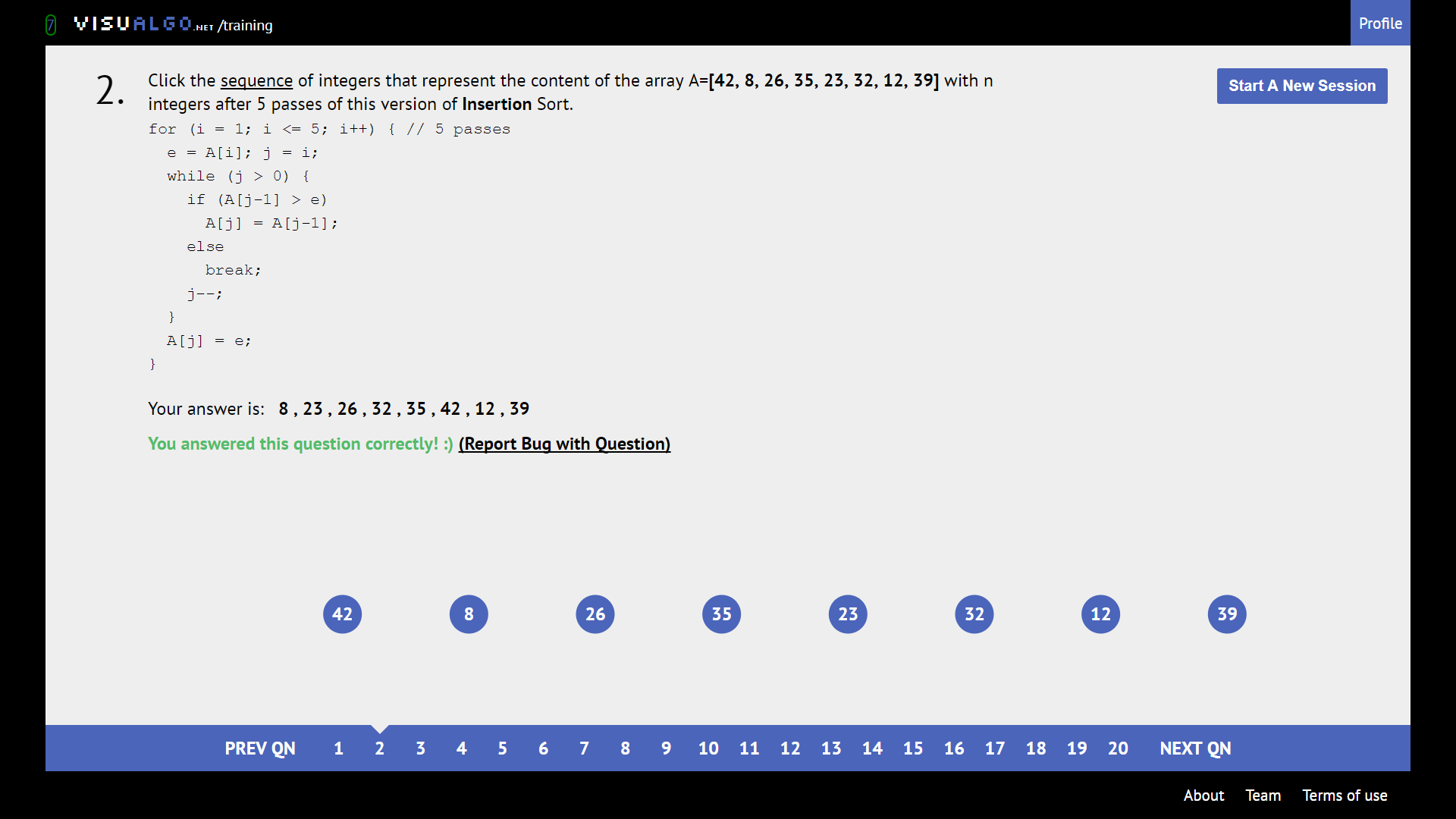
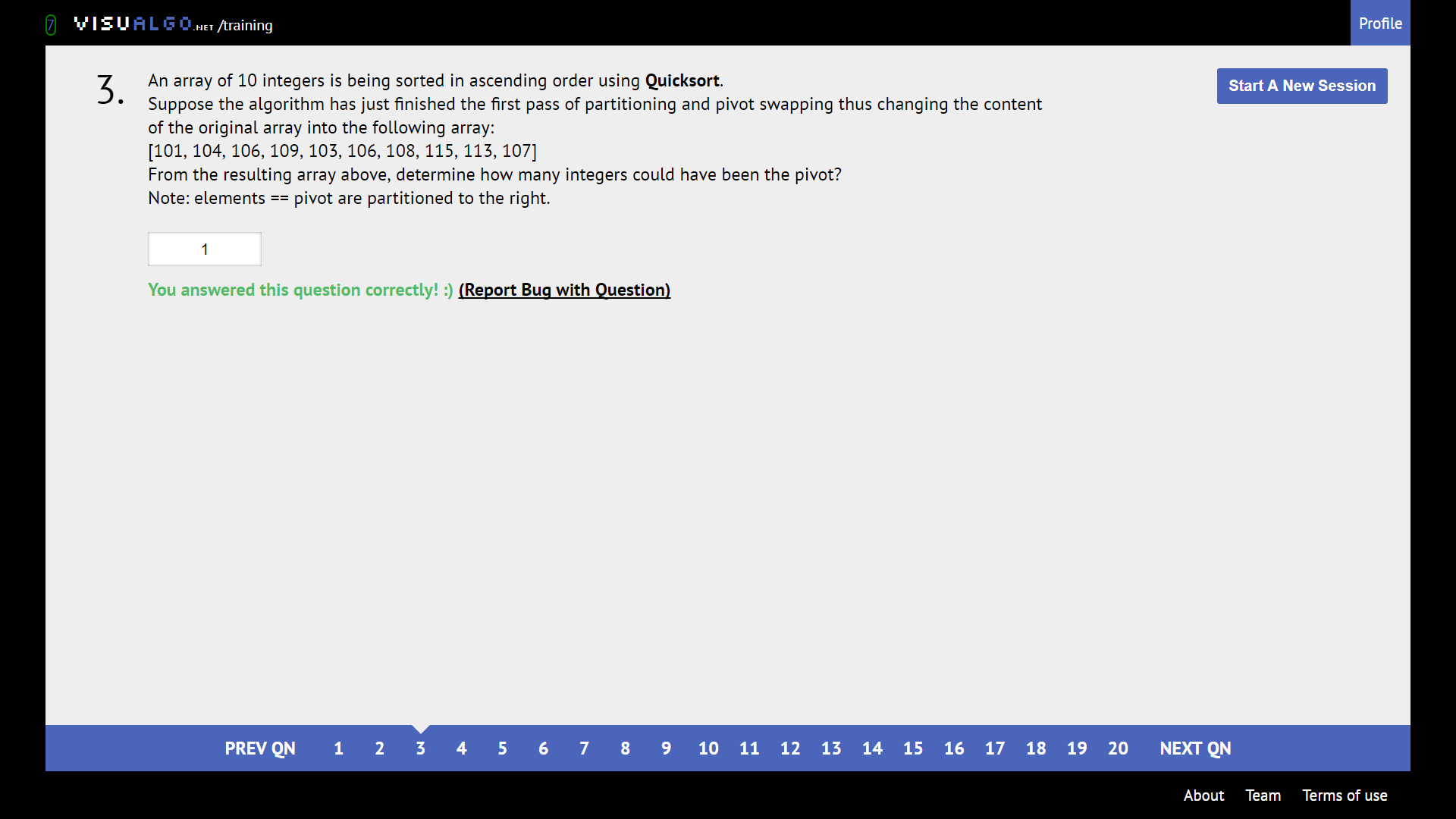
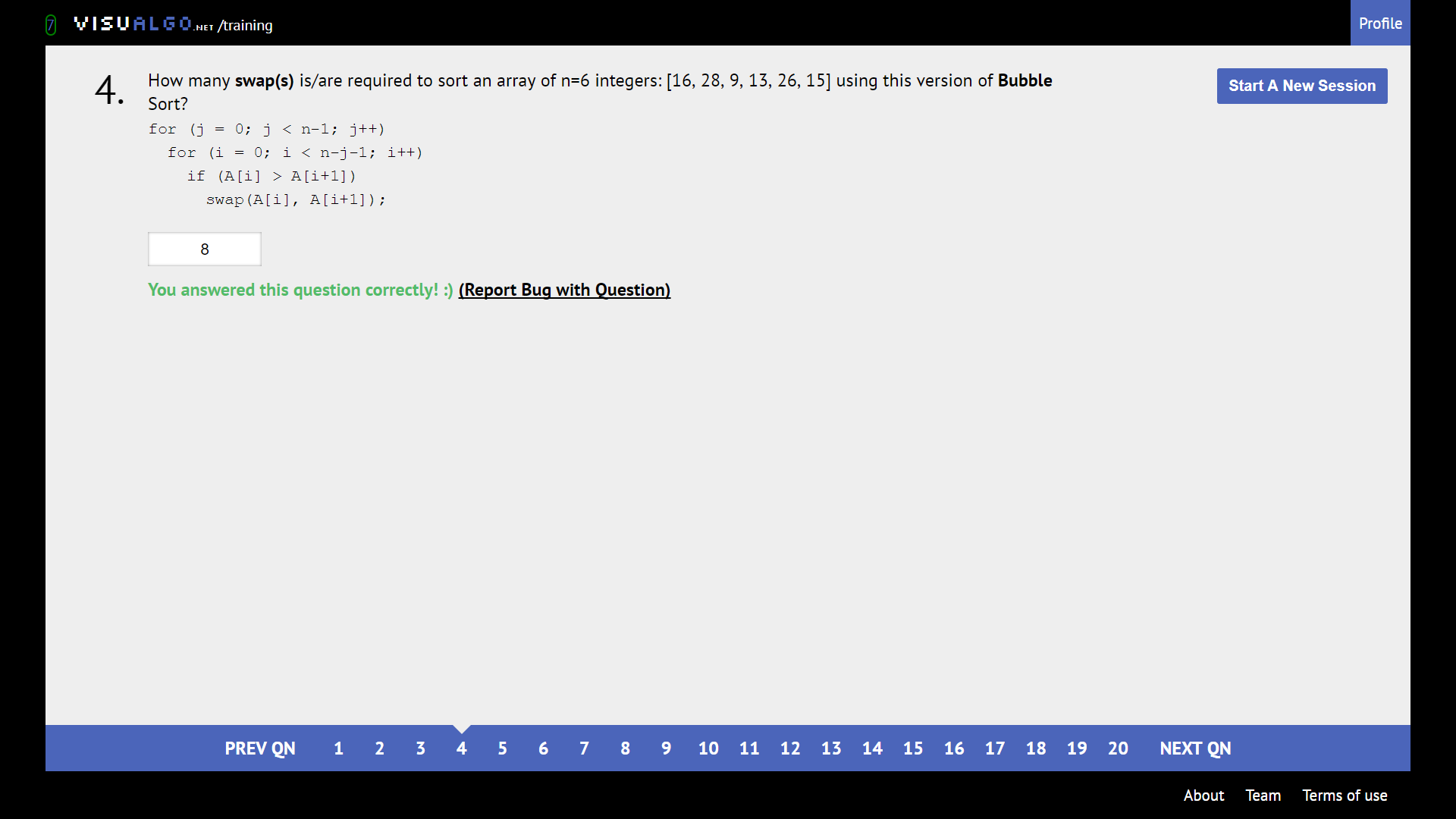
**Sorting**



First n+1 numbers will be sorted. Insertion sort is STABLE, so the rest of the elements is unaffected.



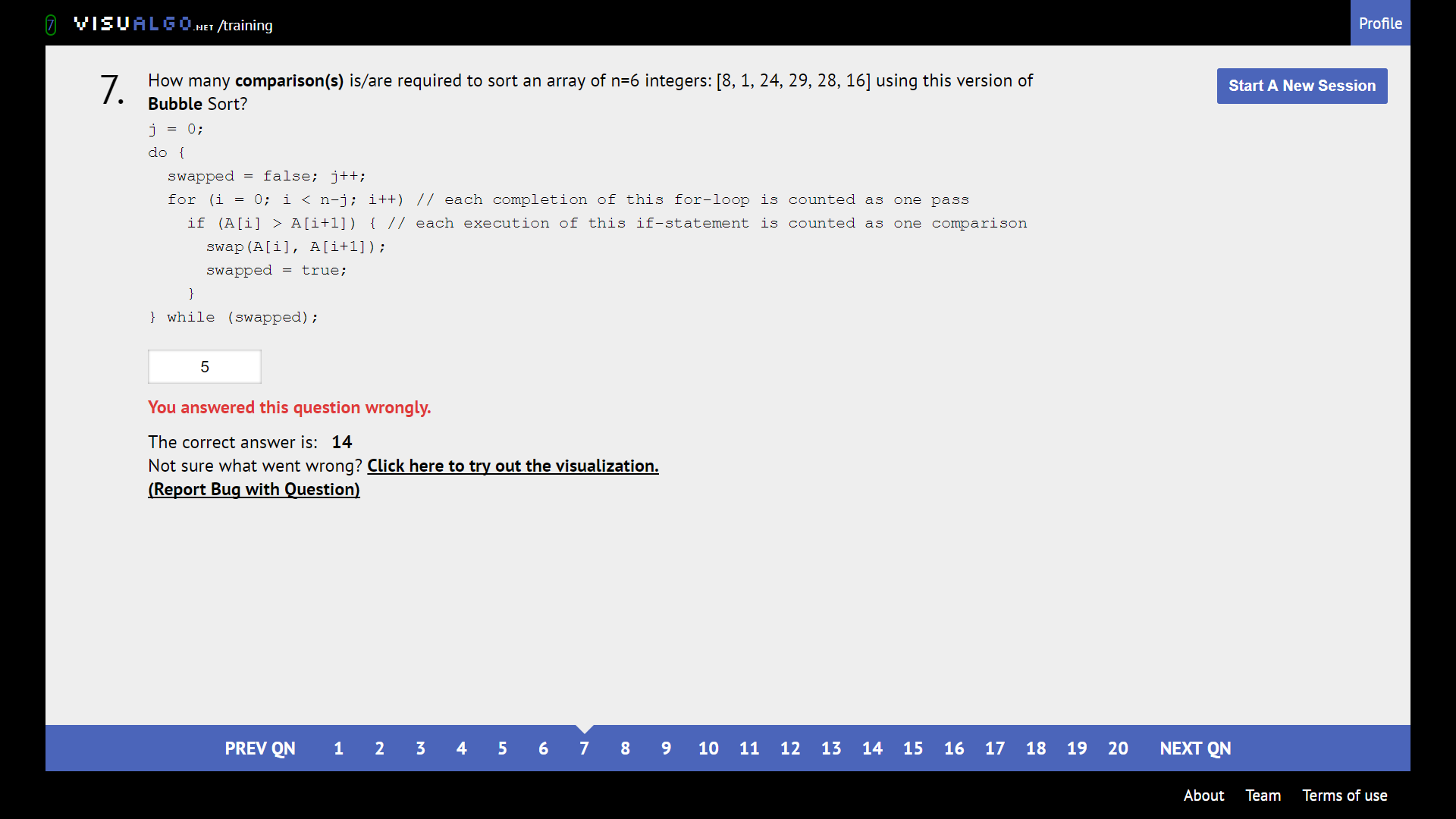
Find integer that will be greater than all to its left and smaller than all to its right

