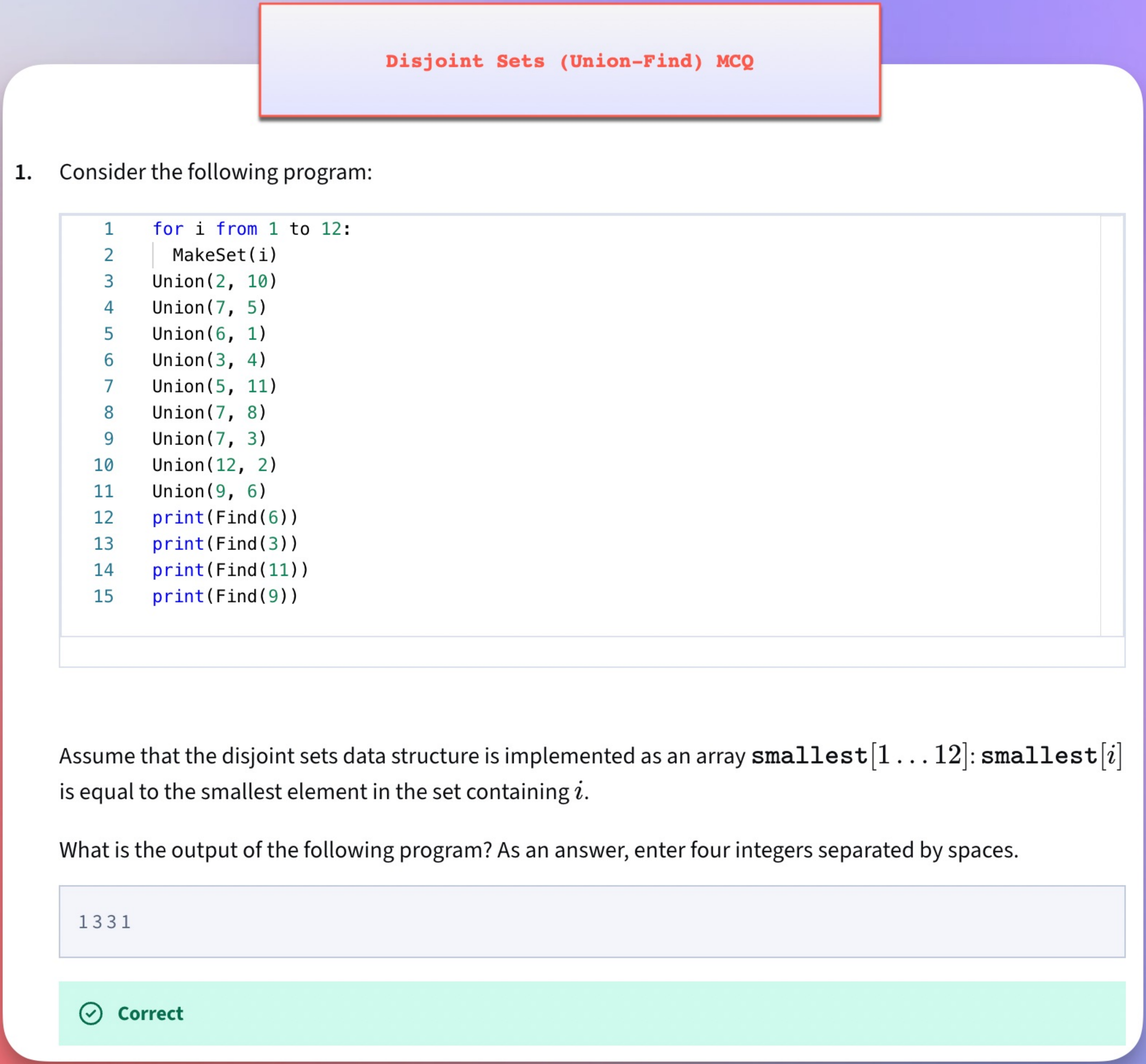


Disjoint Sets (Union-Find) MCQ

1. Consider the following program:

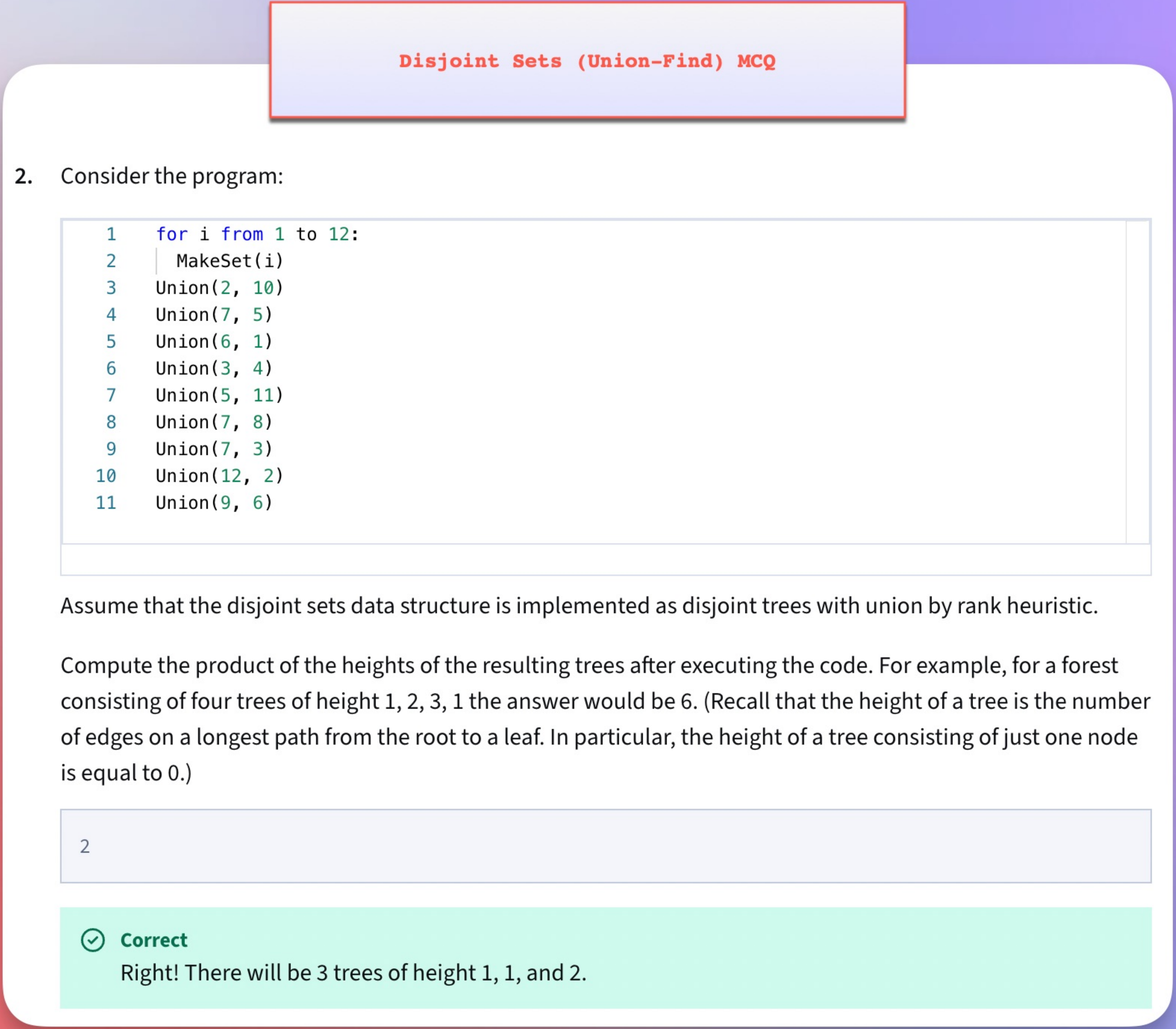
```
for i from 1 to 12:
  MakeSet(i)
Union(2, 10)
Union(7, 5)
Union(6, 1)
Union(3, 4)
Union(5, 11)
Union(7, 8)
Union(7, 3)
Union(12, 2)
Union(9, 6)
print(Find(6))
print(Find(3))
print(Find(11))
print(Find(9))
```

Assume that the disjoint sets data structure is implemented as an array $\texttt{smallest}[1\ldots12]$: $\texttt{smallest}[i]$ is equal to the smallest element in the set containing i.

