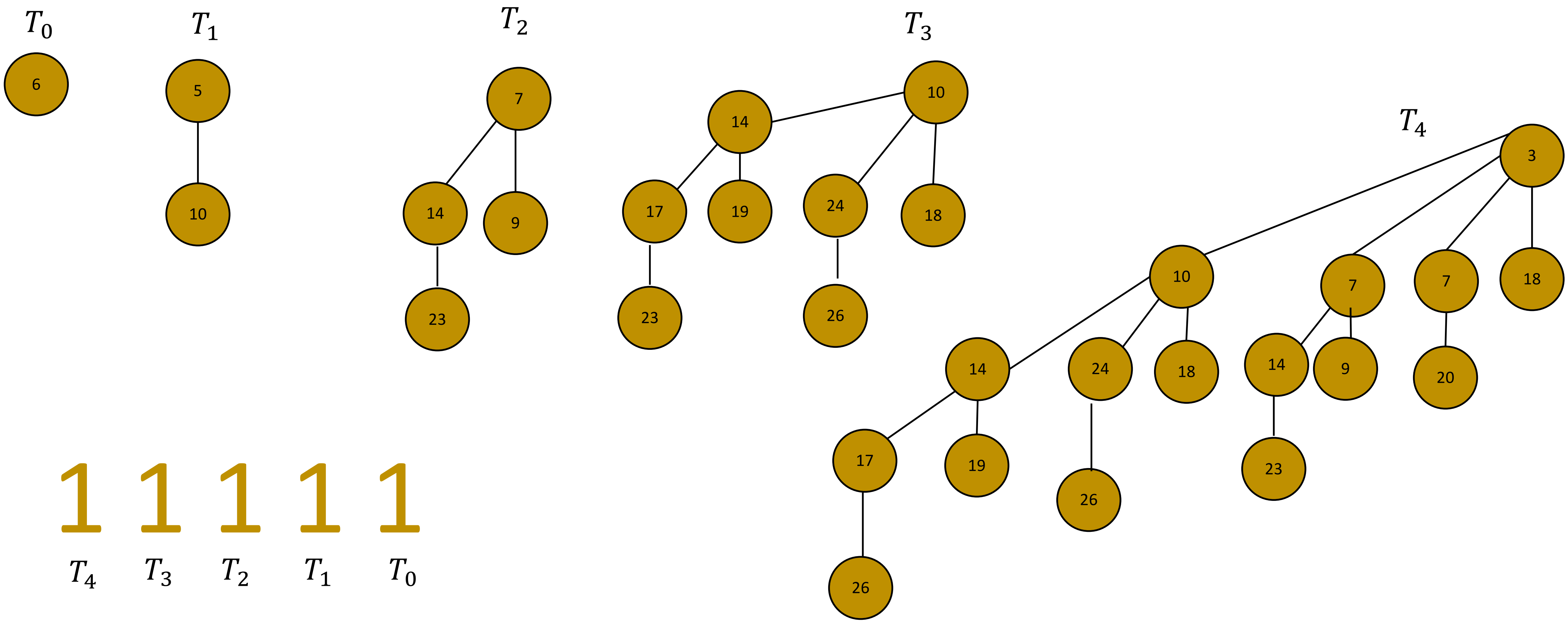
Best scenario: no combines (carries)



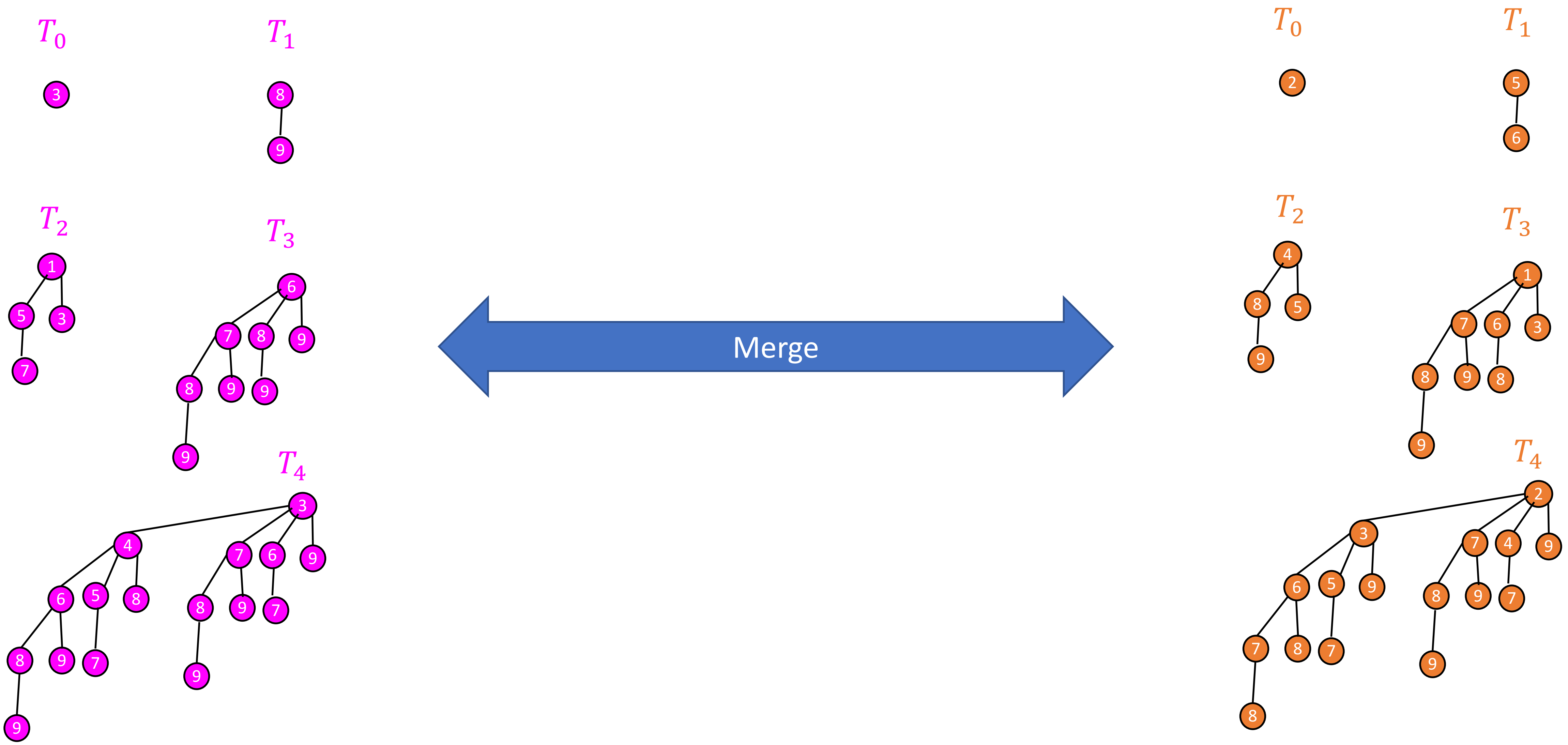


$$\begin{array}{r} 01100 \\ +\ 10011 \\ \hline 11111 \end{array}$$

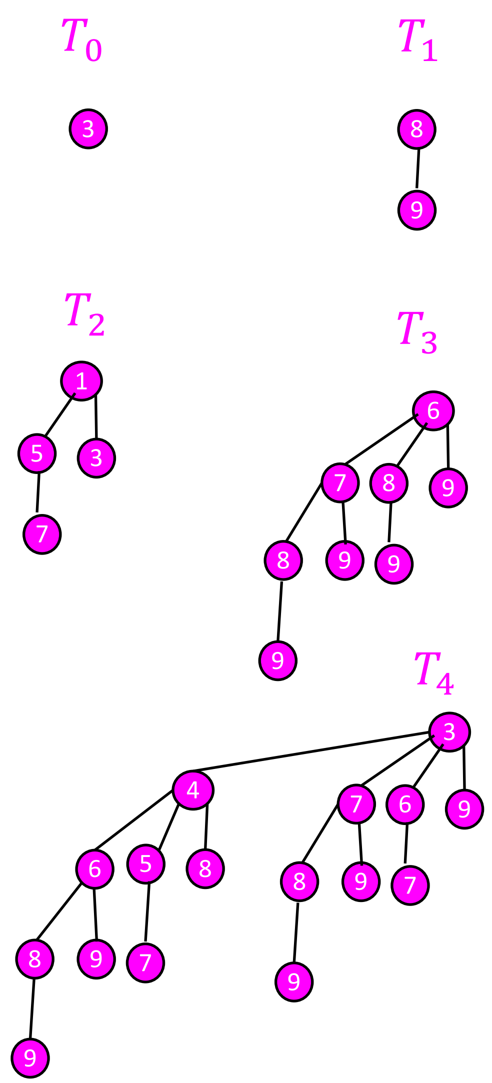
Best scenario: no combines (carries)

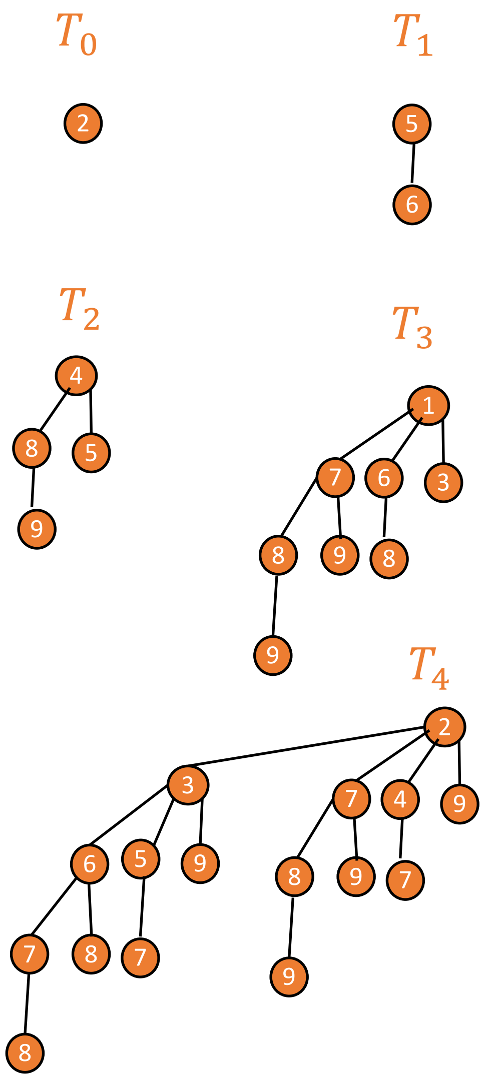


Worst scenario: maximum combines (carries)