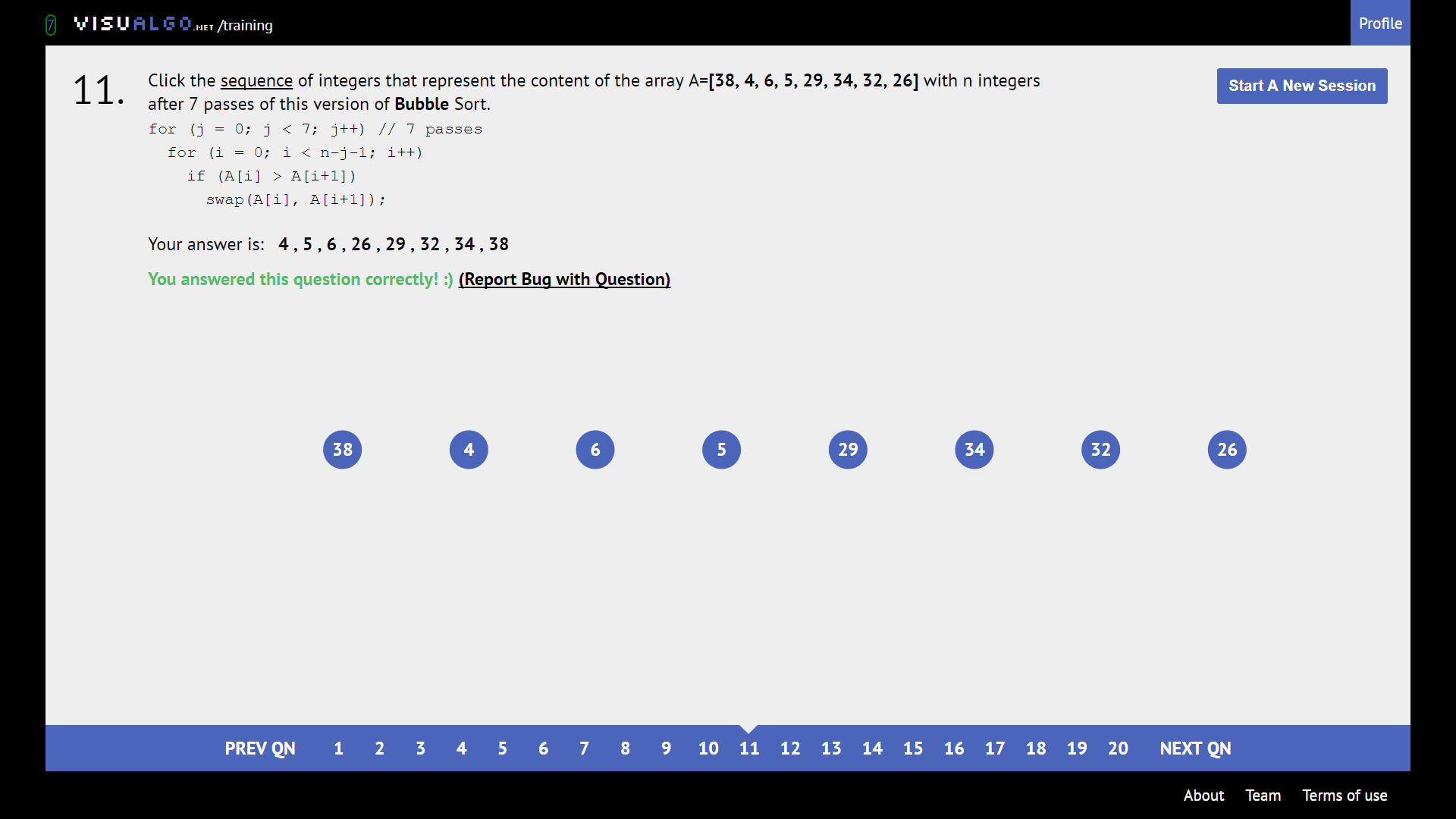


Brute force count swaps

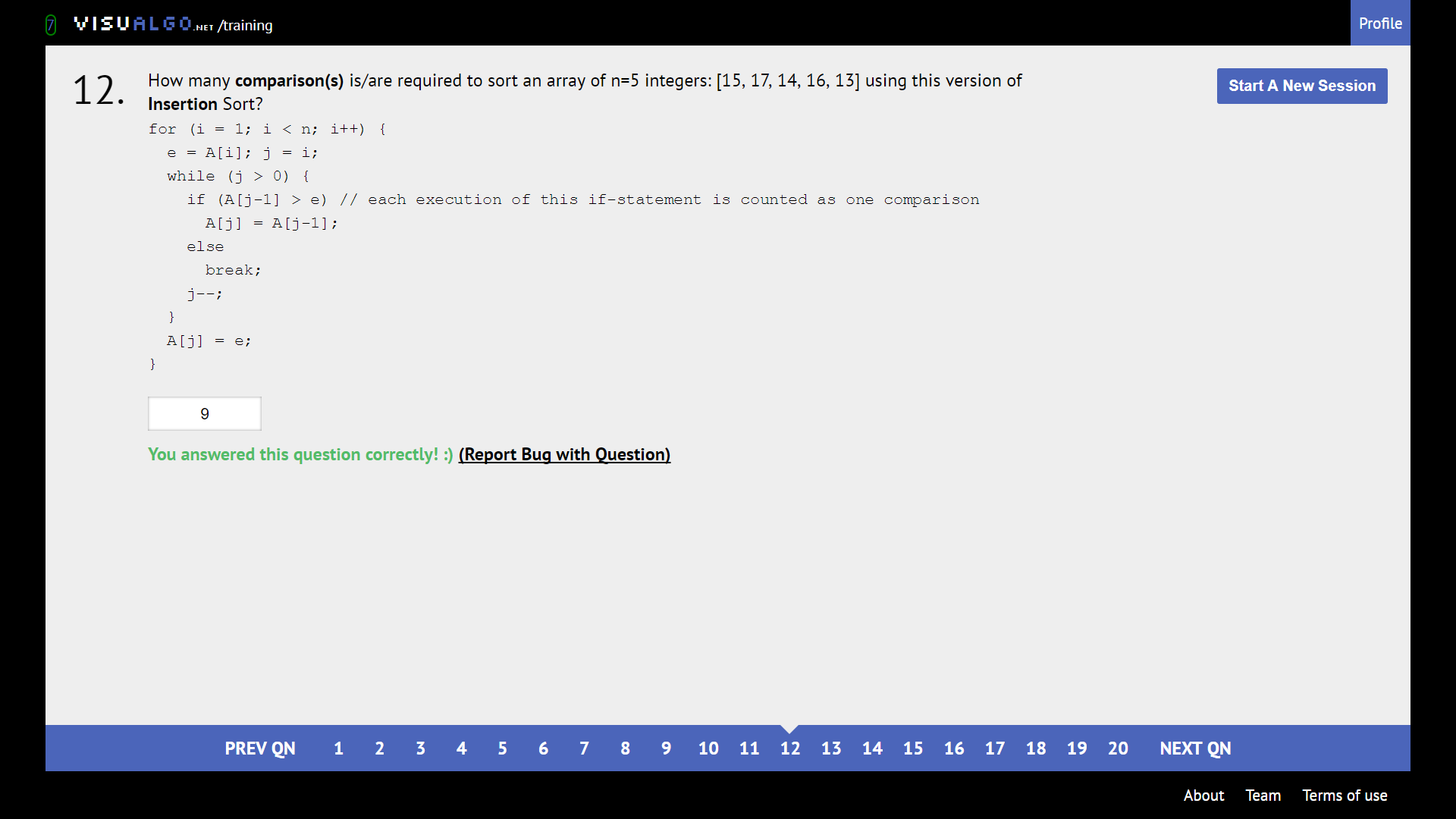


Selection sort is UNSTABLE. Keep track of where elements are swapped to.

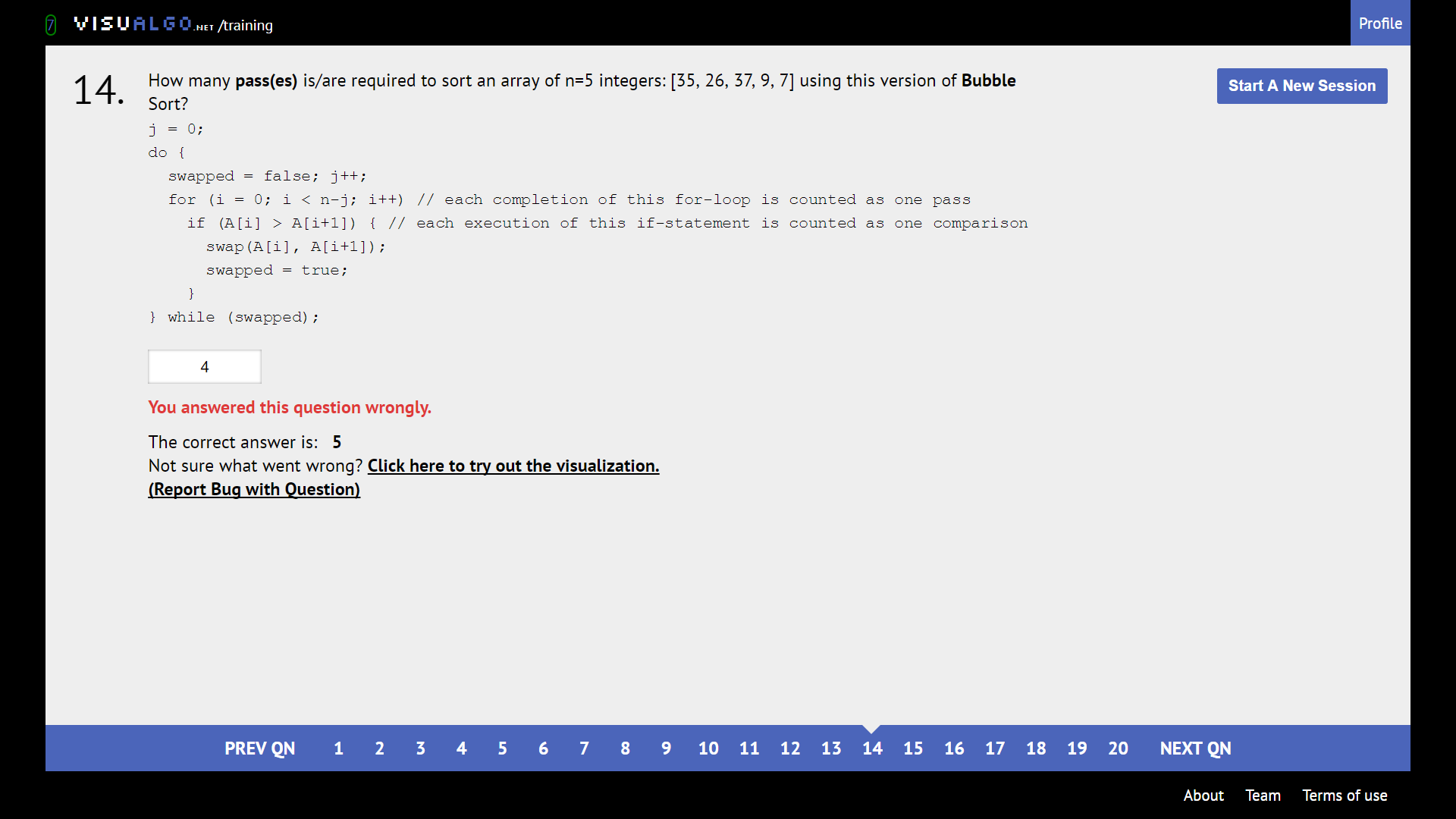




Biggest 7 elements are bubbled right. So, this sequence of 8 integers will be fully sorted lol