What is the output of the following program? As an answer, enter four integers separated by spaces.

```
1 3 3 1
```

Correct



Disjoint Sets (Union-Find) MCQ

2. Consider the program:

```
for i from 1 to 12:
    MakeSet(i)
Union(2, 10)
Union(7, 5)
Union(6, 1)
Union(3, 4)
Union(5, 11)
Union(7, 8)
Union(7, 3)
Union(12, 2)
Union(9, 6)
```

Assume that the disjoint sets data structure is implemented as disjoint trees with union by rank heuristic.

Compute the product of the heights of the resulting trees after executing the code. For example, for a forest consisting of four trees of height 1, 2, 3, 1 the answer would be 6. (Recall that the height of a tree is the number of edges on a longest path from the root to a leaf. In particular, the height of a tree consisting of just one node is equal to 0.)

2

Correct
Right! There will be 3 trees of height 1, 1, and 2.

Disjoint Sets (Union-Find) MCQ

3. Consider the following program:

```
for i from 1 to n:
  MakeSet(i)
for i from 1 to n-1:
  Union(i, i+1)
```

Assume that the disjoint sets data structure is implemented as disjoint trees with union by rank heuristic.

What is the number of trees in the forest and the maximum height of a tree in this forest after executing this code? (Recall that the height of a tree is the number of edges on a longest path from the root to a leaf. In particular, the height of a tree consisting of just one node is equal to 0.)

- One tree of height $\log_2 n$.
- n trees, the maximum height is 1.
- Two trees, both of height 1.
- $\log_2 n$ trees, the maximum height is 1.
- $n/2$ trees, the maximum height is 2.
- [✓ One tree of height 1.]

Correct

Disjoint Sets (Union-Find) MCQ

4. Consider the following program:

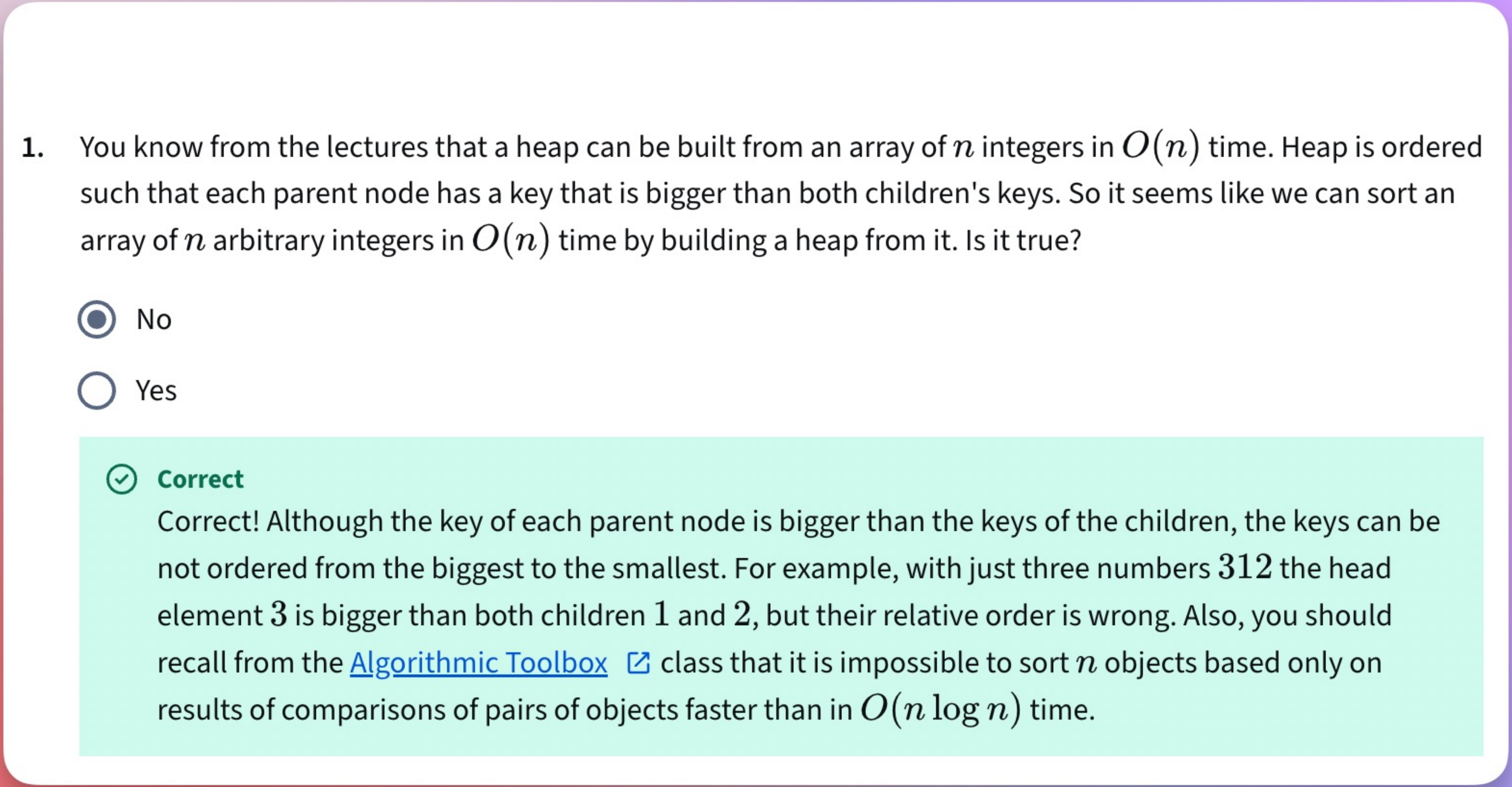
```
for i from 1 to 60:
  MakeSet(i)
for i from 1 to 30:
  Union(i, 2*i)
for i from 1 to 20:
  Union(i, 3*i)
for i from 1 to 12:
  Union(i, 5*i)
for i from 1 to 60:
  Find(i)
```