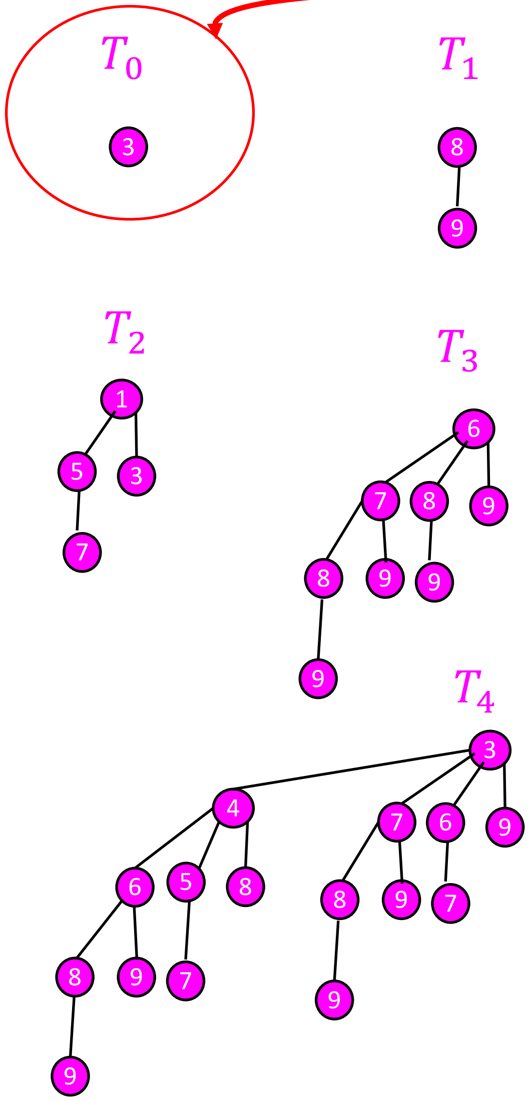
Worst scenario: maximum combines (carries)

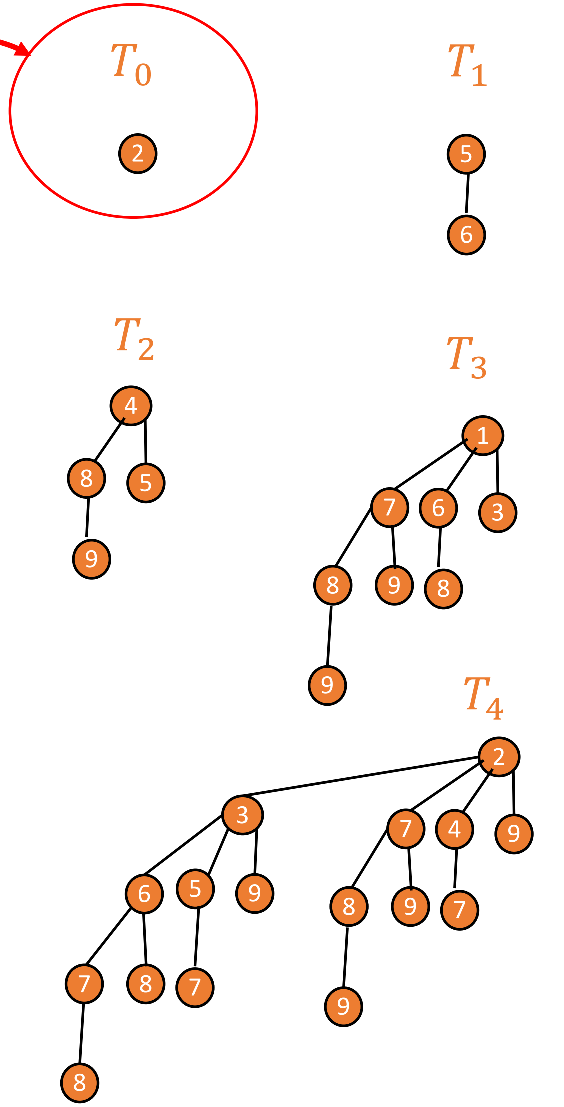




$$
\begin{array}{r}
\underline{1111} \\
11111 \\
+\ \underline{11111} \\
111110
\end{array}
$$

Worst scenario: maximum combines (carries)

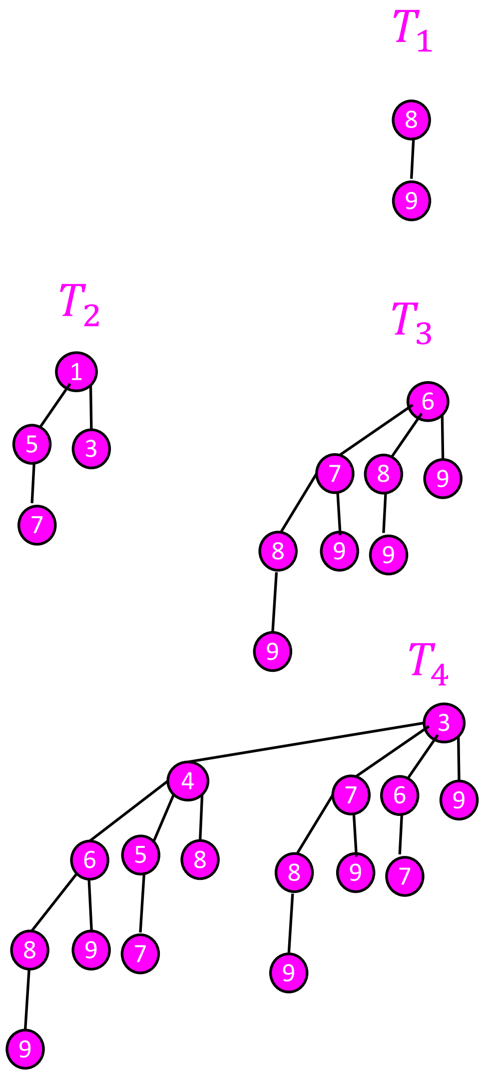




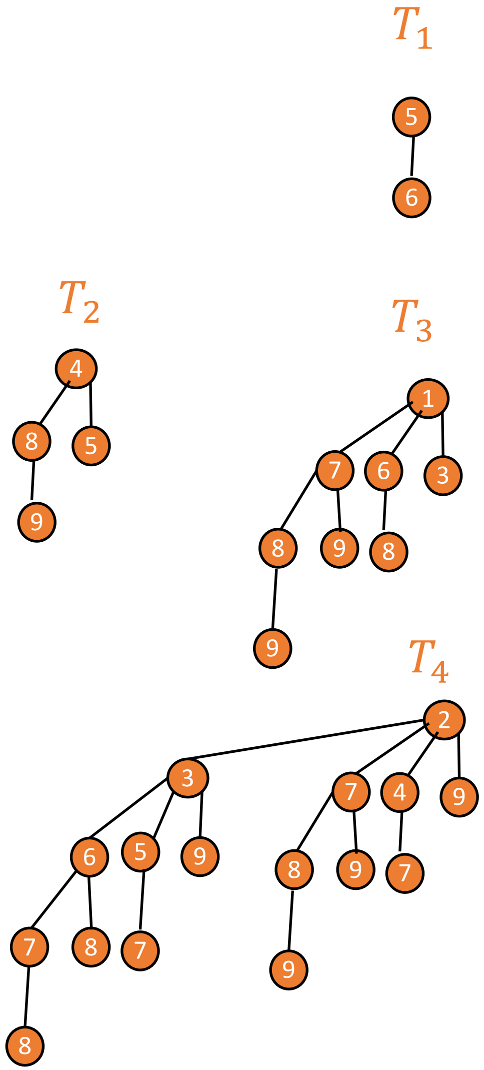

```
  1111
  11111
+ 11111
 111110
```

Gotta combine the two T_0s, which means that there will **not** be a T_0 in the final queue (LSB of sum is 0).

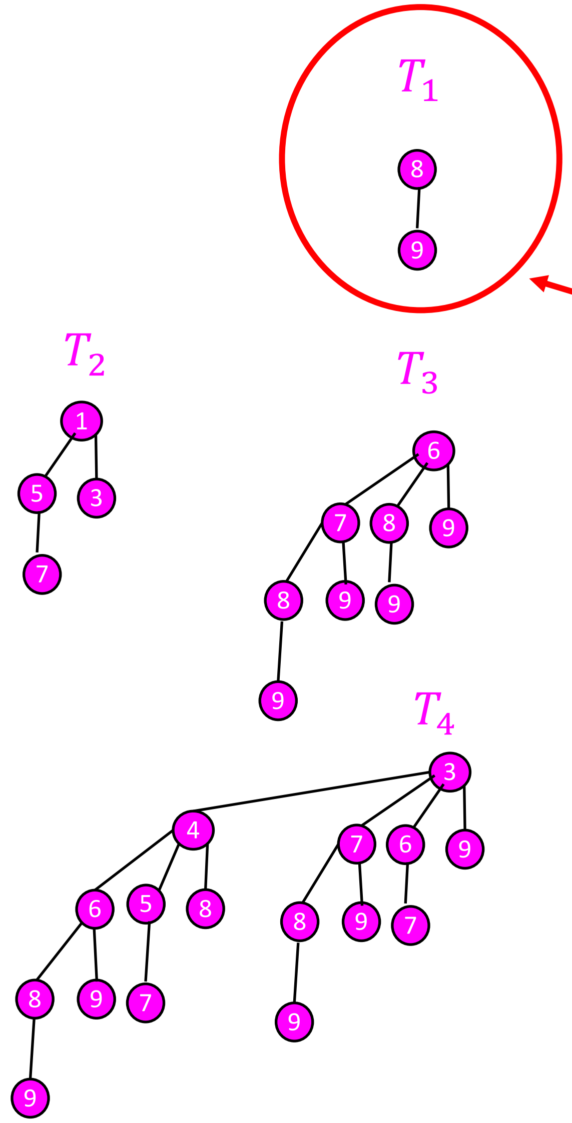
Worst scenario: maximum combines (carries)