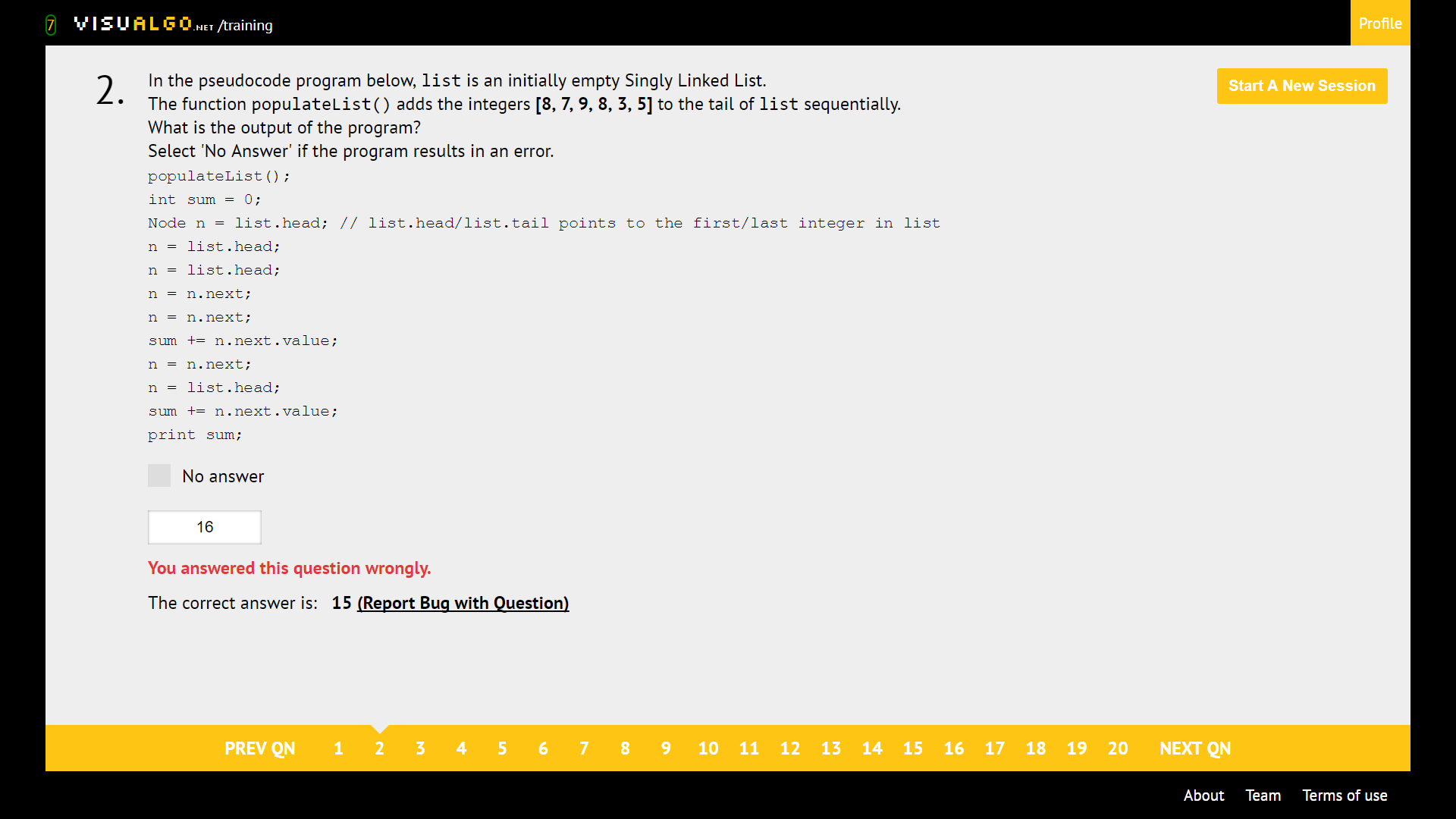


Become the algorithm

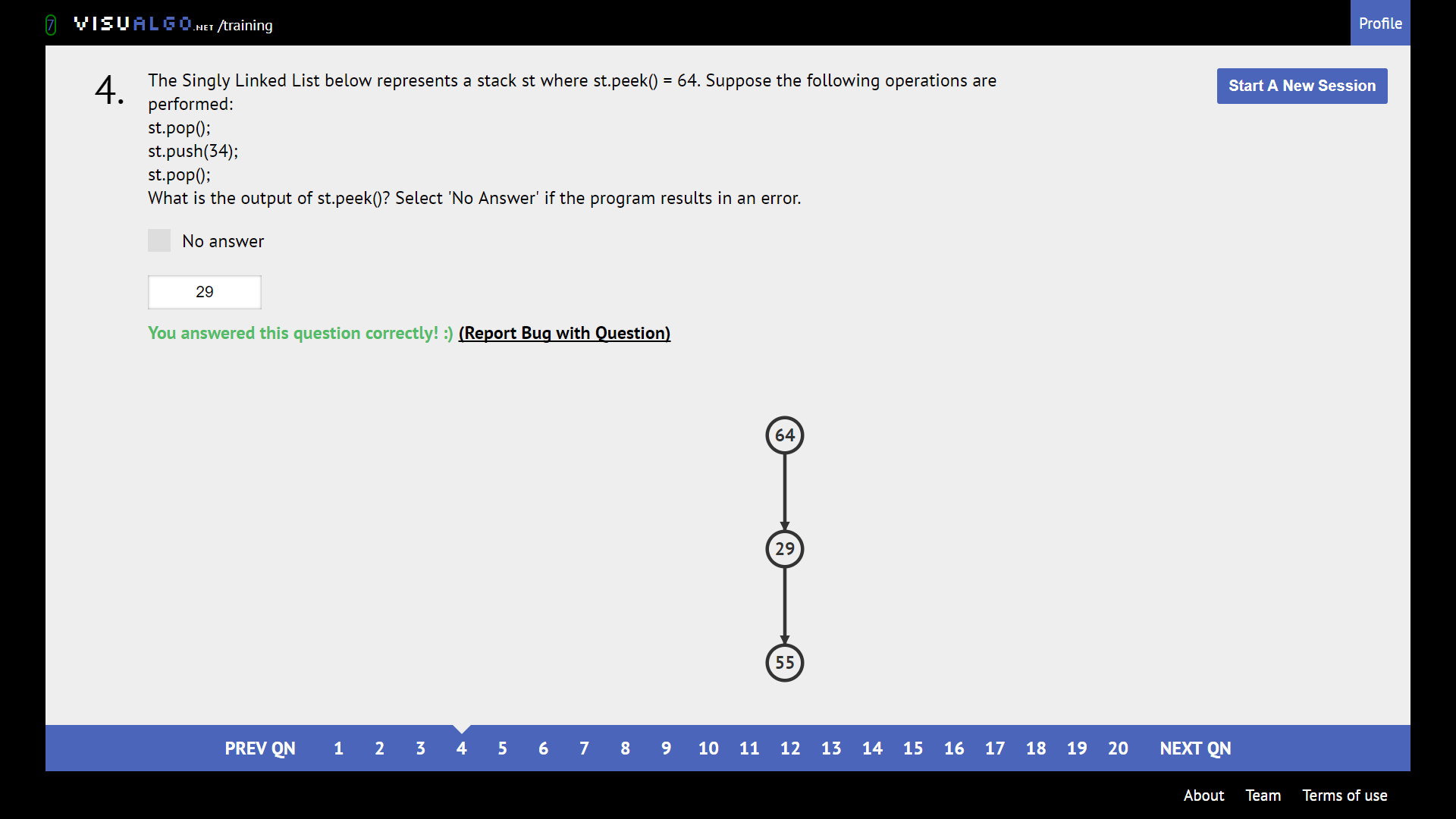


Become the algorithm. This is the “smart” version which breaks if no new swaps are made in the pass-through.

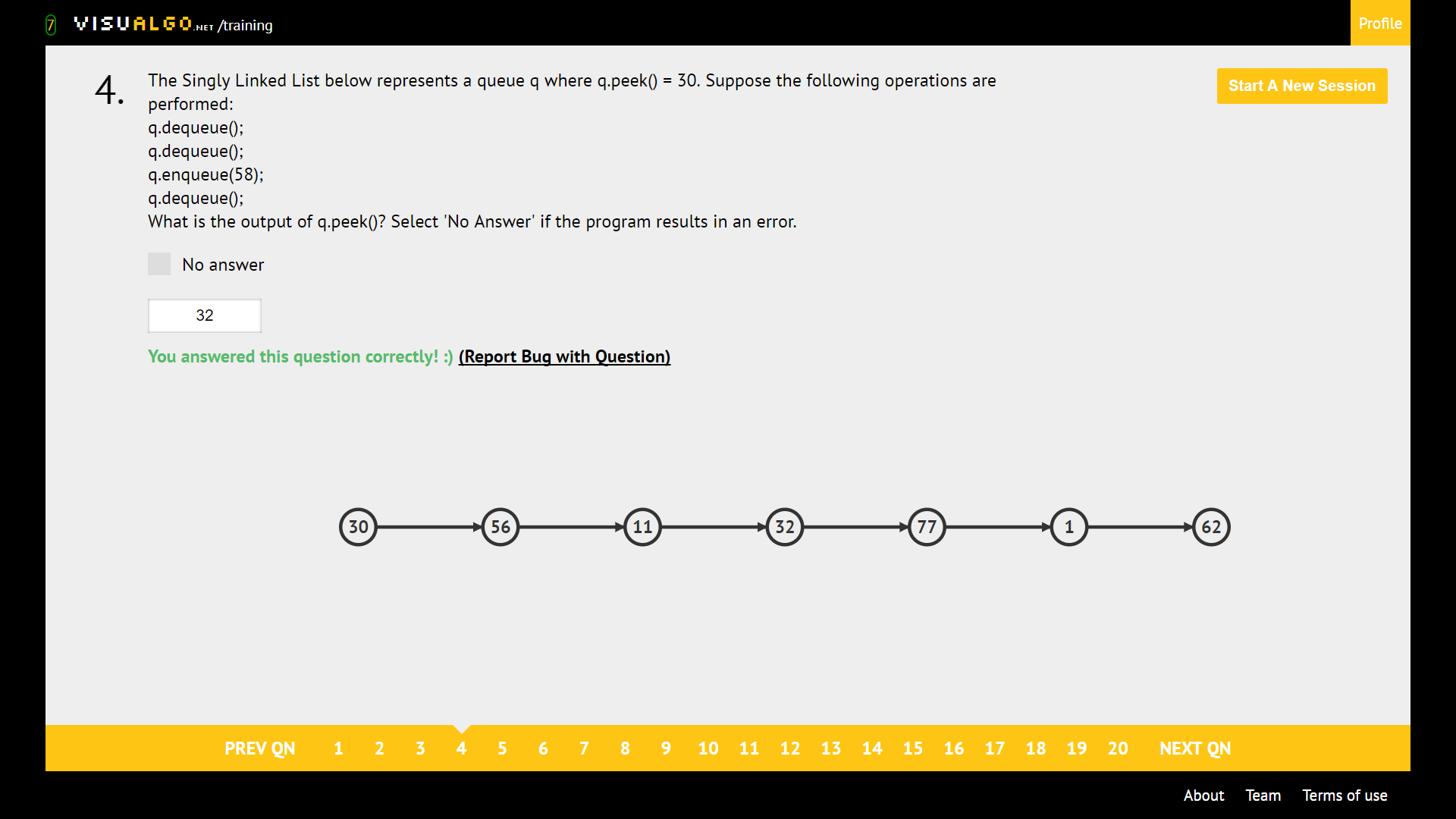
**Linked list**

****

Just be careful. Tail pointer is usually not updated.



Add and remove stuff from the TOP, just like tray stacking



Enqueue adds at the RIGHT, removes from the LEFT

**Binary Heap**

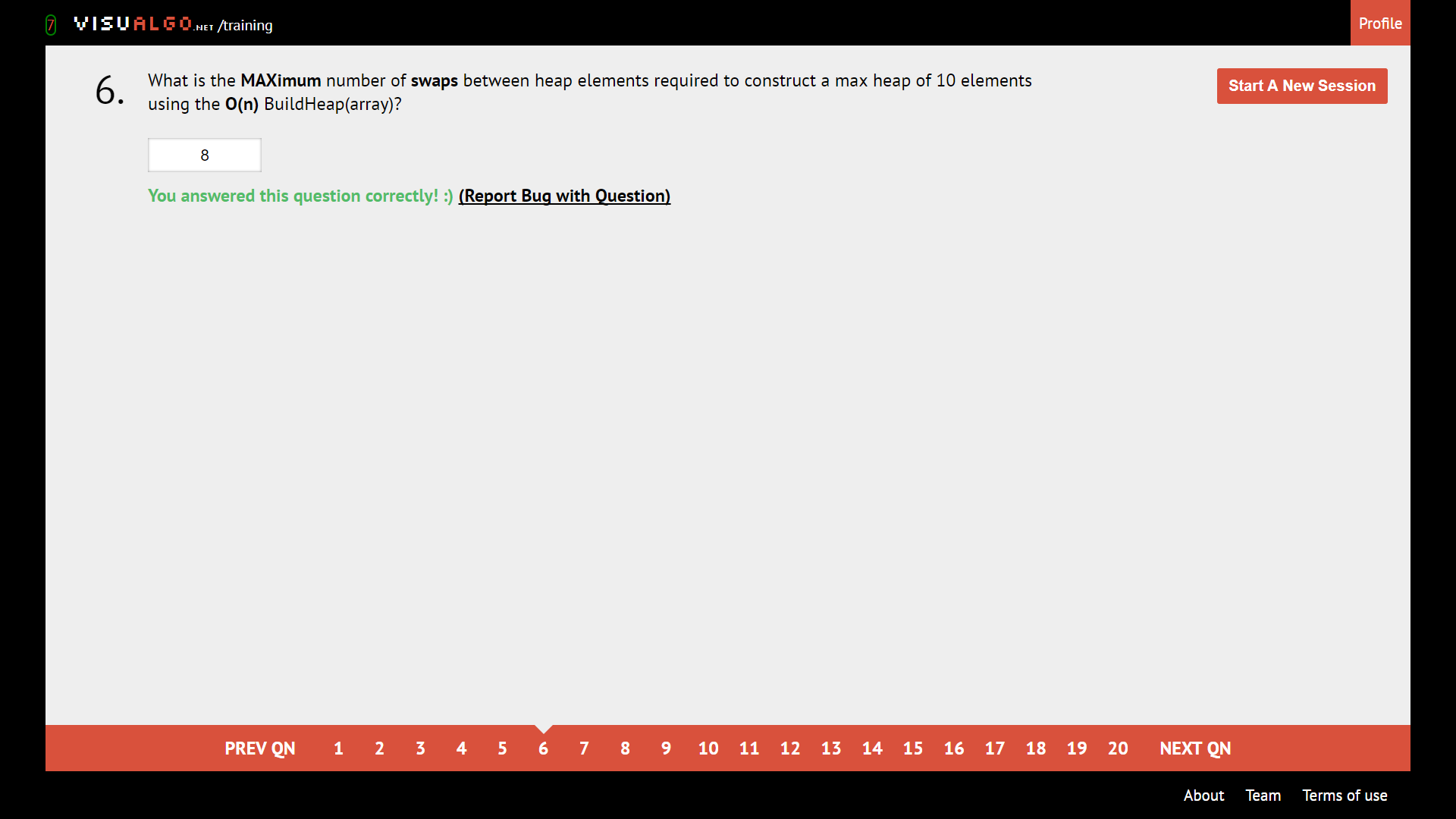
Valid Max Heap: Make sure parent bigger than child