Assume that the disjoint sets data structure is implemented as disjoint trees with union by rank heuristic and with path compression heuristic.

Compute the maximum height of a tree in the resulting forest. (Recall that the height of a tree is the number of edges on a longest path from the root to a leaf. In particular, the height of a tree consisting of just one node is equal to 0.)

1

Correct
There is at least one tree of height 1 in the forest. Also, all trees have height at most 1, since the last for-loop calls Find() for all 60 elements. Since path compression is used, each non-root node will be attached directly to the corresponding root in this loop, and hence all the trees will have height at most 1.



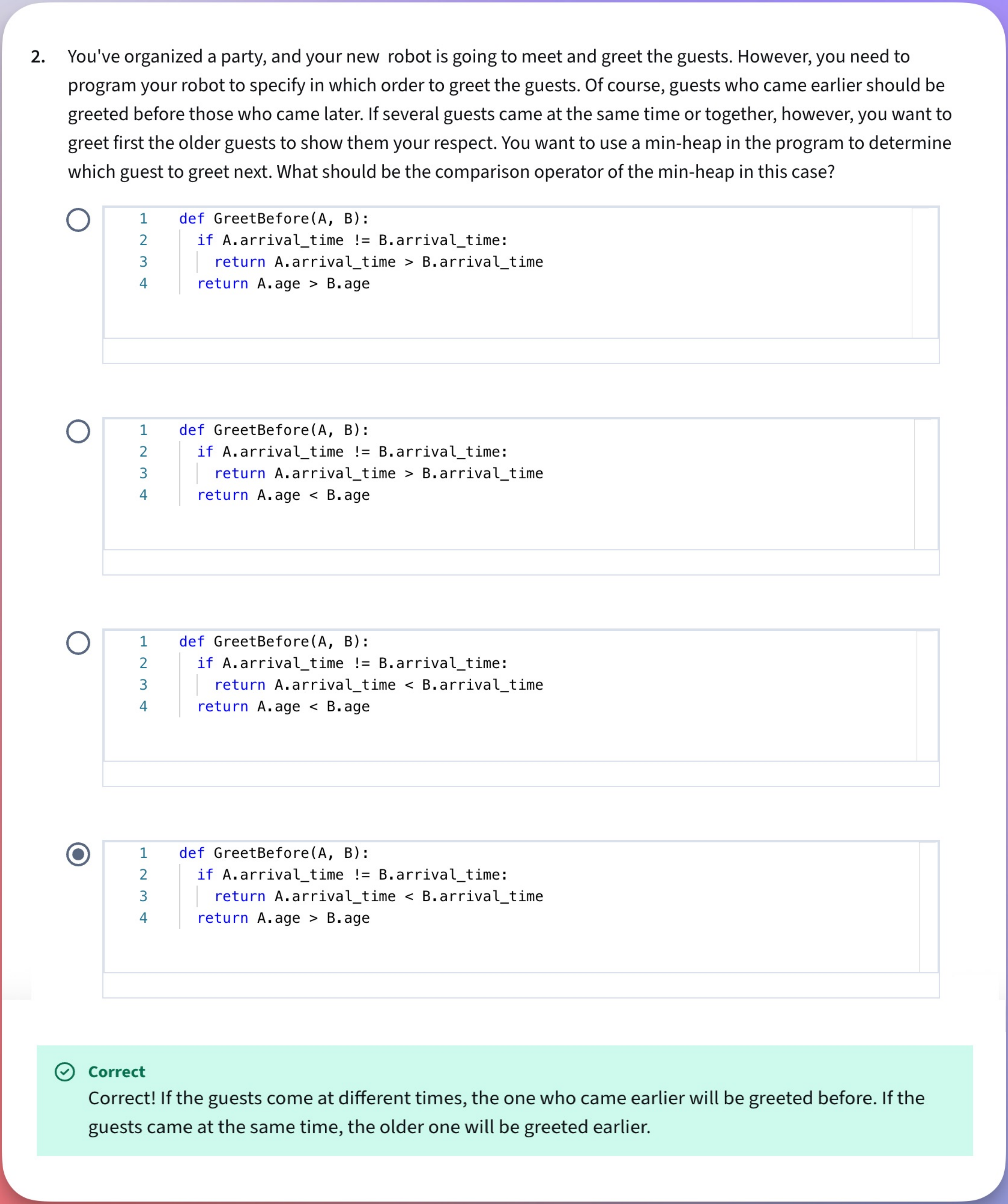
1. You know from the lectures that a heap can be built from an array of n integers in $O(n)$ time. Heap is ordered such that each parent node has a key that is bigger than both children's keys. So it seems like we can sort an array of n arbitrary integers in $O(n)$ time by building a heap from it. Is it true?