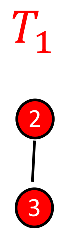


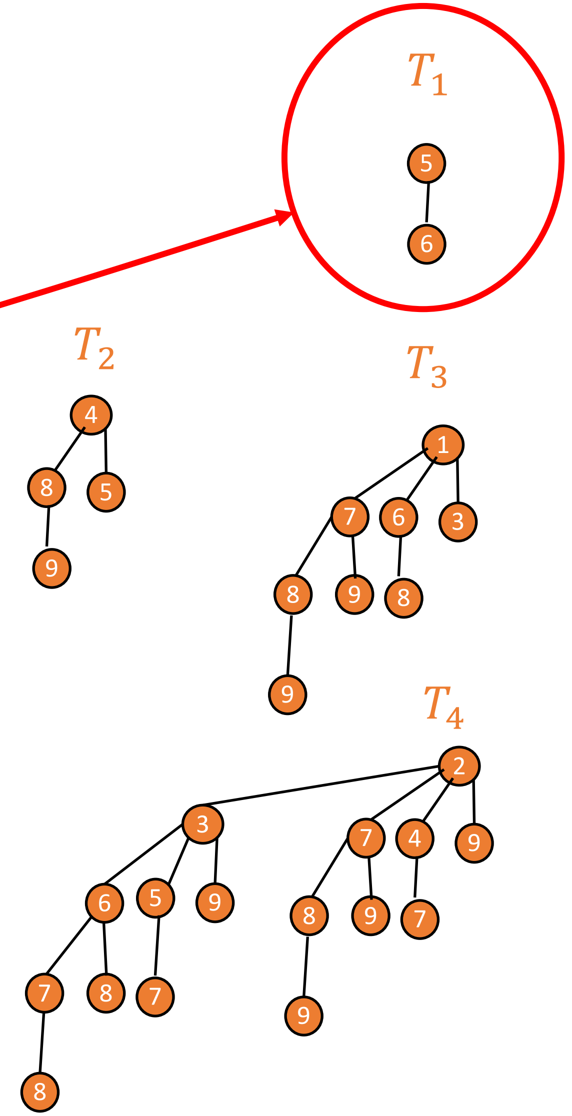
$$
\begin{array}{r}
1111 \\
11111 \\
+\ 11111 \\
\hline
111110
\end{array}
$$



Worst scenario: maximum combines (carries)



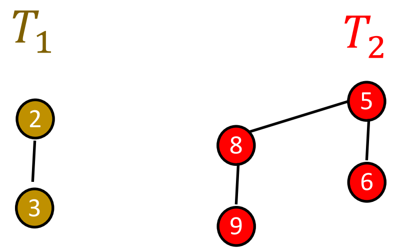


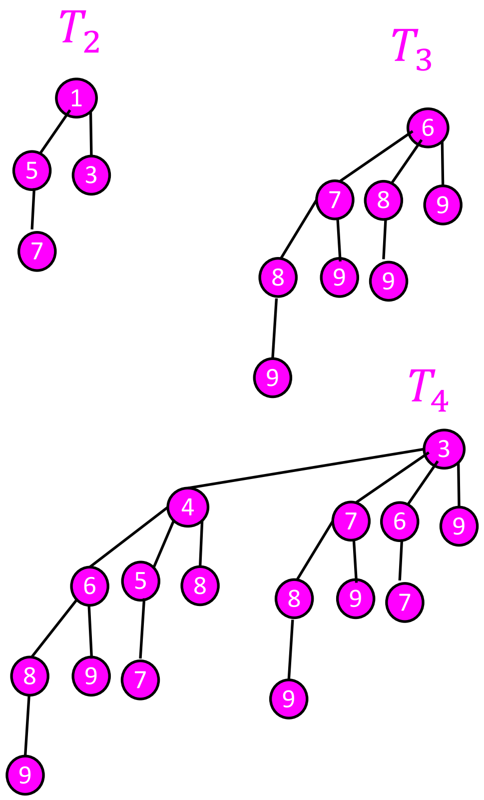


1111
11111
+ 11111
111110

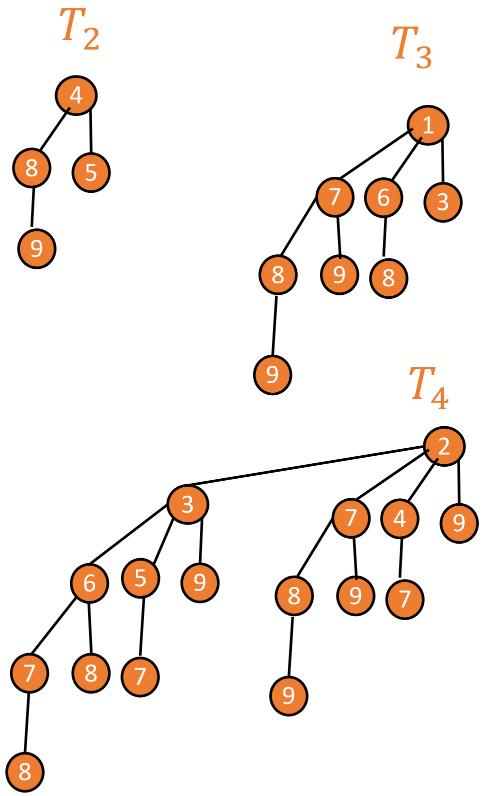
Combine the two queues' T_1s to produce a T_2 which we will carry over, while the previously produced T_1 persists in the output!

Worst scenario: maximum combines (carries)

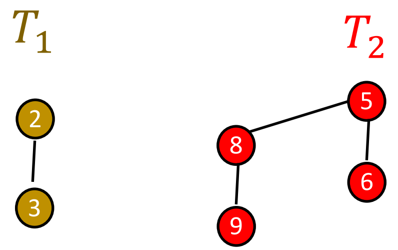


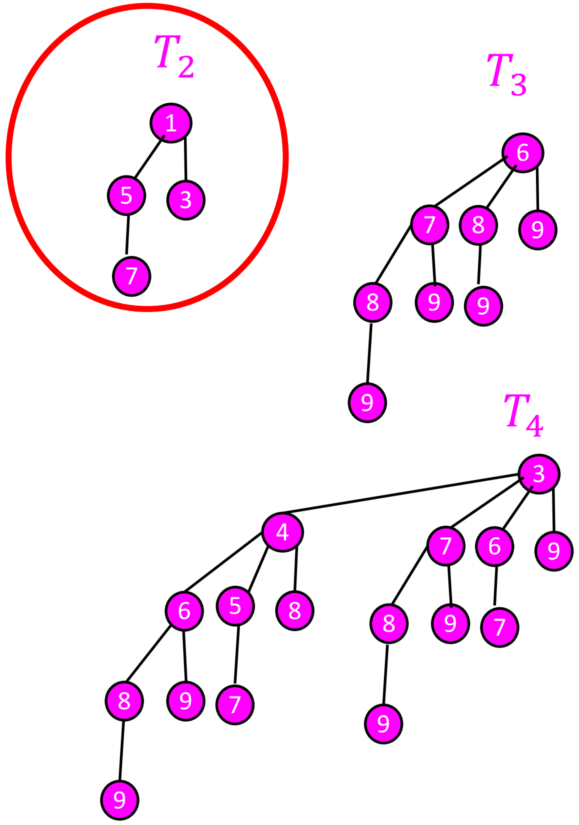


$$
\begin{array}{r}
1111 \\
11111 \\
+\ 11111 \\
\hline
111110
\end{array}
$$

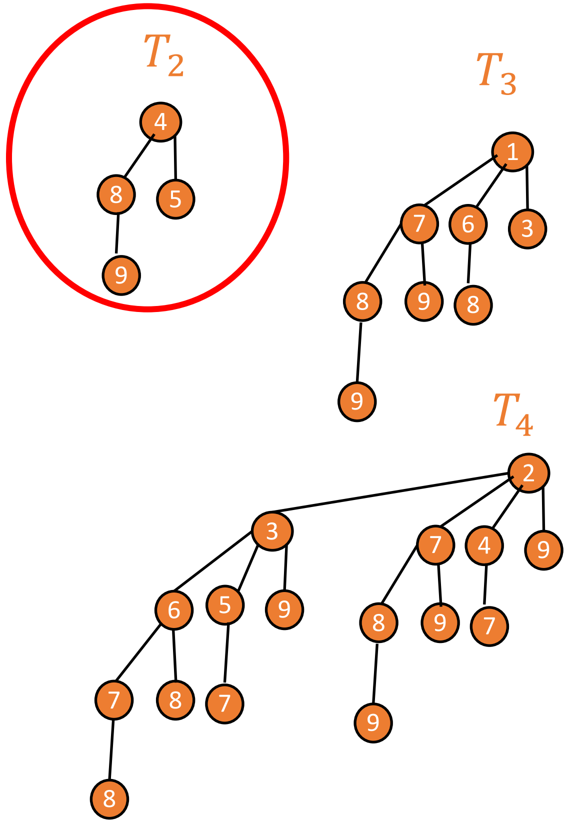


Worst scenario: maximum combines (carries)

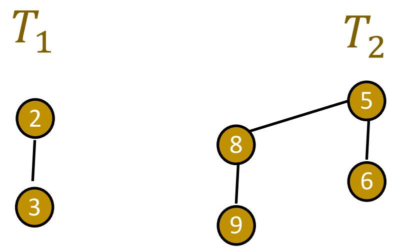


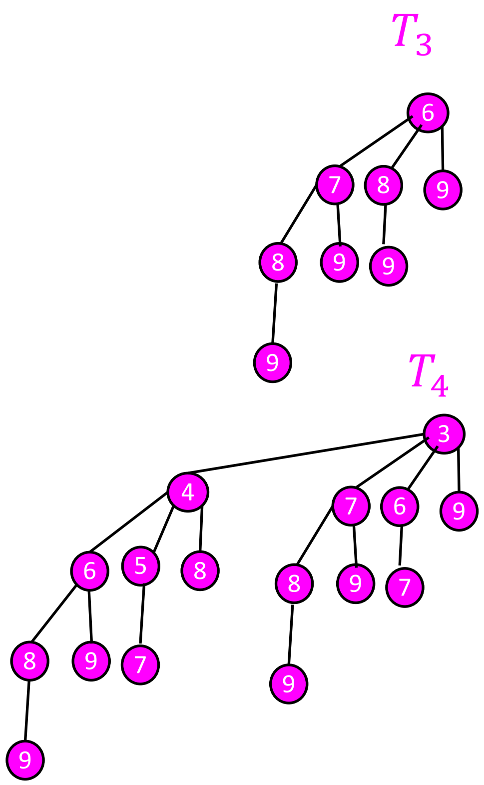


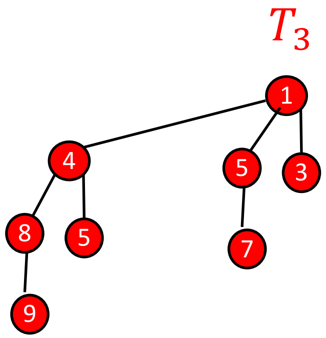
$$
\begin{array}{r}
\underline{1111} \\
11111 \\
+\ \underline{11111} \\
111110
\end{array}
$$