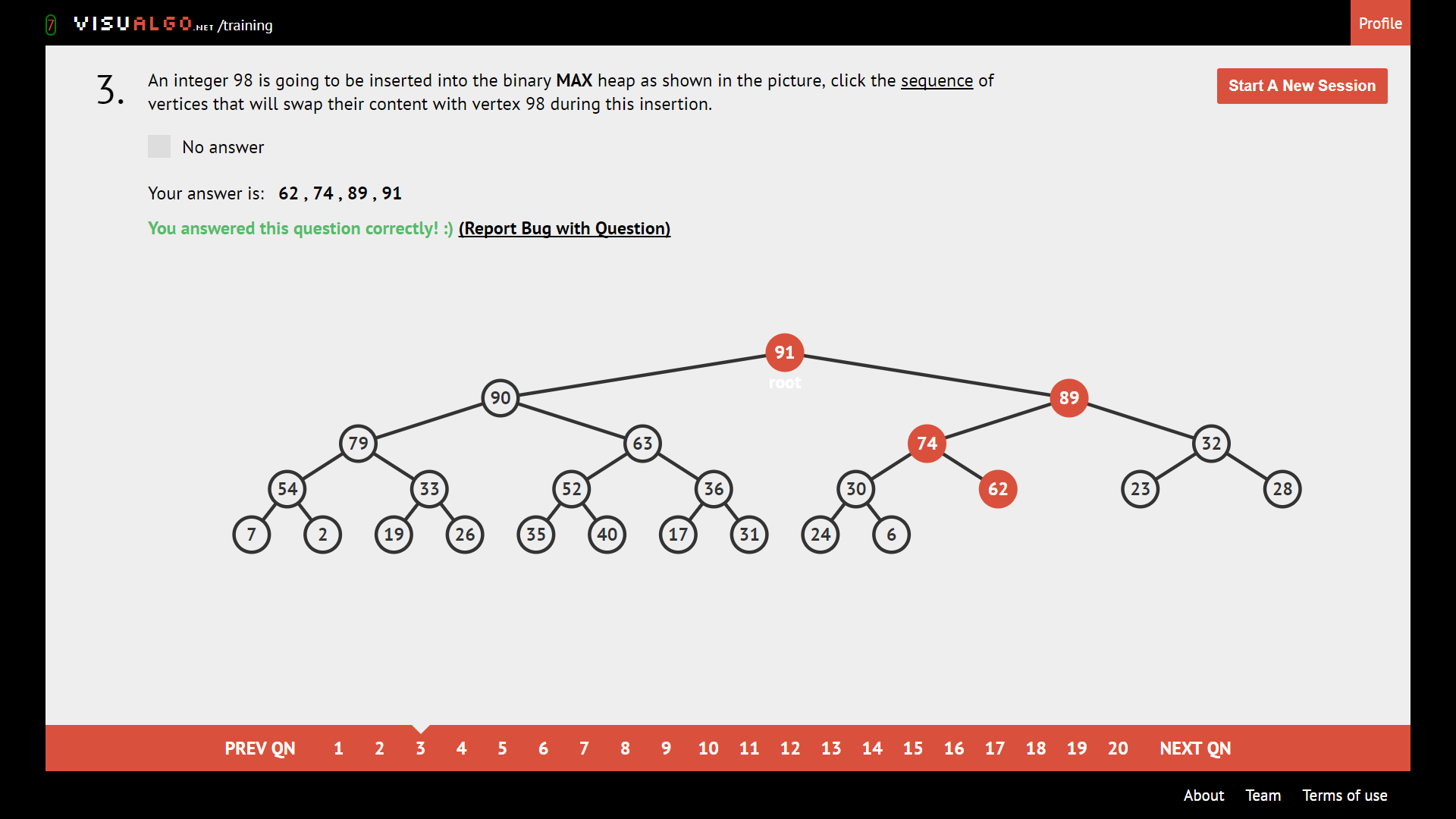
Refer to chart



Refer to chart

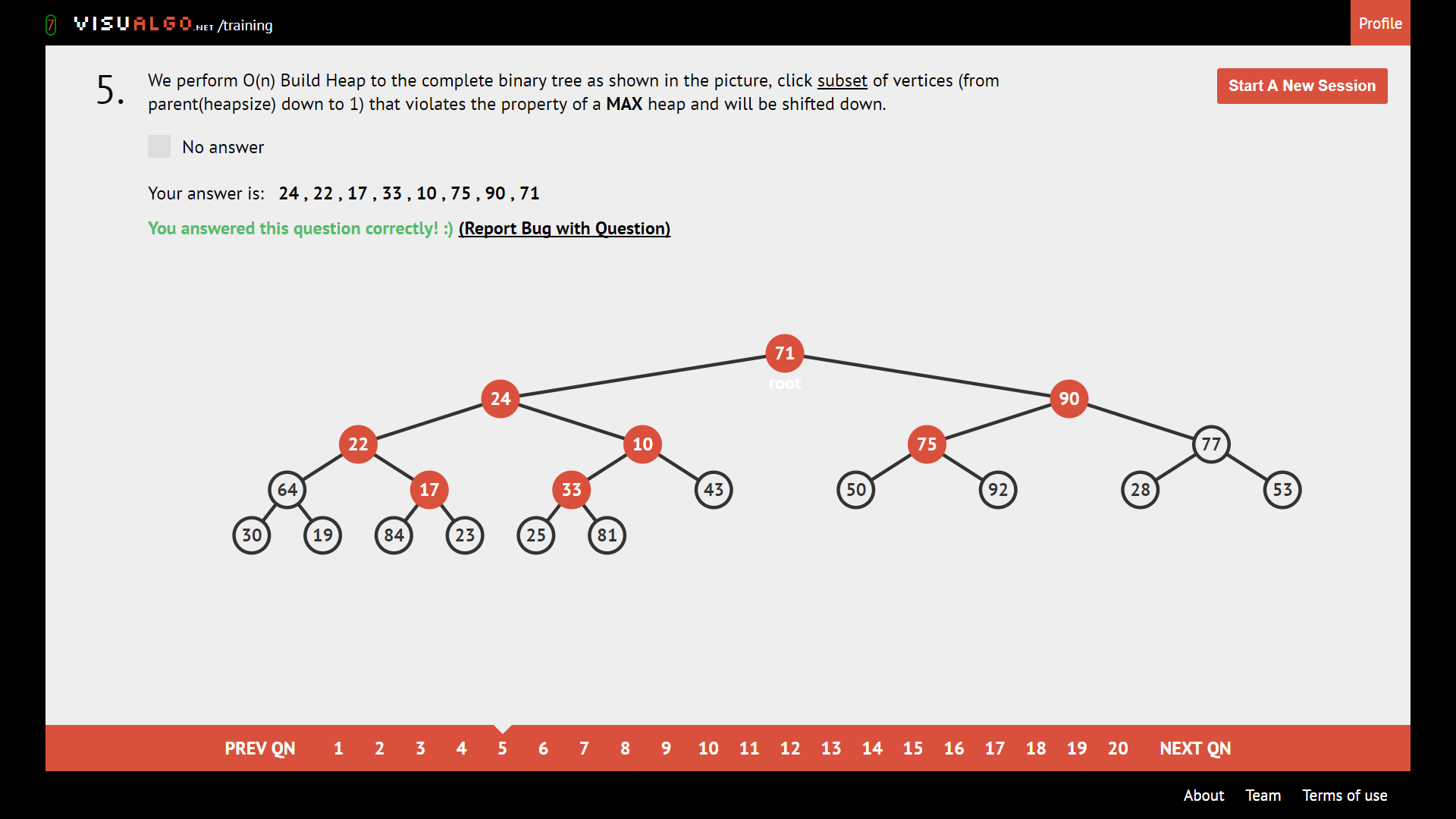


Refer to chart.



Add to bottom-most and left-most available spot. Bubble it up.

Violates the property of a MAX heap

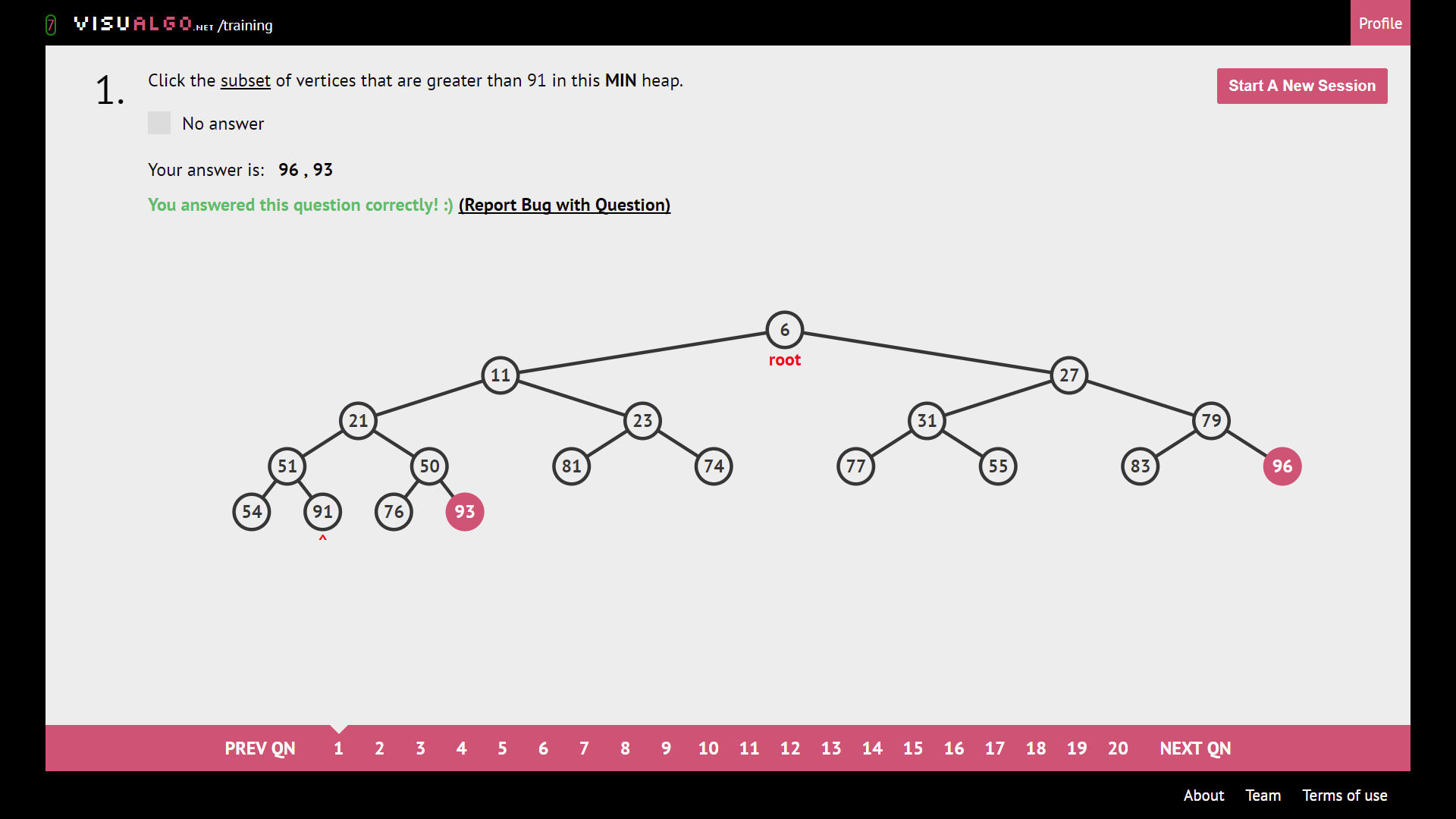


Vertices that are smaller than any of their descendants will be shifted down.

Remain in the binary MAX heap after extraction



Smallest 17 – 13 = 4 elements are remaining



Find all vertex bigger than number. This q is dumb.

**True or False:**

Commutative

Graphical user interface, text, application, email

Description automatically generated

Calling ShiftDown(i) ∀i > heapsize/2

Graphical user interface, text, application

Description automatically generated

The third largest element in a Binary Max Heap that contains > 3 distinct integers is always one of the children of the root?

Graphical user interface, text

Description automatically generated with medium confidence

leaves

A screenshot of a computer

Description automatically generated with medium confidence

Click on all internal vertices

Ans: click on **all the roots of subtrees accept main root**

Chart, line chart, scatter chart

Description automatically generated

**UFDS**

What is the root of the resulting set?

Ans: Always take the set with the **greatest height**

Diagram

Description automatically generated

Click the subset of vertexes such that UFDS structure changes from path compression

Ans: always pick the **vertexes that are not 1 distance/weight away from the root**

A picture containing chart

Description automatically generated

Given n = number, what is the appropriate height given h = number?

Ans: if 2^h < n 🡪 true, n is what they give you, 2^h is what you compute below

Else 2^h > n 🡪 false

**Bitmasking/UFDS tallest tree**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| h | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| N = 2^h | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 |

Given n = number, is it possible to call unionSet(i, j) and/or findSet(i) operations to get a single tree of h = number?

Ans: given h = number, **if == h,** then we put true as seen below,

Math.floor(Log\_2(11)) = 3.

Graphical user interface, text, application

Description automatically generated

In this case, log\_2(15).floor() = 3 which is != 4, so false.

Graphical user interface, text, application, email

Description automatically generated

Click the subset of vertices that belong to same set

Chart, scatter chart

Description automatically generated with medium confidence

**BST**

BST: Left child < current < Right child.

