# Congratulations! You passed!

**Grade received** 100% **Latest Submission Grade** 100% **To pass** 80% or higher

**Go to next item**

**Loading...**

**1.** Using the convention followed by the video lessons, given three disjoint sets (1,3,5,7), (2,8) and (4,6), which one of these sets would be referenced by the value 3? **1 / 1 point**

- (1,3,5,7)
- (2,8)
- (4,6)
- None of the above.

**Correct**
A disjoint set is referenced by one of the values in the set.

**2.** What is the union of the disjoint sets (1,3,5,7) and (2,8)? **1 / 1 point**

- (1,2,3,5,7,8)
- (3,11)
- (2,6,8,10,14,16,24,40,56)
- ((1,2),(1,8),(3,2),(3,8),(5,2),(5,8),(7,2),(7,8))

**Correct**
The union of disjoint sets contains only each element from the first set and each element from the second set.

**3.** What happens when you take the union of two disjoint sets that contain the same value? **1 / 1 point**

- ○ The elements cancel and neither appears in the union of the two disjoint sets.
- ◉ Two different disjoint sets by definition can never share the same value.
- ○ The union operation must first check to see if the same element appears in both disjoint sets and then ensures the element appears only once in the resulting union set.
- ○ Any element found in both disjoint sets will appear twice in the union of these two disjoint sets.

✓ **Correct**

Disjoint sets represent a partitioning of unique values into subsets that do not have any items in common. That is, each value belongs to exactly one of the sets. This is why each element can be used as an array index look up its "up-tree" parent, which represents the set the element belongs to.

**4.** According to the distjoint set array representation in the video lessons, Which of the following arrays would NOT be a valid representation of the disjoint set (1,3,5,7)? **1 / 1 point**

◉

| **3** | **-1** | **5** | **-1** | **7** | **-1** | **1** | **-1** |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

○

| **5** | **-1** | **-1** | **-1** | **3** | **-1** | **1** | **-1** |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

○

| **-1** | **-1** | **1** | **-1** | **1** | **-1** | **1** | **-1** |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| -1 | -1 | 1 | -1 | 3 | -1 | 5 | -1 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Correct**

This is indeed not valid because there is no root of the up-tree. Element 1 points to element 3 which points to element 5 which points to element 7 which points to element 1, so no element in this disjoint set is the root and would represent the disjoint set.

**5.** When encoding height into the root of an up-tree, what value should be placed in element 7 of the following array? **1 / 1 point**

| 3 | -1 | 7 | -1 | 7 | -1 | ??? |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

- -1
- -4
- -3
- -2

**Correct**

The value should be equal to -1 minus the height. A singleton disjoint set would have height zero but there is no -0 and 0 would point to the 0th element of the array, so we increment the height by one and negate it before storing it in the root of the up-tree.

**6.** When encoding size into the root of an up-tree, what value should be placed in element 7 of the following array? **1 / 1 point**

| 3 | -1 | 7 | -1 | 7 | -1 | ??? |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

- (●) -4
- ( ) -2
- ( ) -1
- ( ) -3

✓ **Correct**
Correct. This up-tree represents the disjoint set (1,3,5,7) which has four elements.

**7.** When computing the union of two disjoint sets represented as up-trees in an array, (using proper path compression) which of these strategies results in a better overall run time complexity than the other options? **1 / 1 point**

- ( ) Always make the up-tree with fewer elements a subtree of the root of the up-tree with more elements.
- ( ) Always make the up-tree with a shorter height a subtree of the root of the up-tree with a larger height.
- ( ) The overall run time complexity is not affected by which up-tree is chosen to become a subtree of the other up-tree.
- (●) Using either size or height strategies above results in the same overall run time complexity.

✓ **Correct**
Indeed, the overall time complexity is the same when using either size or height strategies.

**8.** Recall that the iterated log function is denoted log*(n) and is defined to be **1 / 1 point**

- 0 for n <= 1, and
- 1 + log*(log(n)) for n > 1.

Let lg*(n) be this iterated log function computed using base 2 logarithms.

Blue Waters, housed at the University of Illinois, is the fastest university supercomputer in the world. It can run about 2^53 (about 13 quadrillion) instructions in a second. There are about 2^11 seconds in half an hour, so Blue Waters would run 2^64 instructions in about half an hour.

Which one of the following is equal to lg*(2^64)?

- (●) 5
- ( ) 6
- ( ) 65536
- ( ) 64

✓ **Correct**
lg*(2^64) = 1 + lg*(64) = 1 + 1 + lg*(6) = 1+ 1 + 1 + lg*(~2.6) = 1 + 1 + 1 + 1 + lg*(1.4) = 1 + 1 + 1 + 1 + 1 + lg*(0.5) = 5

**9.** Which of these is considered the least run-time complexity? **1 / 1 point**

- ( ) O(log N)
- (●) O(1)
- ( ) O(log* N)
- ( ) O(log log N)

✓ **Correct**
Constant time is indeed better than iterated-log time complexity, which is "practically" constant time, but does grow ever so slightly as N increases.

**10.** Which of the following best describes "path compression" as described in the video lessons to accelerate disjoint set operations? (Here we say "parent pointer" to mean whatever form of indirection is used to refer from a child to its parent; this could be a **1 / 1 point**

literal pointer or it could be an array index as in the lectures.)

- (selected) When traversing the up-tree from an element to its root, if any elements in the traversal (including the first element, but excluding the root itself) do not point directly to the root as their parent yet, they will have their parent pointer changed to point directly to the root.
- When the root of the up-tree containing an element is found, the element and all of its siblings that share the same parent have their parent pointers reset to point to the root node.
- When the root of an element's node is found, all of the descendants of the root have their parent pointer set to the root.
- When the root of the up-tree containing an element is found, both the element and its parent will always have their parent pointers set to point to the root node.

**Correct**

That's right: Path compression only flattens the lineage of nodes in an up-tree from an element to the root, and not all of the elements in the up-tree every time. This has amortized benefits as the data structure is optimized over the process of several union and find operations.