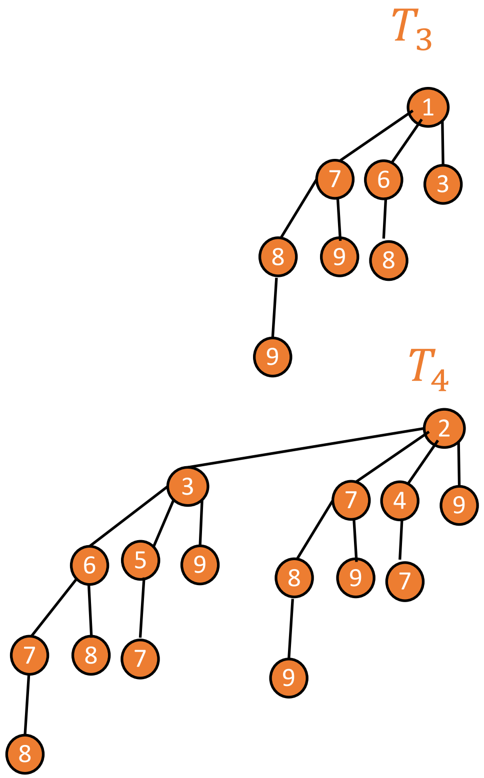
Worst scenario: maximum combines (carries)



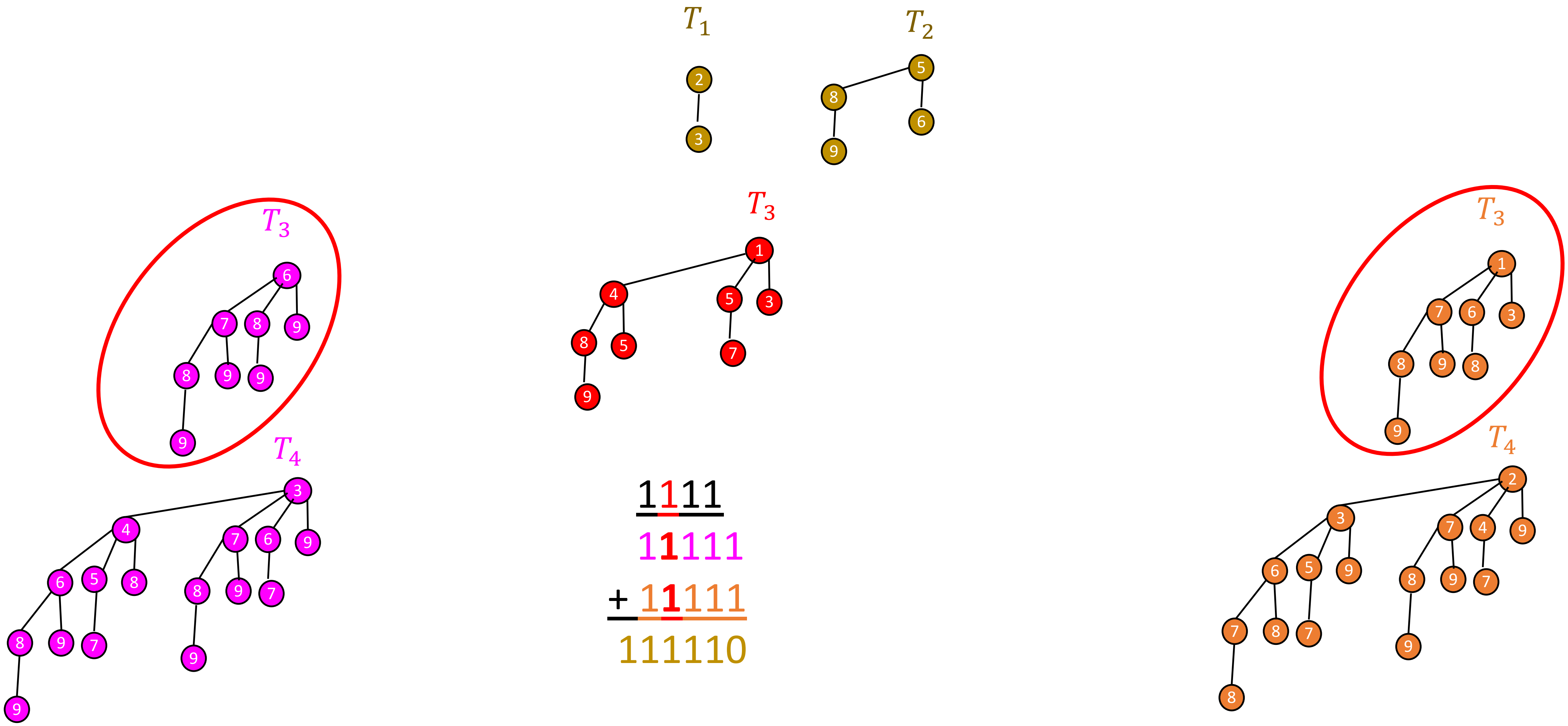




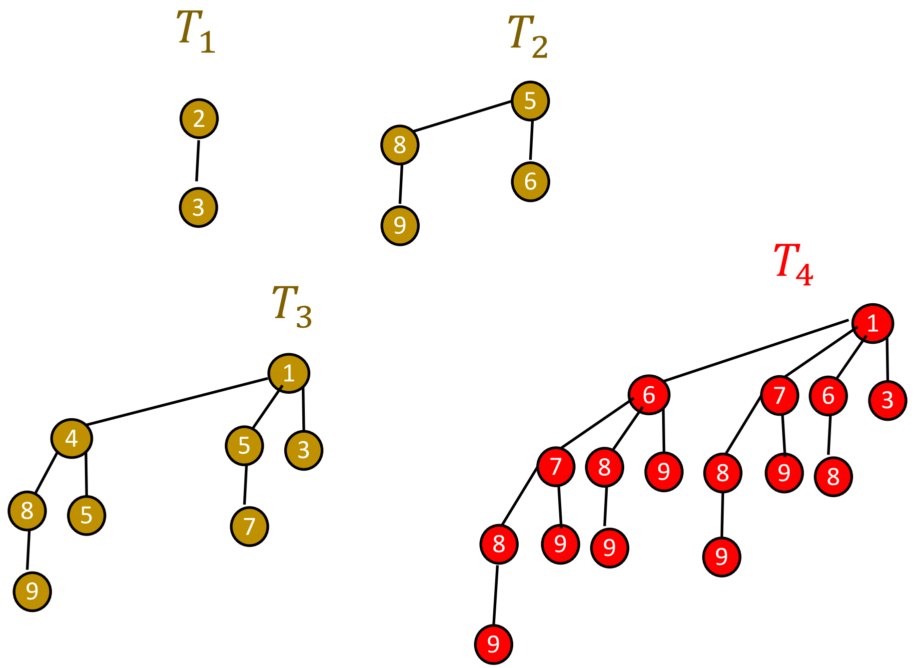
$$
\begin{array}{r}
\underline{1111} \\
11111 \\
+\ \underline{11111} \\
111110
\end{array}
$$

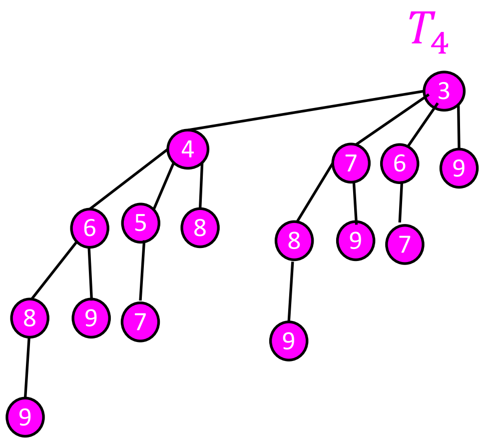


Worst scenario: maximum combines (carries)

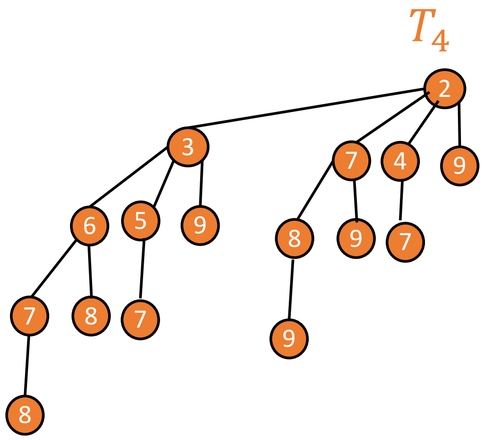


Worst scenario: maximum combines (carries)