Successor: Most direct way to the next element bigger than the node of interest

Predecessor: Most direct way to the next element smaller than the node of interest

**Pre**order: **Print first**, then check children

**In**order: Check left, **then print**, then check right

**Post**order: Check children first, **then print.**

**Postorder Ex: left bottom to left top, right bottom to right top, root**

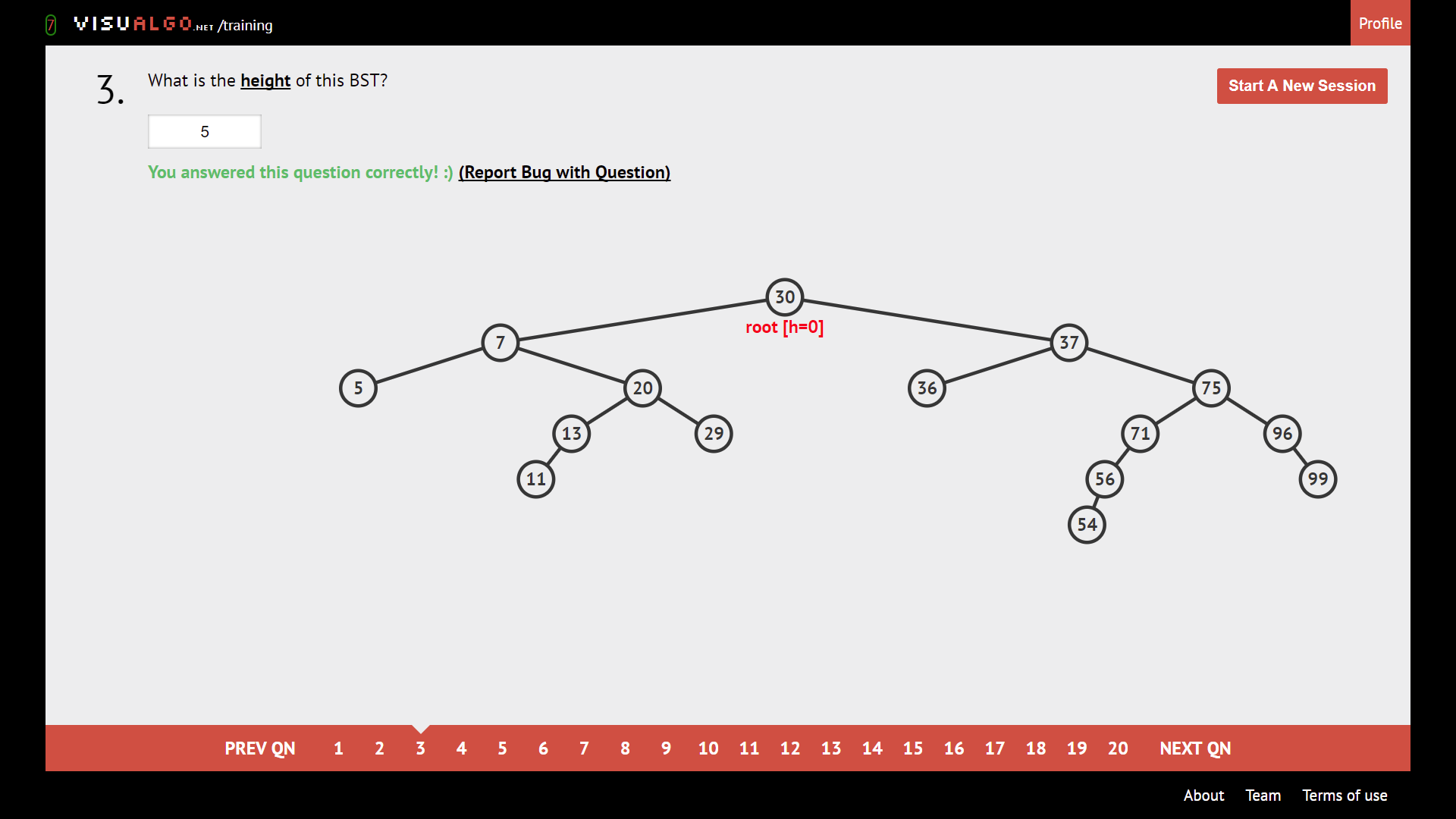
Chart, line chart

Description automatically generated

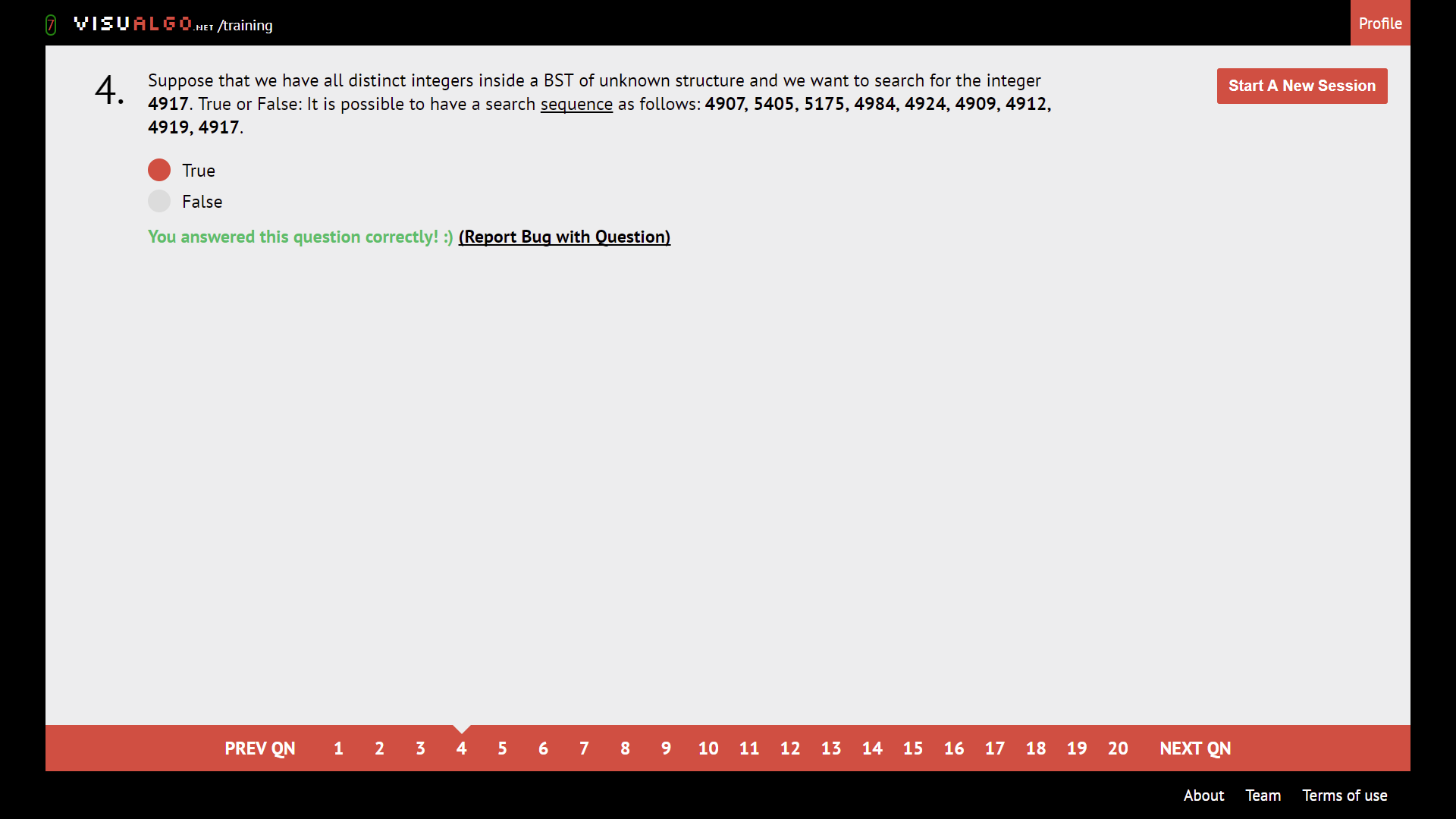
**Preorder Ex: root left all the way, then left subtree then right subtree**

Chart, line chart, scatter chart

Description automatically generated

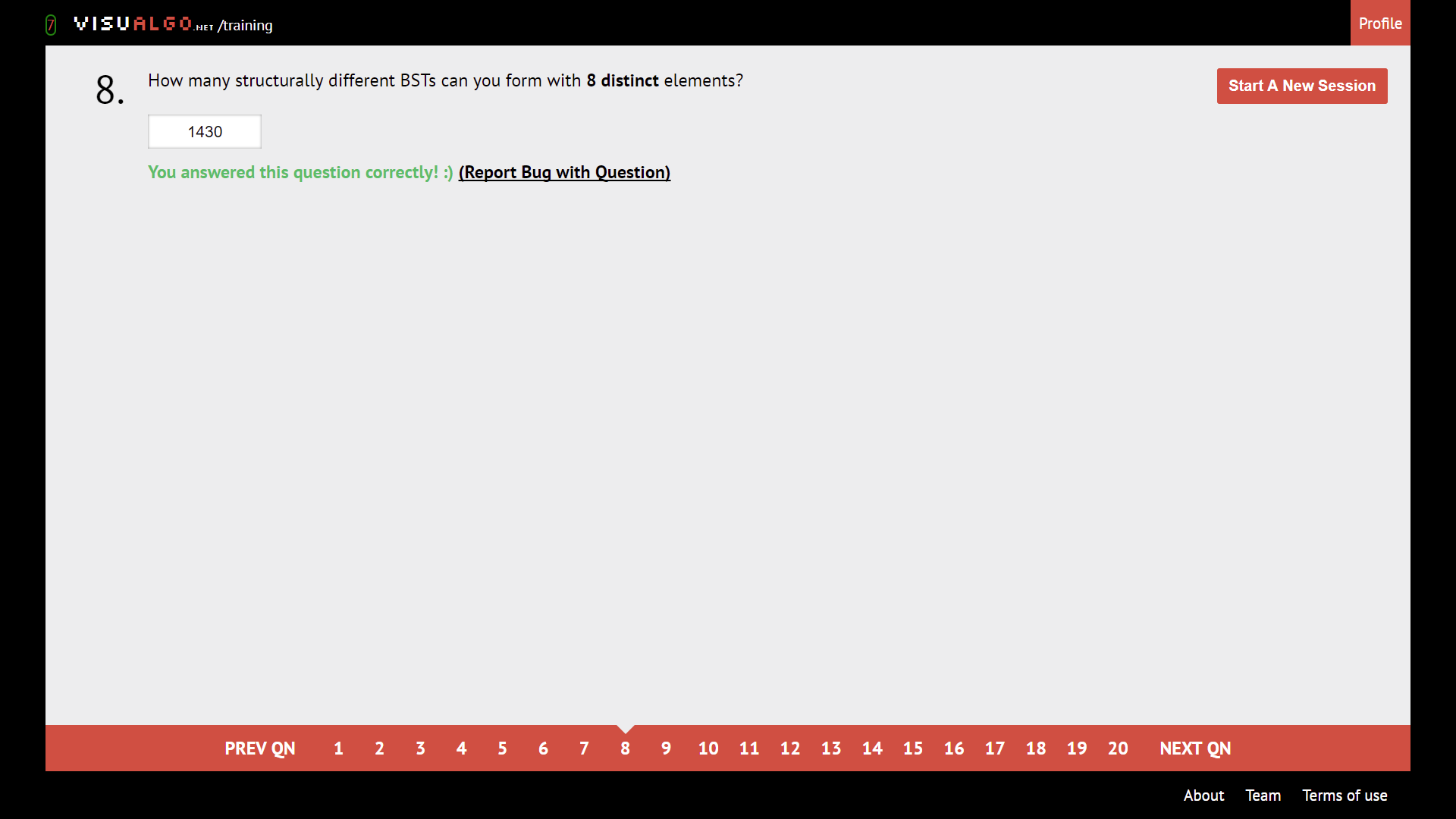


Largest number of edges from root to leaf.

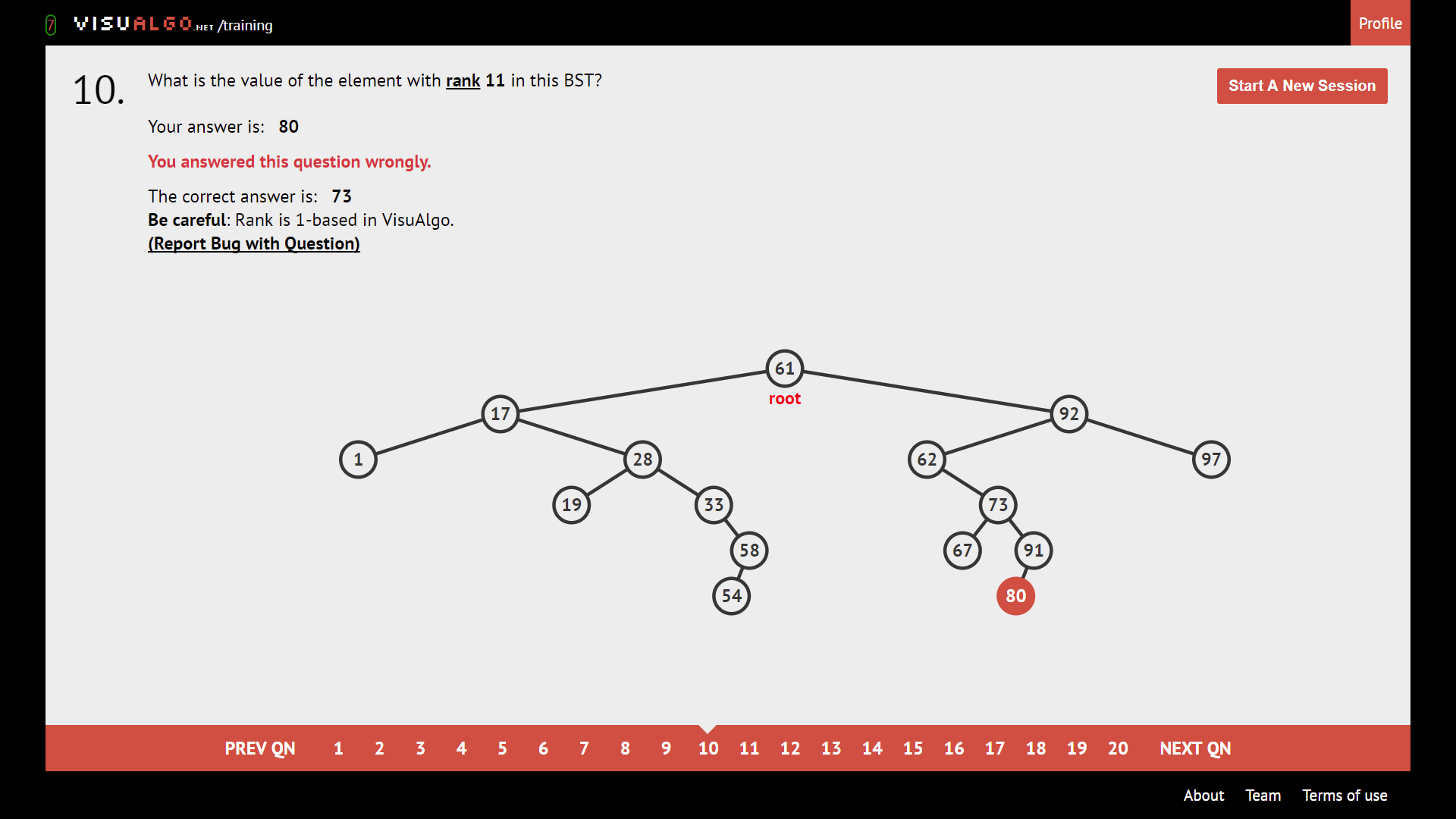


If target is bigger, next number should be bigger; if target smaller, next number should be smaller.