   [✓ No]

   Yes

   Correct

   Correct! Although the key of each parent node is bigger than the keys of the children, the keys can be not ordered from the biggest to the smallest. For example, with just three numbers 312 the head element 3 is bigger than both children 1 and 2, but their relative order is wrong. Also, you should recall from the Algorithmic Toolbox class that it is impossible to sort n objects based only on results of comparisons of pairs of objects faster than in $O(n \log n)$ time.



2. You've organized a party, and your new robot is going to meet and greet the guests. However, you need to program your robot to specify in which order to greet the guests. Of course, guests who came earlier should be greeted before those who came later. If several guests came at the same time or together, however, you want to greet first the older guests to show them your respect. You want to use a min-heap in the program to determine which guest to greet next. What should be the comparison operator of the min-heap in this case?

```
def GreetBefore(A, B):
  if A.arrival_time != B.arrival_time:
    return A.arrival_time > B.arrival_time
  return A.age > B.age
```

```
def GreetBefore(A, B):
  if A.arrival_time != B.arrival_time:
    return A.arrival_time > B.arrival_time
  return A.age < B.age
```

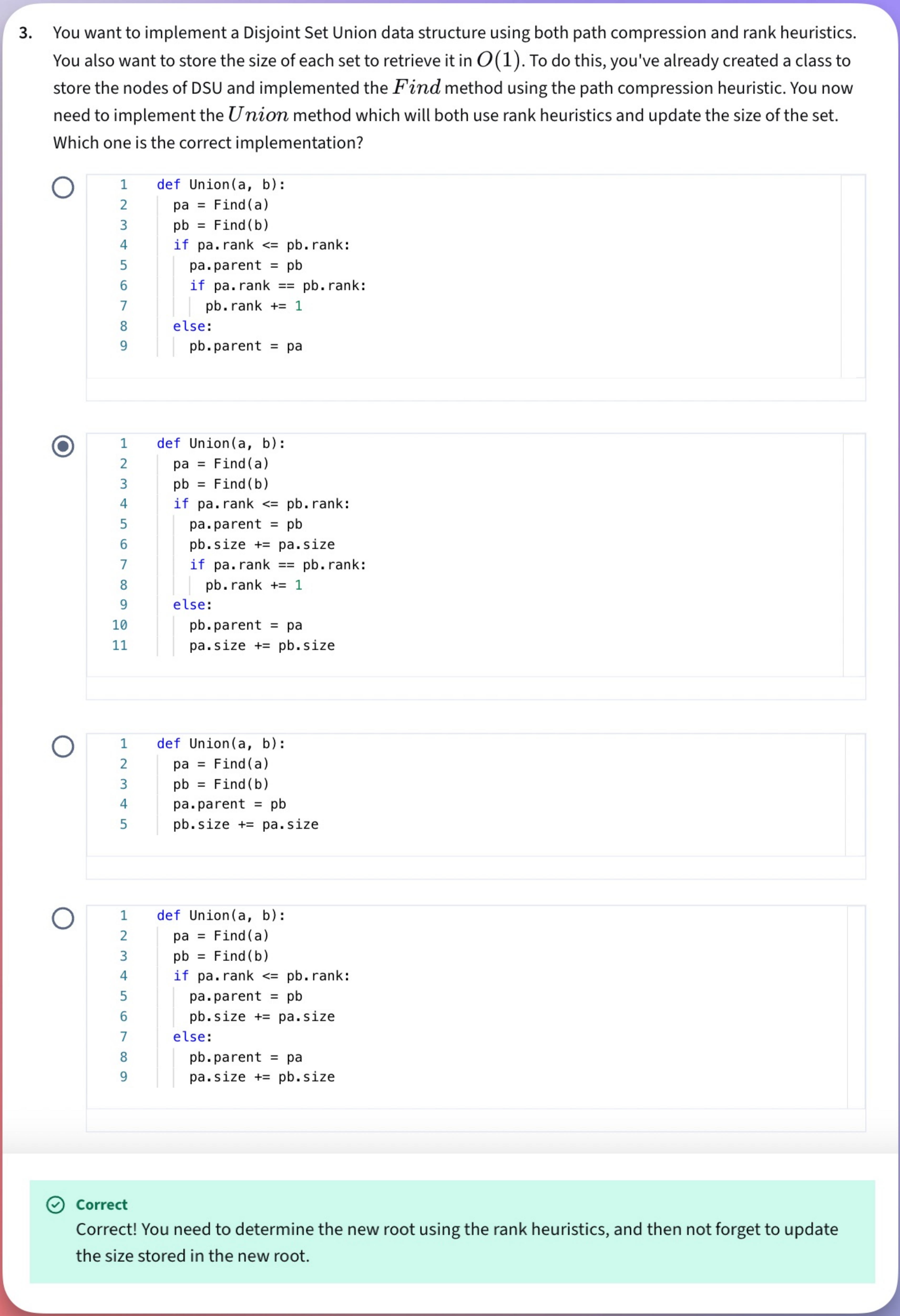
```
def GreetBefore(A, B):
  if A.arrival_time != B.arrival_time:
    return A.arrival_time < B.arrival_time
  return A.age < B.age
```

[✓]

```
def GreetBefore(A, B):
  if A.arrival_time != B.arrival_time:
    return A.arrival_time < B.arrival_time
  return A.age > B.age
```

Correct

Correct! If the guests come at different times, the one who came earlier will be greeted before. If the guests came at the same time, the older one will be greeted earlier.



3. You want to implement a Disjoint Set Union data structure using both path compression and rank heuristics. You also want to store the size of each set to retrieve it in $O(1)$. To do this, you've already created a class to store the nodes of DSU and implemented the $Find$ method using the path compression heuristic. You now need to implement the $Union$ method which will both use rank heuristics and update the size of the set. Which one is the correct implementation?

```
def Union(a, b):
  pa = Find(a)
  pb = Find(b)
  if pa.rank <= pb.rank:
    pa.parent = pb
    if pa.rank == pb.rank:
      pb.rank += 1
  else:
    pb.parent = pa
```

[✓

```
def Union(a, b):
  pa = Find(a)
  pb = Find(b)
  if pa.rank <= pb.rank:
    pa.parent = pb
    pb.size += pa.size
    if pa.rank == pb.rank:
      pb.rank += 1
  else:
    pb.parent = pa
    pa.size += pb.size
```

]

```
def Union(a, b):
  pa = Find(a)
  pb = Find(b)
  pa.parent = pb
  pb.size += pa.size
```

```
def Union(a, b):
  pa = Find(a)
  pb = Find(b)
  if pa.rank <= pb.rank:
    pa.parent = pb
    pb.size += pa.size
  else:
    pb.parent = pa
    pa.size += pb.size
```

✓ **Correct**

Correct! You need to determine the new root using the rank heuristics, and then not forget to update the size stored in the new root.