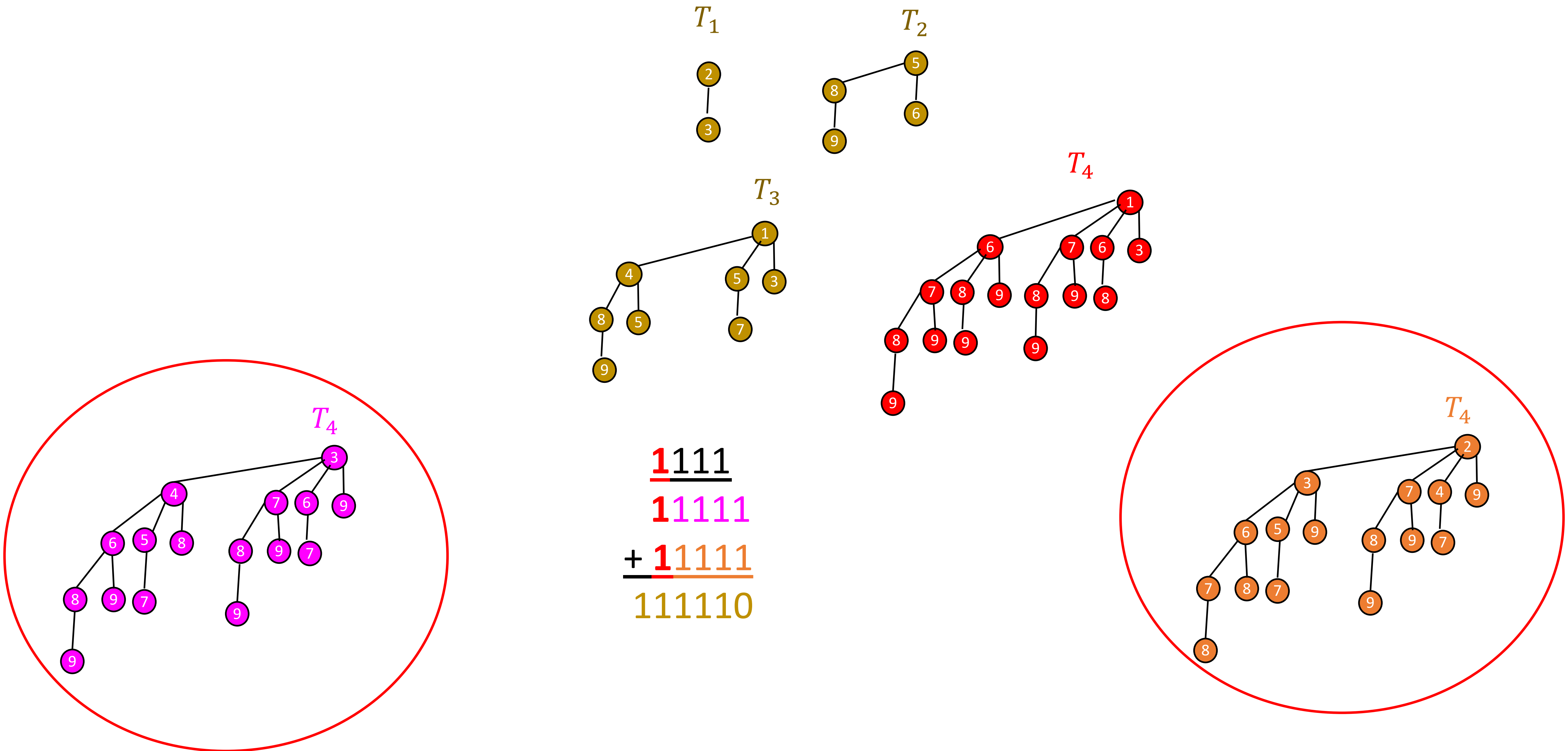




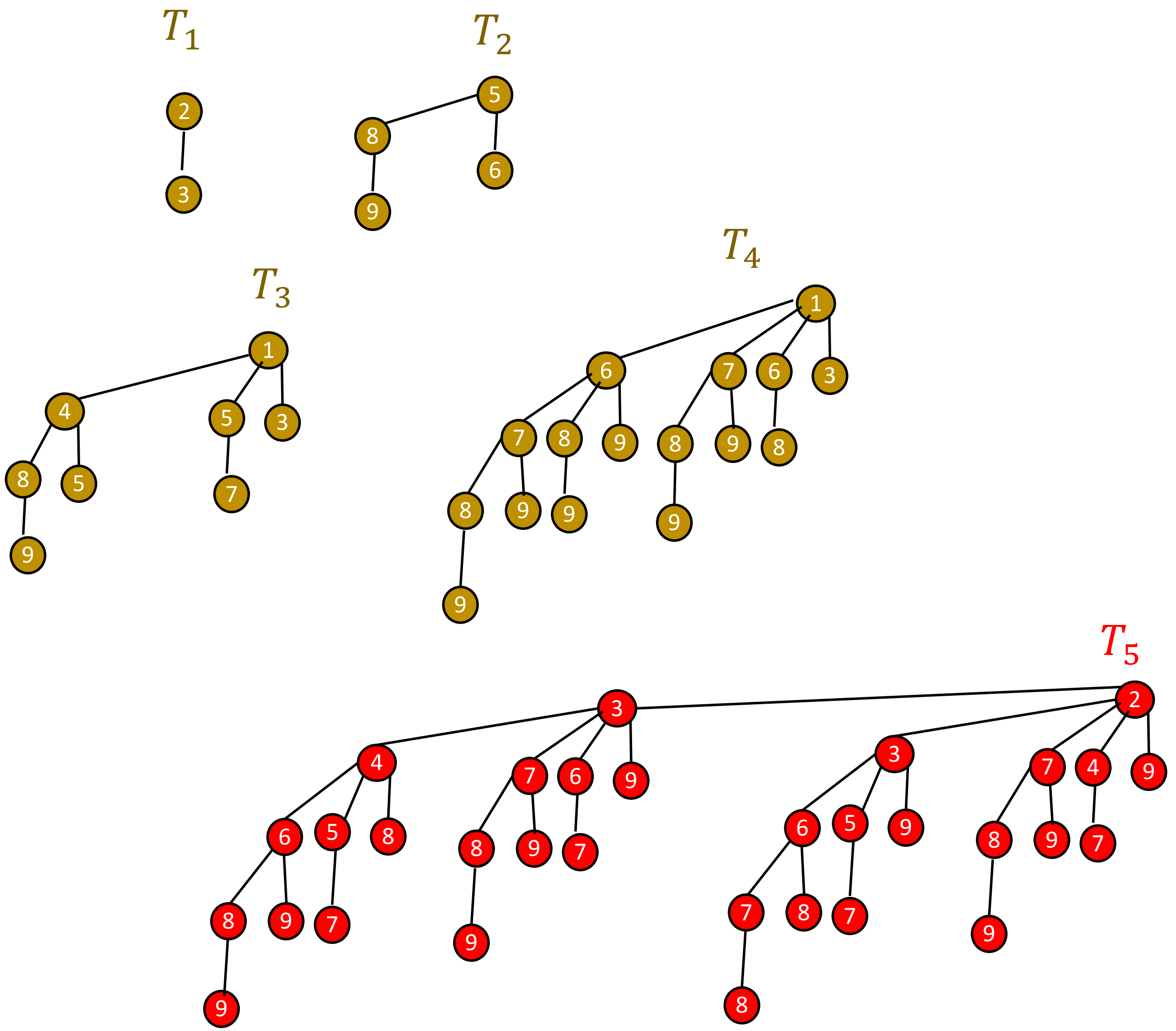
$$
\begin{array}{r}
1111 \\
11111 \\
+\ 11111 \\
\hline
111110
\end{array}
$$



Worst scenario: maximum combines (carries)



Worst scenario: maximum combines (carries)

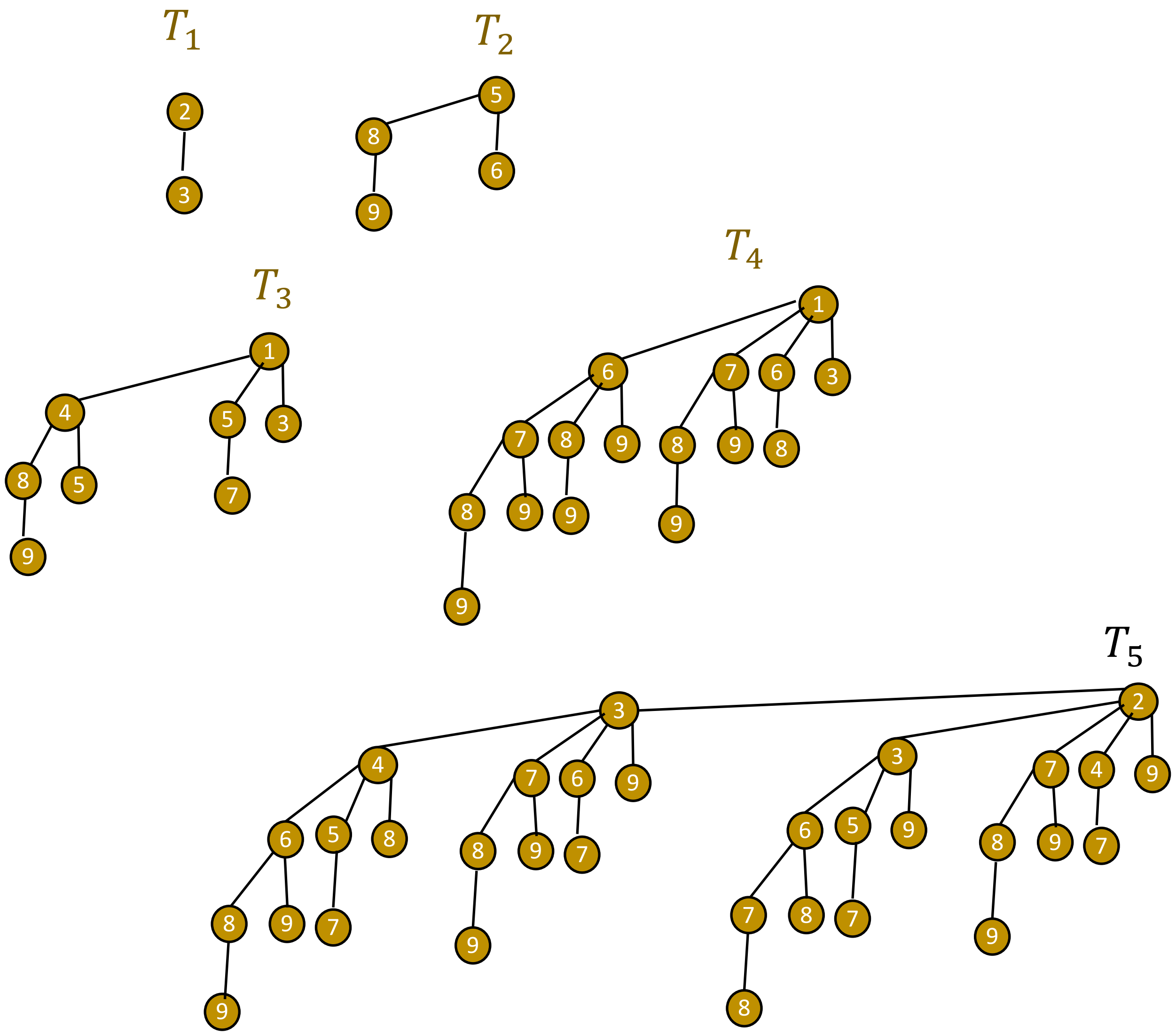


$$\begin{array}{r} \underline{1111} \\ 11111 \\ +\ \underline{11111} \\ 111110 \end{array}$$

Worst scenario: maximum combines (carries)

Had to merge 5 pairs of trees, so paid 5 unit costs.