Distinct elements with BSTs

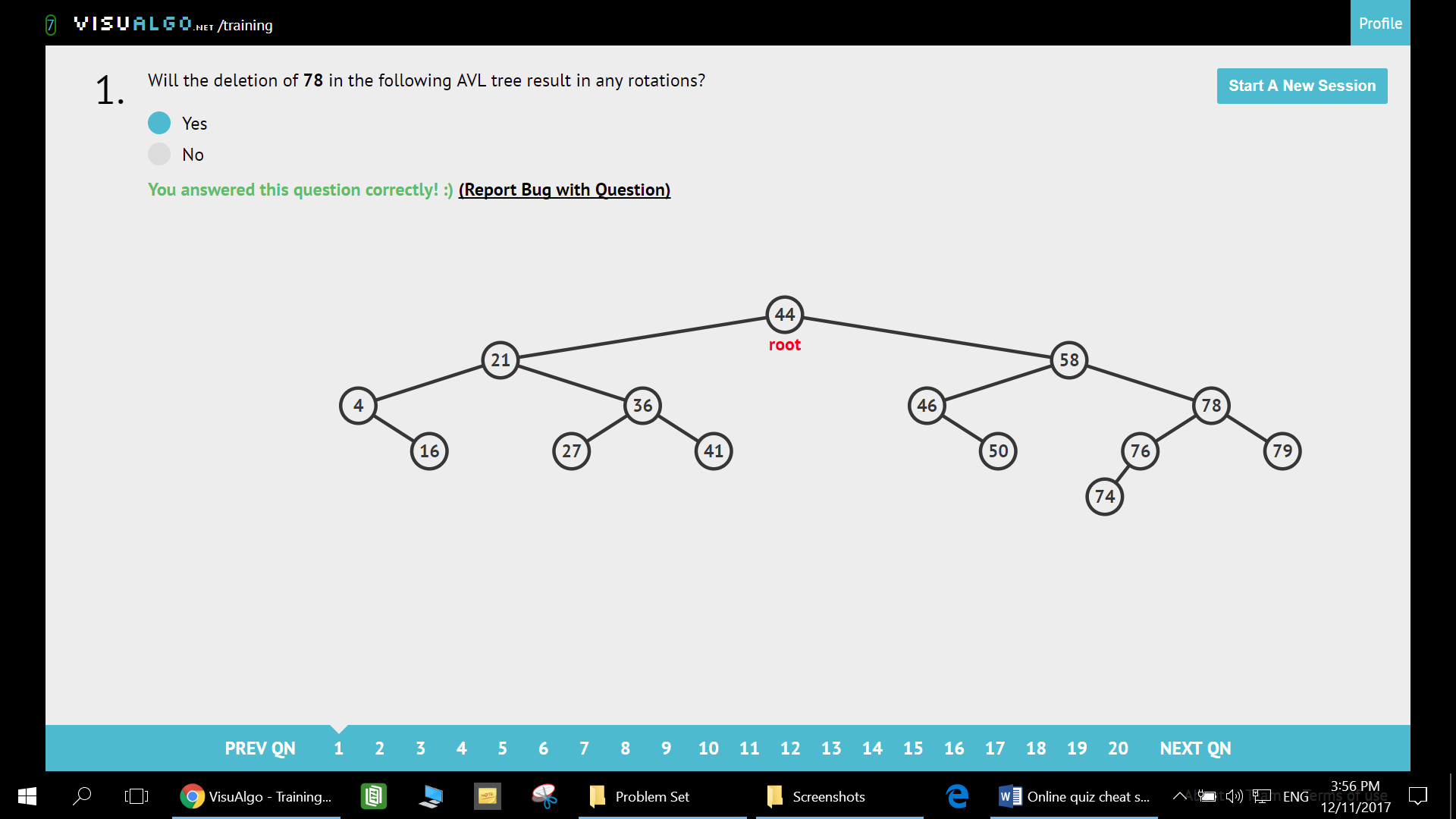


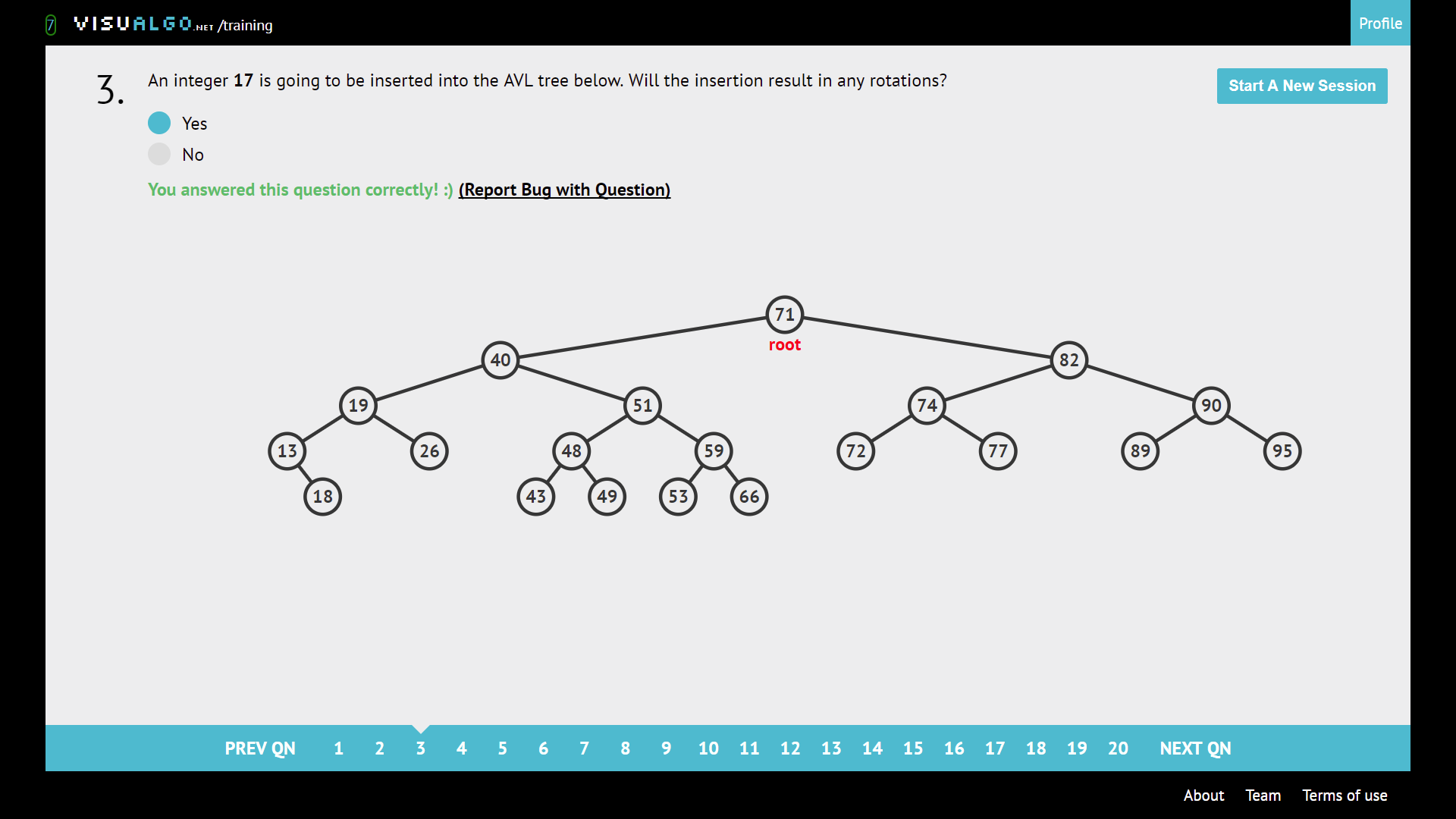
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Elements | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| BSTs | 0 | 1 | 2 | 5 | 14 | 42 | 132 | 429 | 1430 | 4862 | 16796 |



1-indexed. Leftmost is 1, count until rank.

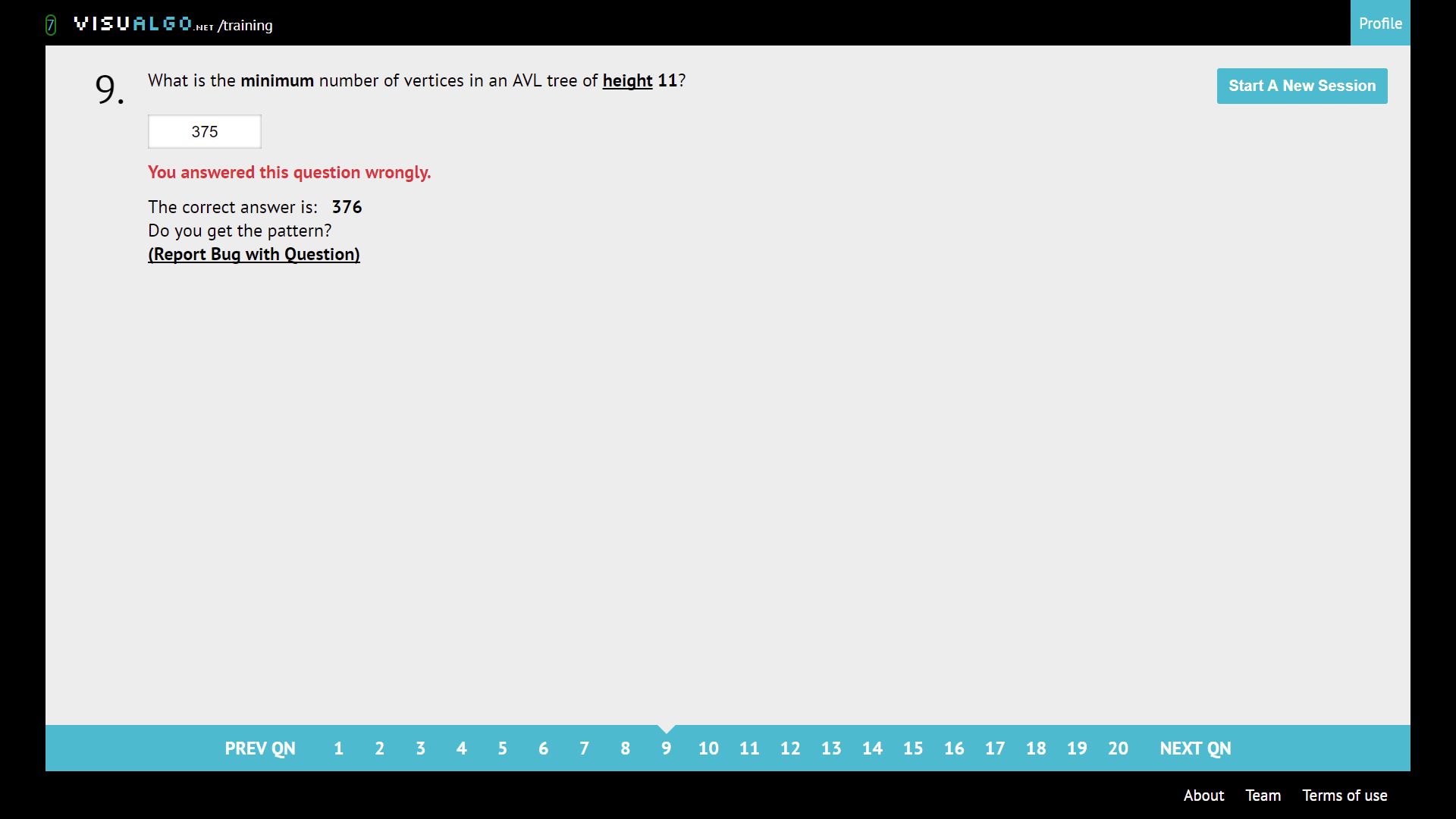
**AVL**



Search upwards starting from the replaced node and see if anyone complains

Insert into correct position and check for rotations.

What is the minimum number of vertices in an AVL tree?



Height(n) = height(n-1) + height(n-2) + 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Height | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Vertices | 1 | 2 | 4 | 7 | 12 | 20 | 33 | 54 | 88 | 143 | 232 | 376 | 609 | 986 | 1596 | 2583 |

^ the chart is available behind also

What is the maximum number of vertices in an AVL tree?