Generally, for adding k-length bit vectors of all 1s (or, merging “full” binomial queues of rank k), we pay k unit costs.



$$
\begin{array}{r}
\underline{1111} \\
11111 \\
+\ \underline{11111} \\
111110
\end{array}
$$

How does k grow?

- Suppose we want to merge two binomial queues of the same maximum rank r. Both queues have n keys total.

How does k grow?

- Suppose we want to merge two binomial queues of the same maximum rank r. Both queues have n keys total.
- How does k, the number of paid unit costs, grow as a function of n?

How does k grow?

- Suppose we want to merge two binomial queues of the same maximum rank r. Both queues have n keys total.
- How does k, the number of paid unit costs, grow as a function of n?

$\mathcal{O}(n)$

$\mathcal{O}(1)$

$\mathcal{O}(\log_2 n)$

Something else (what?)

How does k grow?

- Suppose we want to merge two binomial queues of the same maximum rank r. Both queues have n keys total.
- How does k, the number of paid unit costs, grow as a function of n?

$\mathcal{O}(n)$

$\mathcal{O}(1)$

$\mathcal{O}(\log_2 n)$

Something else (what?)

- Worst-case scenario is two r −bit vectors of all 1s, which means that we will have to merge all same-rank pairs of trees.
- But we already know that a Binomial Queue with n elements has $\lceil \log_2 n \rceil$ trees!

How does k grow?

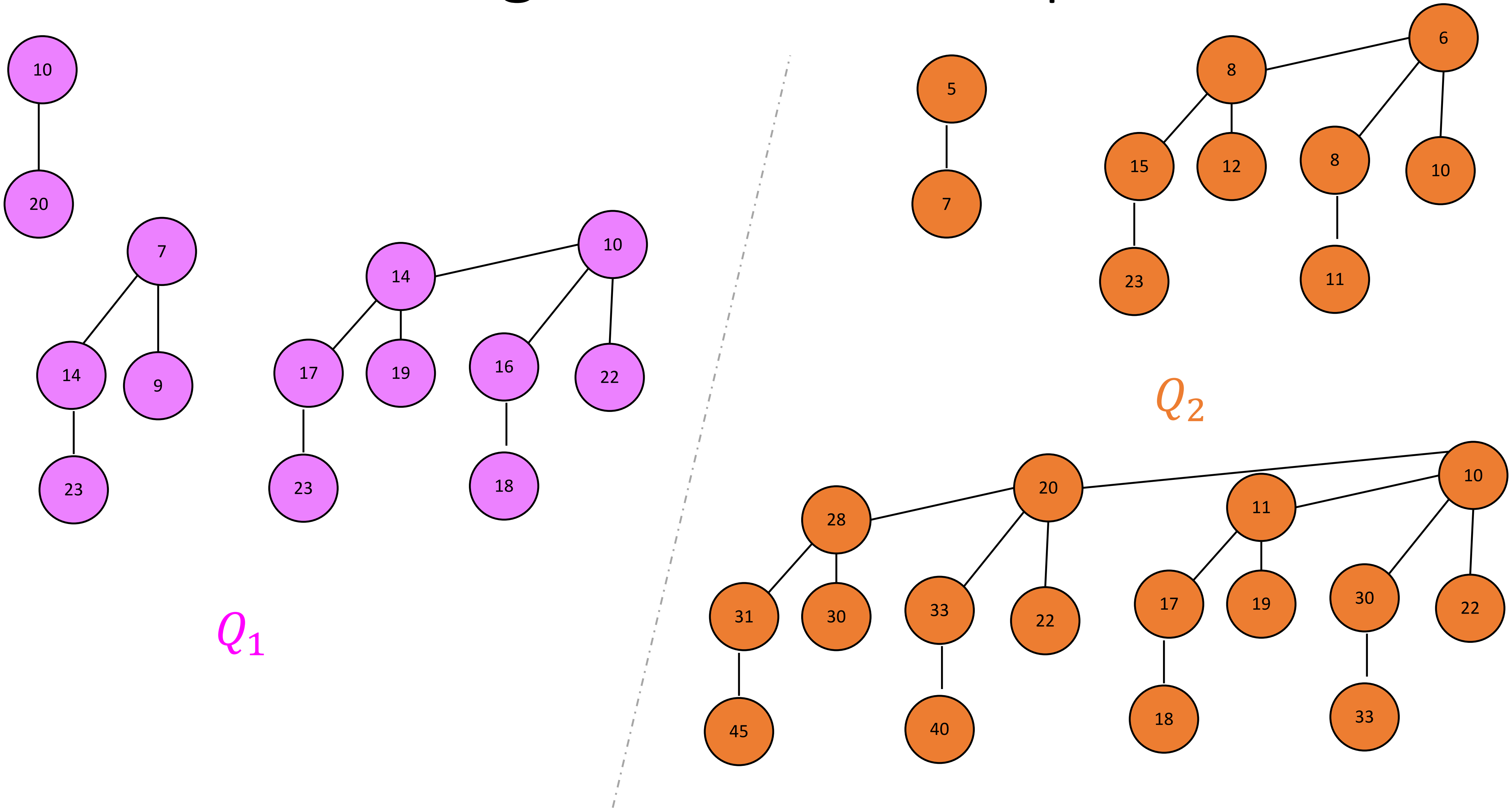
- Suppose we want to merge two binomial queues of the same maximum rank r. Both queues have n keys total.
- How does k, the number of paid unit costs, grow as a function of n?

highest rank present = maximum payable cost!

$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_2 n)$	Something else (what?)

- Worst-case scenario is two r −bit vectors of all 1s, which means that we will have to merge all same-rank pairs of trees.
- But we already know that a Binomial Queue with n elements has $\lceil \log_2 n \rceil$ trees!

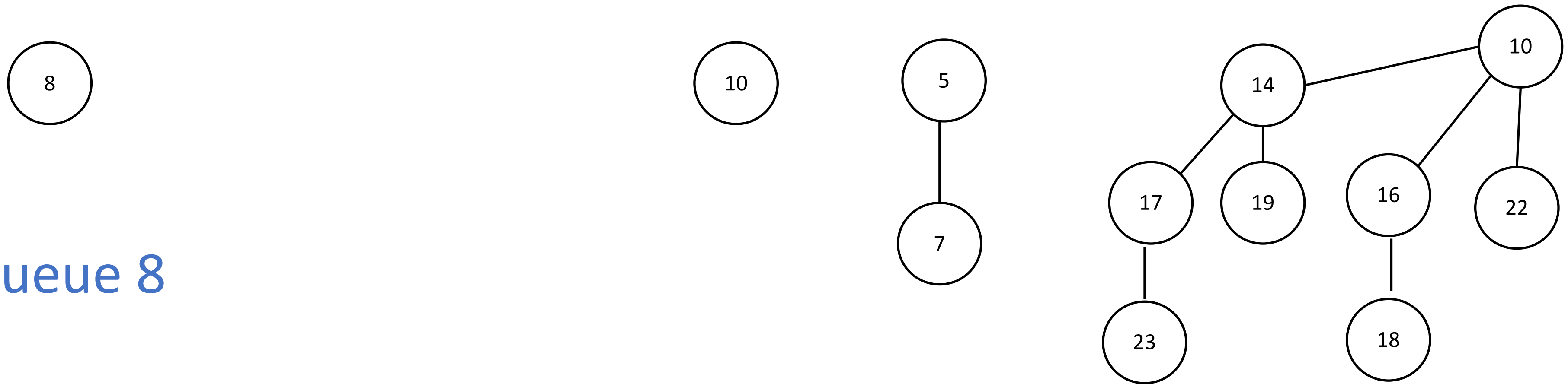
Merge these for me please!



Enqueue

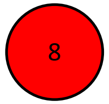
- To enqueue an element in a Binomial Queue, we simply make a T_0 out of it and merge this singleton queue with the rest of the queue.
- Since we can generate up to $\log_2 n$ pairs of trees that we should merge, enqueue() is also a $\mathcal{O}(\log_2 n)$ operation .