Vertices = (2^(h+1)) – 1

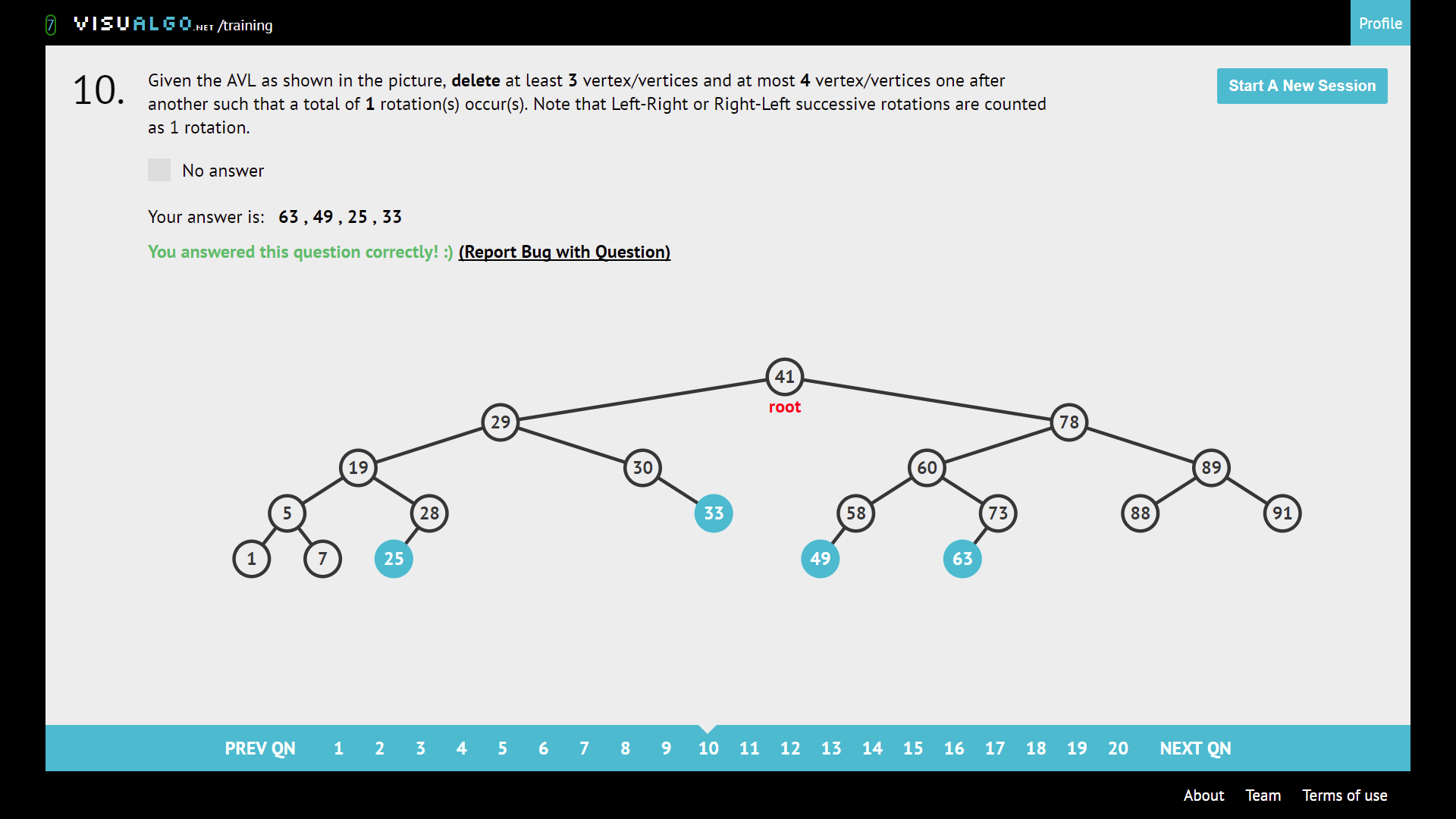
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Height | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Vertices | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 | 1024 | 2047 | 4095 | 8191 |

What is the maximum possible height of a BST with n elements?

Ans: h = n – 1, since BST has no rebalancing, we have O (h = n) traversal time and h = n – 1

What is the maximum possible height of an AVL with n elements?

Ans:



Delete and check for rotation.

Balance factor of vertexes

* Let the chosen vertex be the root, find the height or left subtree – right subtree height

Chart, diagram

Description automatically generated with medium confidence

As seen below, **pick the maximum height of the subtree, given that 21 is the root**

Chart

Description automatically generated

**Graph DS**

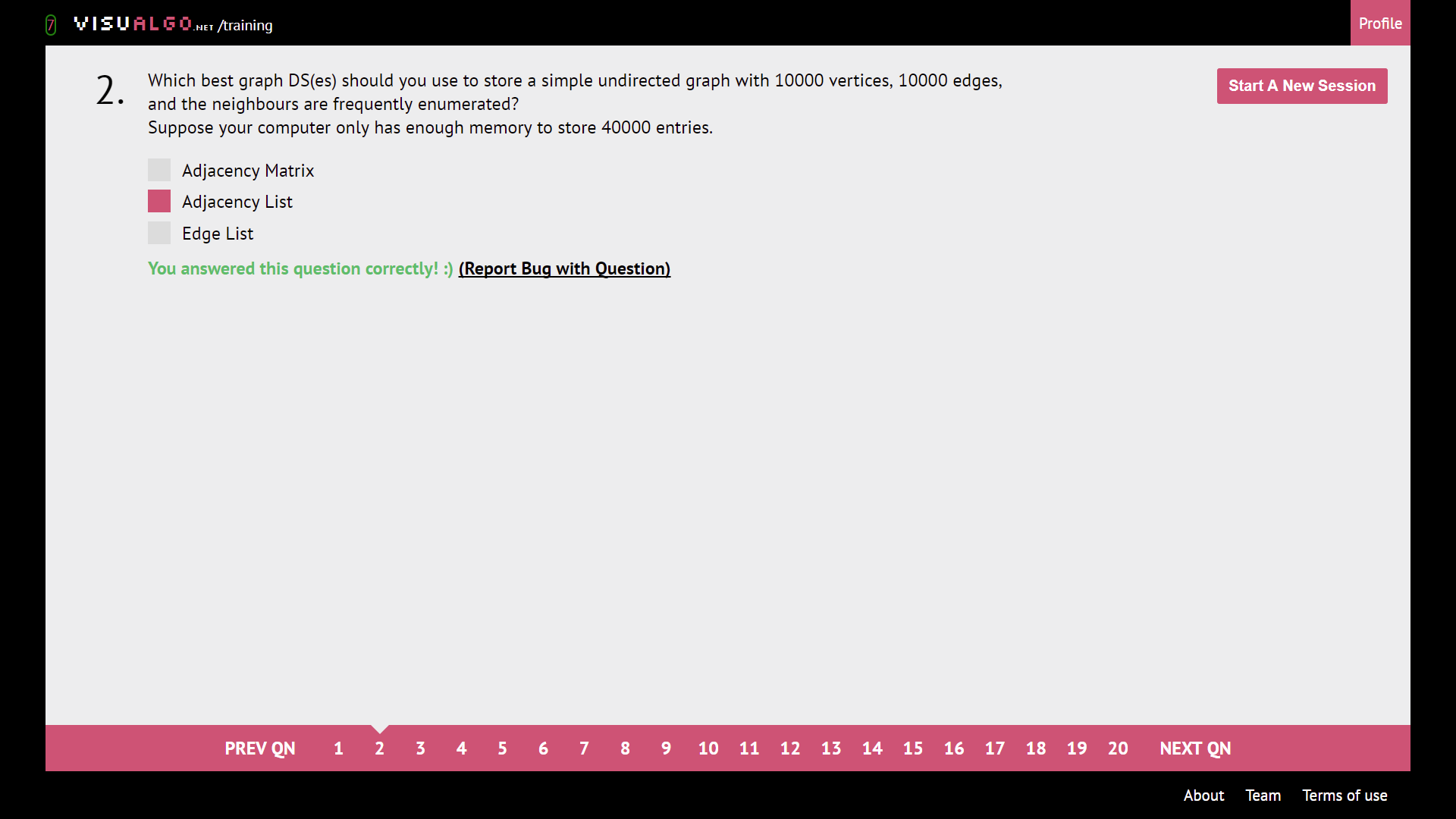
Directed acyclic, just make sure small number -> bigger number.

Tree: V vertices, V-1 edges, acyclic.

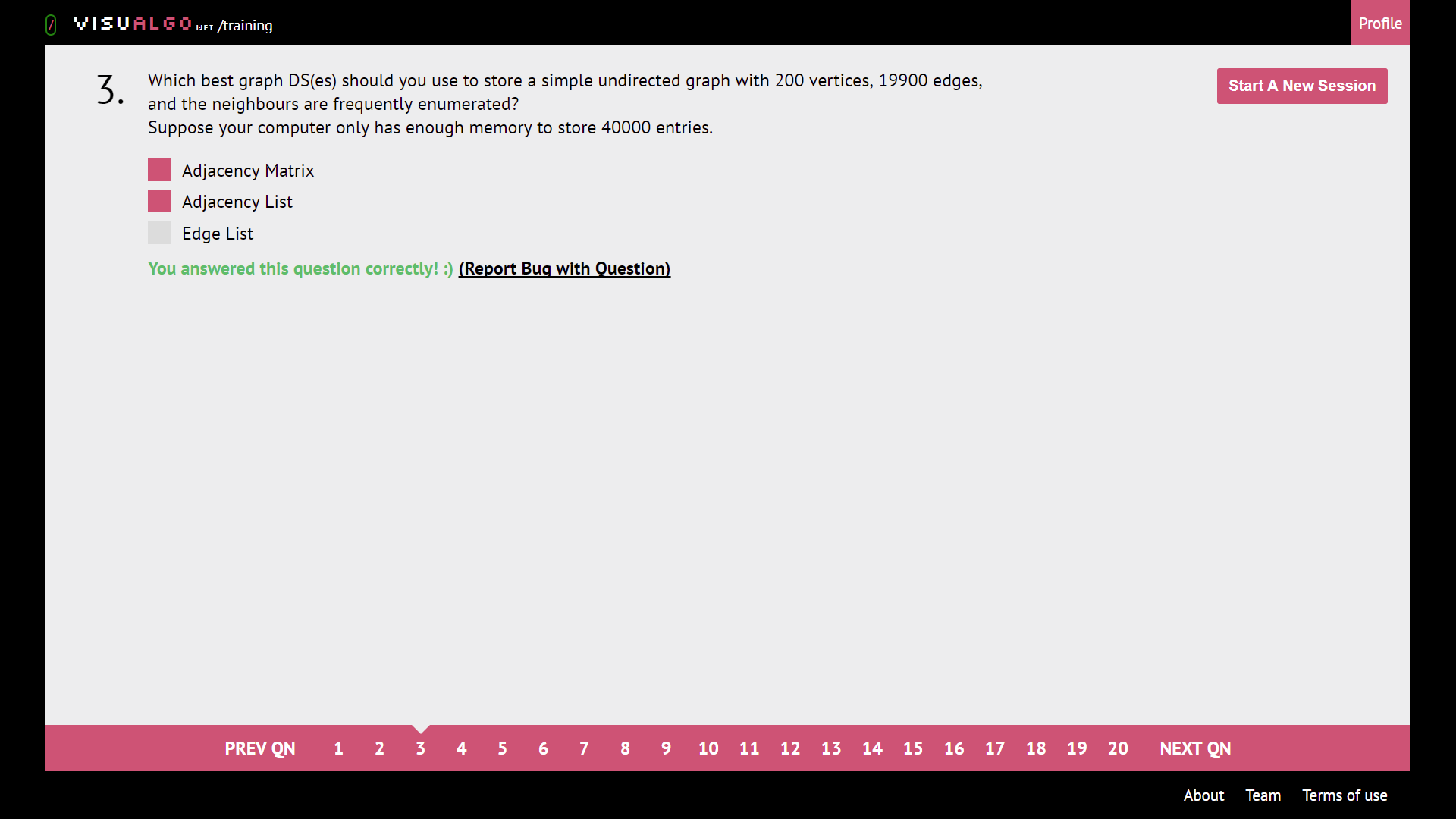
Complete graph: An edge between any pair of vertices.

Connected graph: From any vertex, can get to all other vertex

Connected Components: a single vertex no edges are one component



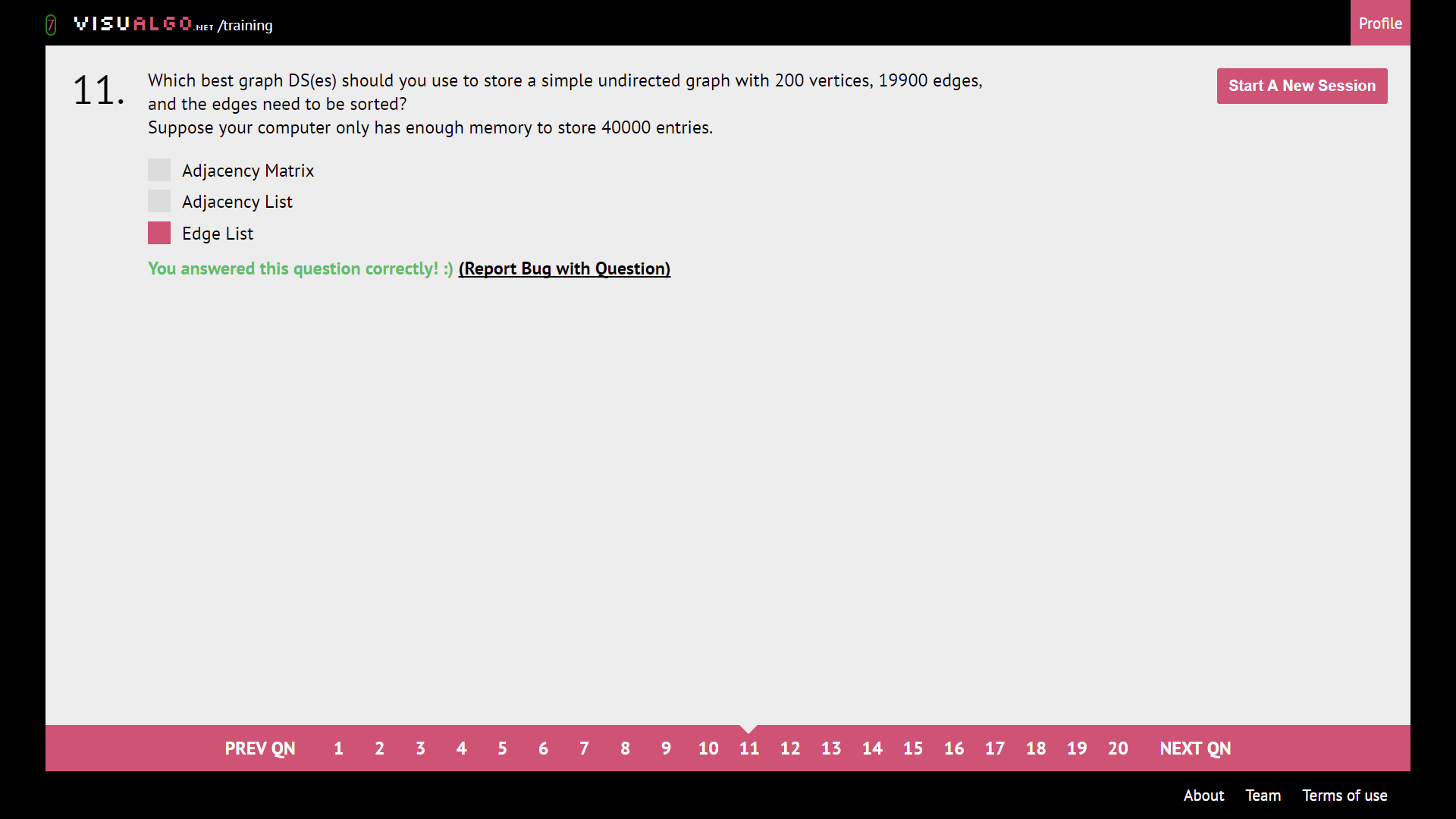
Need to enumerate neighbours, **BUT NOT ENOUGH MEMORY (we need V^2 entries for AM)**



Adjacency Matrix enumerate not as fast but still accepted, **so depends on memory.**



Ask about existence of edge -> AM (Good for Floyd-Warshall’s 4 line wonder :D)



Edge list can be sorted by edge.

|  |  |  |  |
| --- | --- | --- | --- |
|  | AM (Adjacency Matrix) | AL (Adjacency List) | EL (Edge List) |
| Sort | O(V^2) | O(VE log E) | O(E log E) |
| Enumerate | O(V^2) for frequent enumeration | O(1 + E), frequent is O(E) | O(E), frequent O(E) leads to O(VE) |
| Check Existence | O(1) | O(VE), O(V) to iterate all vertexes and O(E) find | O(E), frequent O(E) leads to O(VE) |

* AM: good querying if edge exist
* AL: good enumeration
* EL: good sorting

Adjacency Matrix querying

Ans: Just take [n][m] as **weight of n 🡪 m**

Diagram

Description automatically generated with medium confidence

**How many cells in the Adjacency Matrix are not empty?**

Ans: Basically find the **sum of degrees/number of edges** of the graph

NOTE: for undirected edge 🡪 **multiply by 2**

For directed edge 🡪 **don’t multiply**

**Ex:** We have 6 edges, undirected, so sum of degrees and non-zero graphs are 6 x 2 = 12

Note: **how many cells are filled?**

**Ans: V^2 – sum of degrees 🡪** in this case its 6^2 – 12 = 24

A picture containing diagram

Description automatically generated

Draw a complete graph

Ans: **A complete graph is where every vertex can visit the rest of the vertexes**

A close-up of a pair of glasses

Description automatically generated with low confidence

Sum of all degrees

Ans: if undirected, multiply 2, if directed, don’t multiply. Basically, count all edges

How many entries are there in the Edge List?