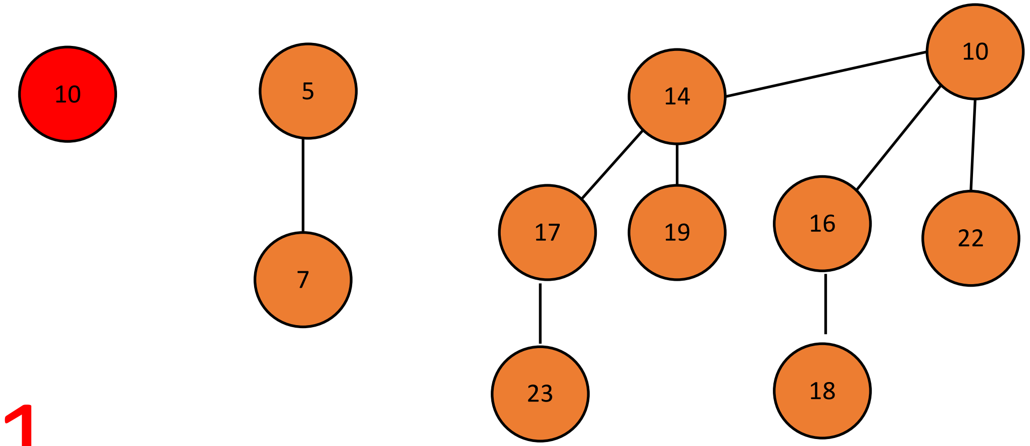
Enqueuing example



Enqueue 8

Enqueuing example

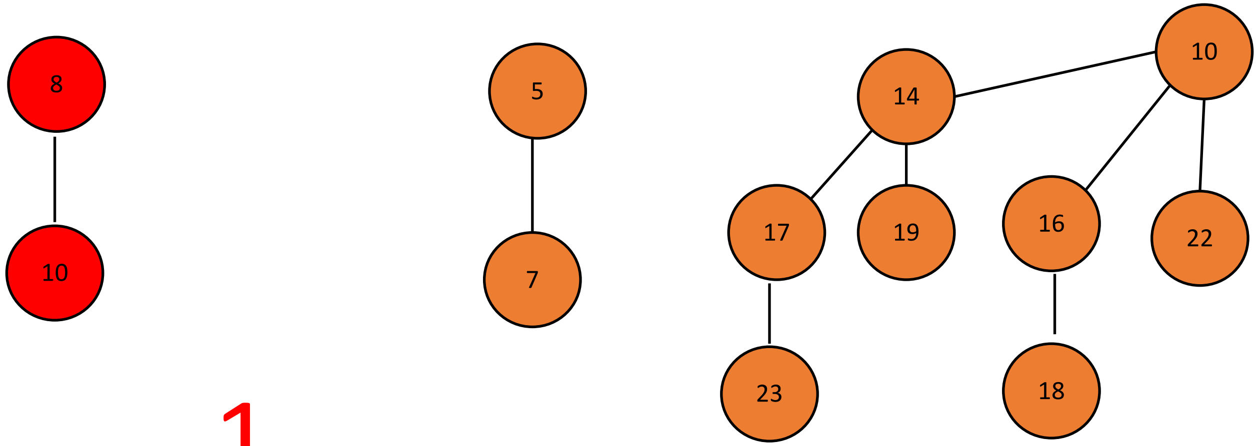




Merge two T_0s into a T_1 …

$$
\begin{array}{r}
\underline{1} \\
1\ 0\ 1\ 1 \\
+\ \underline{0\ 0\ 0\ 1} \\
0
\end{array}
$$

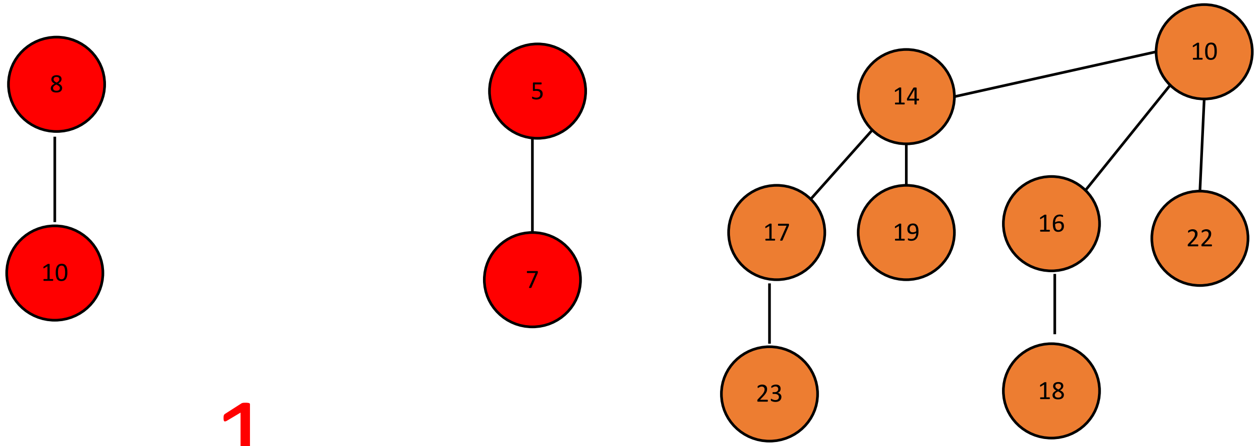
Enqueuing example



Merge two T_0s into a T_1 …

$$
\begin{array}{r}
\underline{1} \\
1\;0\;1\;1 \\
+\;\underline{0\;0\;0\;1} \\
0
\end{array}
$$

Enqueuing example

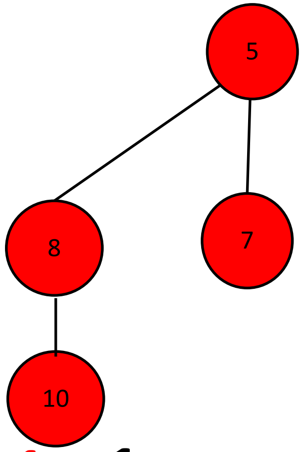


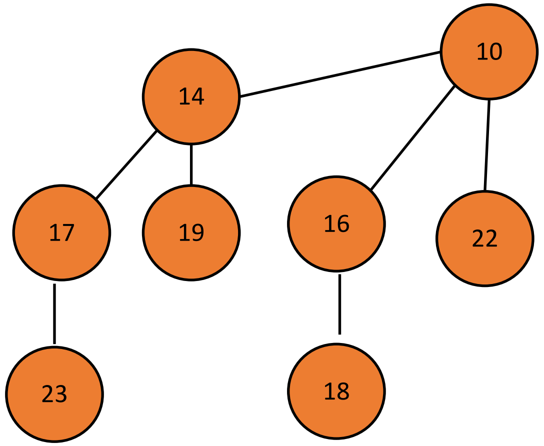
Merge two T_1s into a T_2 …

$$
\begin{array}{r}
\underline{1} \\
1\,0\,1\,1 \\
+\,\underline{0\,0\,0\,1} \\
0
\end{array}
$$

Enqueuing example

Merge two T_1s into a T_2 …

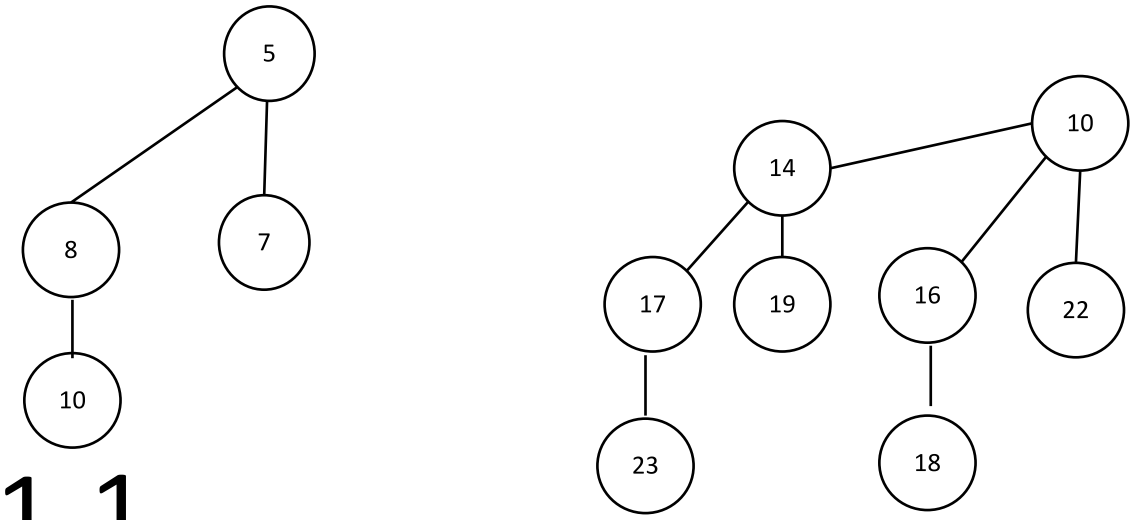




$$\begin{array}{r} \underline{1}\ \underline{1} \\ 1\ 0\ 1\ 1 \\ +\ \underline{0\ 0\ 0\ 1} \\ 0\ 0 \end{array}$$

Enqueuing example

Done! ☺



$$
\begin{array}{r}
\underline{1}\,\underline{1} \\
1\,0\,1\,1 \\
+\,\underline{0\,0\,0\,1} \\
1\,1\,0\,0
\end{array}
$$

`getMin()`

- Since there are $\lceil \log_2 n \rceil$ Binomial Trees in a Binomial queue with n keys, and the roots are connected via a linked list, `getMin()` is $\mathcal{O}(\log_2 n)$.

getMin()

- Since there are $\lceil \log_2 n \rceil$ Binomial Trees in a Binomial queue with n keys, and the roots are connected via a linked list, `getMin()` is $\mathcal{O}(\log_2 n)$.
- Or is it?

`getMin()`

- Since there are $\lceil \log_2 n \rceil$ Binomial Trees in a Binomial queue with n keys, and the roots are connected via a linked list, `getMin()` is $\mathcal{O}(\log_2 n)$.
- Or is it?
- Maintaining a pointer to the minimum element makes it $\mathcal{O}(1)$!

`getMin()`

- Since there are $[\log_2 n]$ Binomial Trees in a Binomial queue with n keys, and the roots are connected via a linked list, `getMin()` is $\mathcal{O}(\log_2 n)$.
- Or is it?
- Maintaining a pointer to the minimum element makes it $\mathcal{O}(1)$!
- Begin from an empty queue, current global minimum is `null`. Every time you insert a new element and merge two trees, compare the minimum of the two roots to the stored global minimum. If smaller, replace global minimum.

`getMin()`

- Since there are $\lceil \log_2 n \rceil$ Binomial Trees in a Binomial queue with n keys, and the roots are connected via a linked list, `getMin()` is $\mathcal{O}(\log_2 n)$.
- Or is it?
- Maintaining a pointer to the minimum element makes it $\mathcal{O}(1)$!

- Begin from an empty queue, current global minimum is `null`. Every time you insert a new element and merge two trees, compare the minimum of the two roots to the stored global minimum. If smaller, replace global minimum.
  - Increases the unit cost of merging two trees by one additional comparison, but it's definitely worth it.

`dequeue() (deleteMin())`

- Dequeueing consists of finding the minimum root, removing it and re-inserting all of its children.
  - Reinserting = put them all in a temporary queue and then merge with current one.

`dequeue() (deleteMin())`

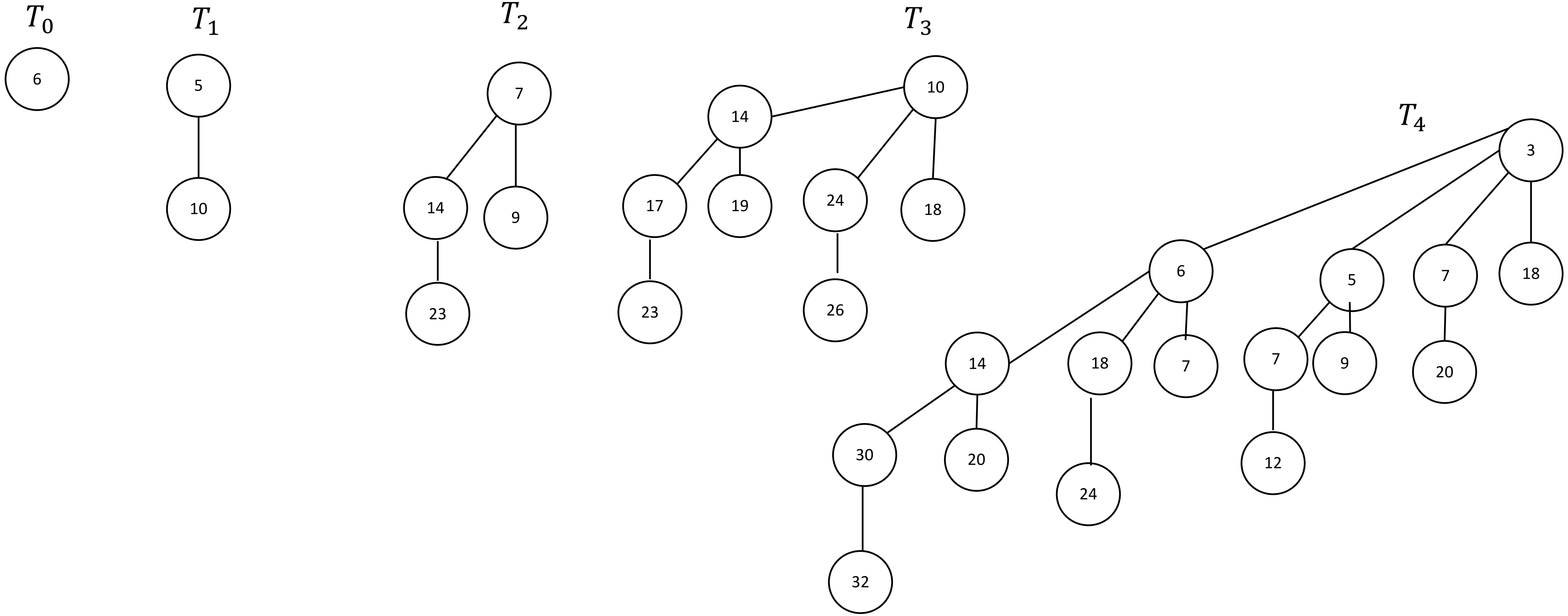
- Dequeueing consists of finding the minimum root, removing it and re-inserting all of its children.
  - Reinserting = put them all in a temporary queue and then merge with current one.
- A Binomial Tree T_k consists of a root pointing to k children.
  - And $k \leq \log_2 n$ (\` $=$ ′ only when we have a single binomial tree in the queue)
  - So we have $\mathcal{O}(\log_2 n)$ trees to enqueue in a temporary queue
    - $\mathcal{O}(\log_2 log_2 n)$ for this step, since the temp queue will have at most $\log_2 n$ binomial trees!

`dequeue() (deleteMin())`

- Dequeueing consists of finding the minimum root, removing it and re-inserting all of its children.
  - Reinserting = put them all in a temporary store and then merge with current one.
- A Binomial Tree T_k consists of a root pointing to k children.
  - And $k \ \leq \log_2 n$ (\` $=$ ′ only when we have a single binomial tree in the queue)
  - So we have $\mathcal{O}(\log_2 n)$ trees to put in a temporary queue
- Then, we merge the temporary queue with the original one.
  - $\mathcal{O}(\log_2 n)$
  - So, in total, we pay the logarithmic cost twice
  - In total: $\mathcal{O}(\log_2 n)$ with a 2 up front.

Dequeueing Example

- Delete the minimum element of this binomial queue.



Some questions

1. Suppose we have a Binomial Queue with 11,021 elements. Does it contain a T_0?

Some questions

1. Suppose we have a Binomial Queue with 11,021 elements. Does it contain a T_0?
   - ***Yes***, since 11,021 is an odd number, and the only way to write it in binary involves a ‘1’ as the LSB (which means that a T_0 ***has*** to be present*)*!

Some questions

1. Suppose we have a Binomial Queue with 11,021 elements. Does it contain a T_0?
   - ***Yes***, since 11,021 is an odd number, and the only way to write it in binary involves a '1' as the LSB (which means that a T_0 ***has*** to be present)!
2. Suppose we have a Binomial Queue with 1023 elements. What is the rank of the highest rank Binomial Tree in this queue?

Some questions

1. Suppose we have a Binomial Queue with 11,021 elements. Does it contain a T_0?
   - ***Yes***, since 11,021 is an odd number, and the only way to write it in binary involves a '1' as the LSB (which means that a T_0 ***has*** to be present*)*!
2. Suppose we have a Binomial Queue with 1023 elements. What is the rank of the highest rank Binomial Tree in this queue?
   - ***9 (nine)***, since $1023 = \sum_{i=0}^{9} 2^i$

Some questions

1. Suppose we have a Binomial Queue with 11,021 elements. Does it contain a T_0?
   - ***Yes***, since 11,021 is an odd number, and the only way to write it in binary involves a '1' as the LSB (which means that a T_0 ***has*** to be present)!
2. Suppose we have a Binomial Queue with 1023 elements. What is the rank of the highest rank Binomial Tree in this queue?
   - ***9 (nine)***, since $1023 = \sum_{i=0}^{9} 2^i$
3. We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **11111111** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

Some questions

3) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **11111111** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

$$
\begin{array}{r}
1\,0\,1\,1\,0\,0\,1\,1 \\
\otimes\ 1\,1\,1\,1\,1\,1\,1\,1 \\
\hline
0\,1\,0\,0\,1\,1\,0\,0
\end{array}
$$

Some questions

3) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **11111111** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

$$
\begin{array}{rl}
 & 1\,0\,1\,1\,0\,0\,1\,1 \quad A \\
\otimes & 1\,1\,1\,1\,1\,1\,1\,1 \\
\hline
 & 0\,1\,0\,0\,1\,1\,0\,0 \quad B
\end{array}
$$

Some questions

3) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **11111111** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

$$\begin{array}{rl} & 1\,0\,1\,1\,0\,0\,1\,1 \quad A \\ \otimes & 1\,1\,1\,1\,1\,1\,1\,1 \\ \hline & 0\,1\,0\,0\,1\,1\,0\,0 \quad B \end{array}$$

$$\begin{array}{rl} & 0\,0\,0\,0\,0\,0\,0 \\ \hline & 1\,0\,1\,1\,0\,0\,1\,1 \\ + & 0\,1\,0\,0\,1\,1\,0\,0 \\ \hline & 1\,1\,1\,1\,1\,1\,1\,1 \end{array}$$

Some questions

3) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **11111111** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

$$\begin{array}{rl} & 1\,0\,1\,1\,0\,0\,1\,1 \quad A \\ \otimes & 1\,1\,1\,1\,1\,1\,1\,1 \\ \hline & 0\,1\,0\,0\,1\,1\,0\,0 \quad B \end{array}$$

$$\begin{array}{rl} & 0\,0\,0\,0\,0\,0\,0 \\ \hline & 1\,0\,1\,1\,0\,0\,1\,1 \\ + & 0\,1\,0\,0\,1\,1\,0\,0 \\ \hline & 1\,1\,1\,1\,1\,1\,1\,1 \end{array}$$

Zero trees merged!

Some questions

4) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with $\boxed{\textbf{00000000}}$ to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

Some questions

4) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **00000000** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

$$\begin{array}{rl} 1\,0\,1\,1\,0\,0\,1\,1 & A \\ \otimes\; 0\,0\,0\,0\,0\,0\,0\,0 & \\ \hline 1\,0\,1\,1\,0\,0\,1\,1 & B \end{array}$$

Some questions

4) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **00000000** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

$$\begin{array}{rl} 1\,0\,1\,1\,0\,0\,1\,1 & A \\ \otimes\; 0\,0\,0\,0\,0\,0\,0\,0 & \\ \hline 1\,0\,1\,1\,0\,0\,1\,1 & B \end{array}$$

$$\begin{array}{r} 0\,0\,1\,1\,0\,0\,1\,1 \\ \hline 1\,0\,1\,1\,0\,0\,1\,1 \\ +\;1\,0\,1\,1\,0\,0\,1\,1 \\ \hline 1\,0\,1\,1\,0\,0\,1\,1\,0 \end{array}$$

Some questions

4) We have a Bin Queue A described by the binary string **10110011**. We take the XOR of this string with **00000000** to produce a new Bin Queue B (with some elements in it, it doesn't matter for this question). If I were to merge A and B, **how many trees would I have to merge**?

$$
\begin{array}{rl}
1\,0\,1\,1\,0\,0\,1\,1 & A \\
\otimes\ 0\,0\,0\,0\,0\,0\,0\,0 & \\
\hline
1\,0\,1\,1\,0\,0\,1\,1 & B
\end{array}
$$

$$
\begin{array}{r}
0\,0\,1\,1\,0\,0\,1\,1 \\
1\,0\,1\,1\,0\,0\,1\,1 \\
+\ 1\,0\,1\,1\,0\,0\,1\,1 \\
\hline
1\,0\,1\,1\,0\,0\,1\,1\,0
\end{array}
$$

Five trees merged!

Binomial vs Binary heaps

Binomial Heap	Binary Heap
Efficient merging ($\mathcal{O}(\log_2 n)$)	Great cache locality (in array implementation)
`getMin()` and `enqueue()` just as efficient	Easy implementation
Big constant in front of $\log_2 n$ for enqueue() and dequeue()	Can be used for sorting (heapsort)
	Inefficient merging ($\mathcal{O}(\lvert q_1 \rvert + \lvert q_2 \rvert)$)