Ans: count number of edges. If undirected, count as 2

**What is the in-degree/out-degree of vertex n**

**Ans:** in-degree refers to number of incoming edges into that vertex. Out-degree refers to number of edges going out from that vertex.

Draw a tree with n vertices

Ans: a tree is an undirected graph in which **any 2 vertices are connected by exactly one path**, or equivalently a **connected acyclic undirected graph**.

Is the graph in a picture a Tree?

Diagram

Description automatically generated with low confidence

How many cells are there in the Adjacency Matrix for this graph?

Ans: V^2, V being total number of vertexes

Select the statement(s) that are true for the graph below.

Ans: **a graph is connected if there is a point from one point to any point in the graph**

A picture containing transport, handcart, bicycle

Description automatically generated

Here is a DAG with 6 vertices and MAX 14 edges

Suppose vertex 0 is removed from the following graph. How many connected components are there in the resultant graph?

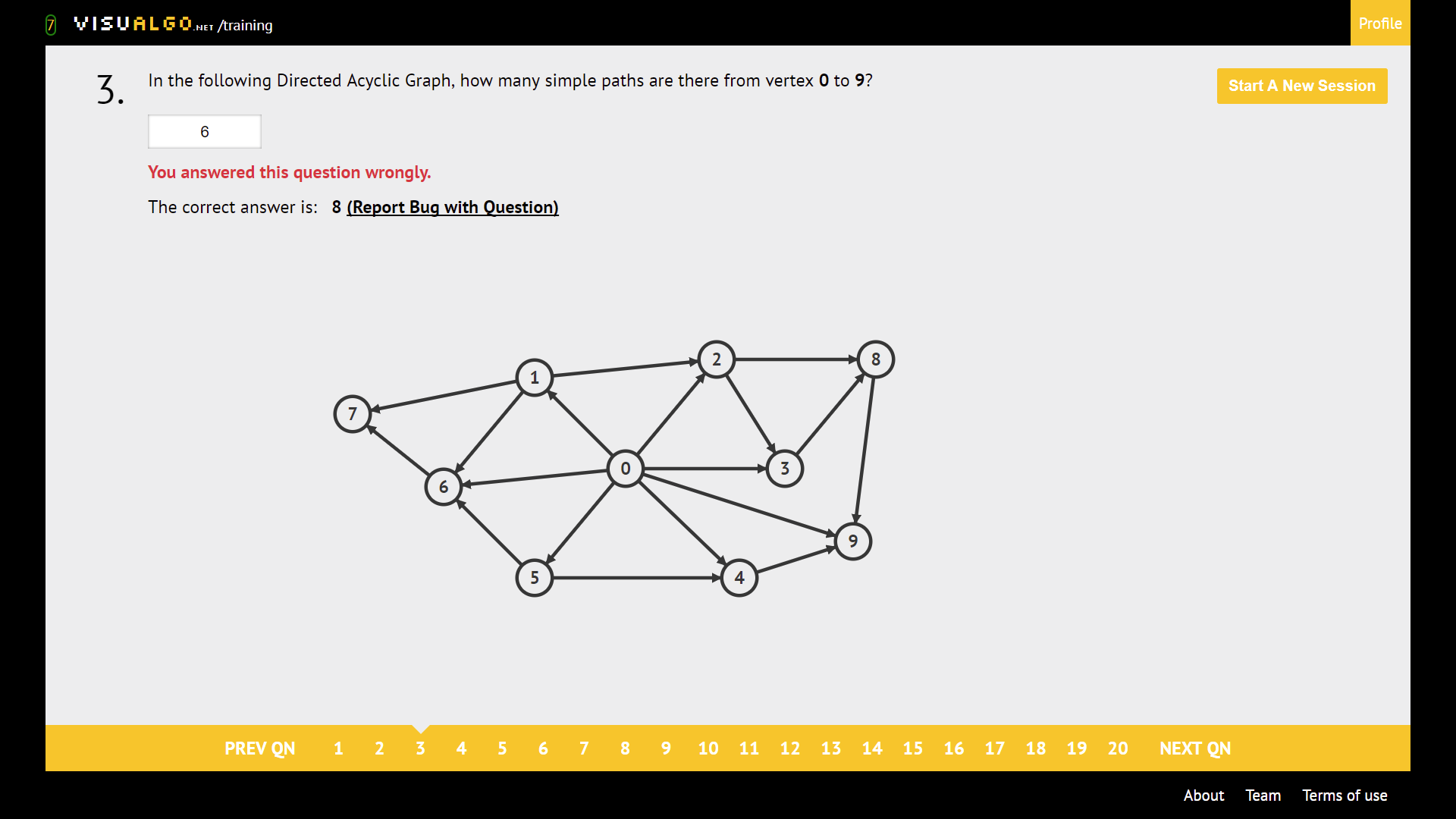
Diagram

Description automatically generated

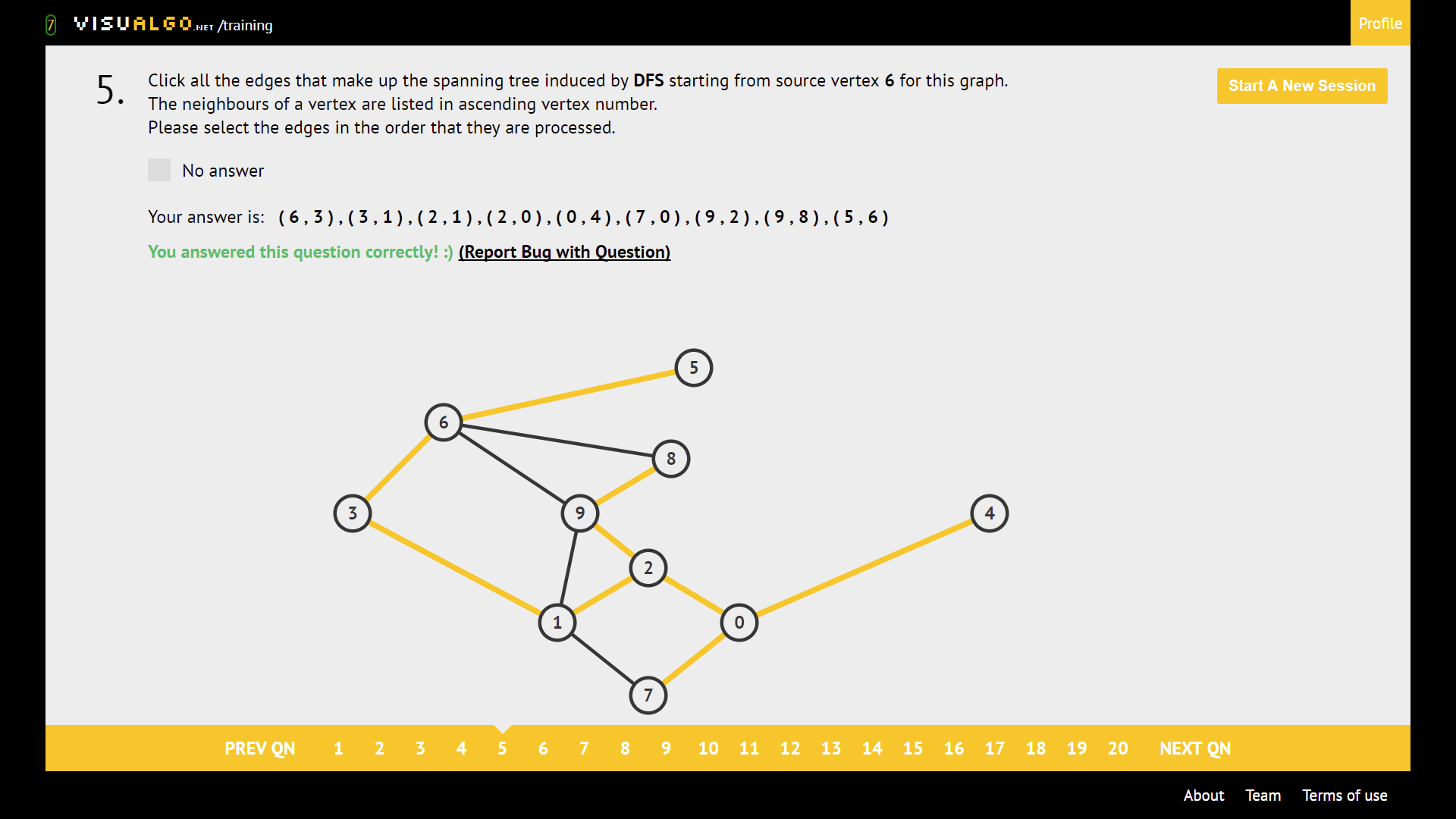
Valid simple path(s)

**Ans: Note: You cannot have vertexes loop back to itself, like 1->1 as simple graphs don’t do that**

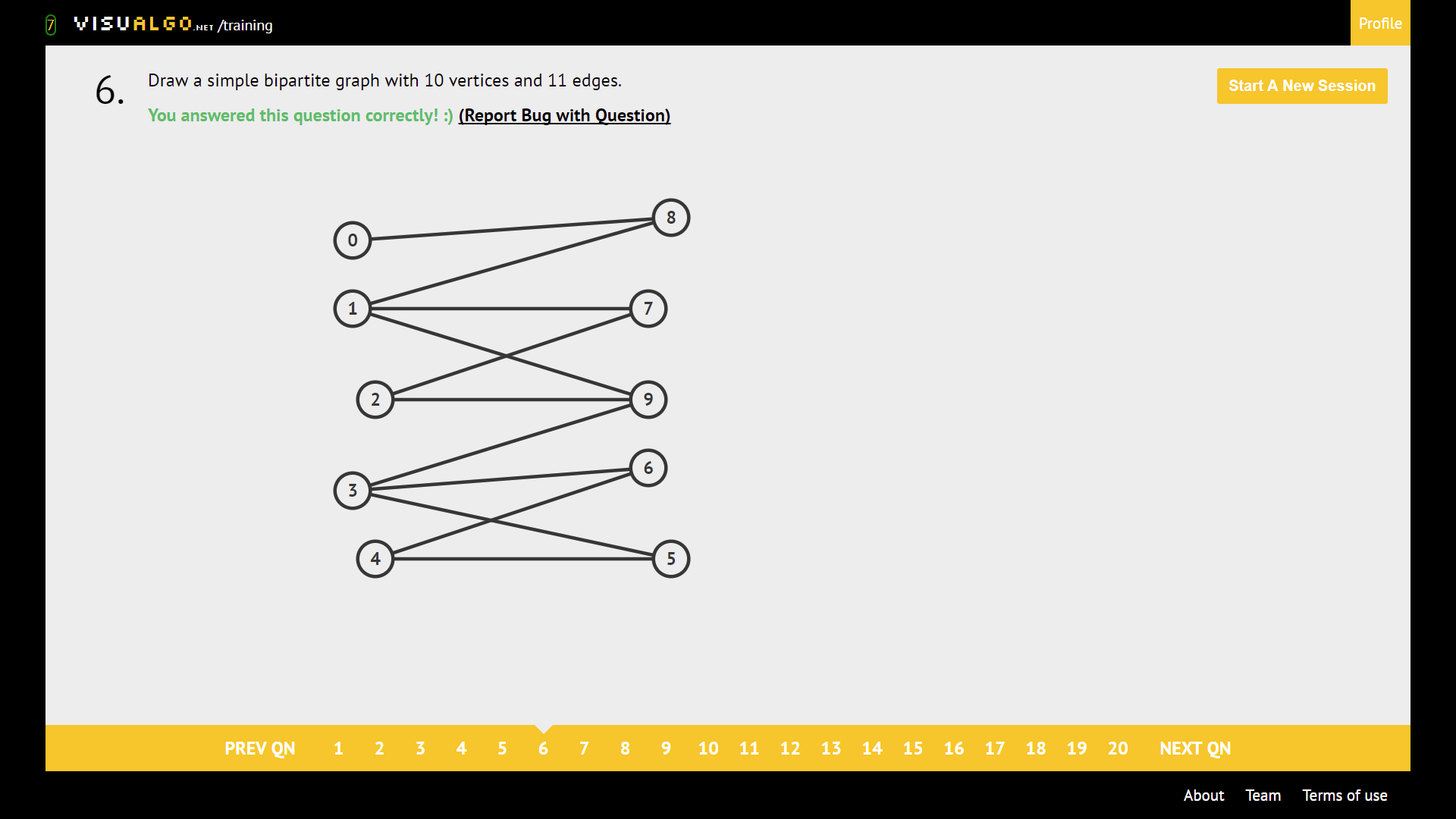
**Graph traversal**



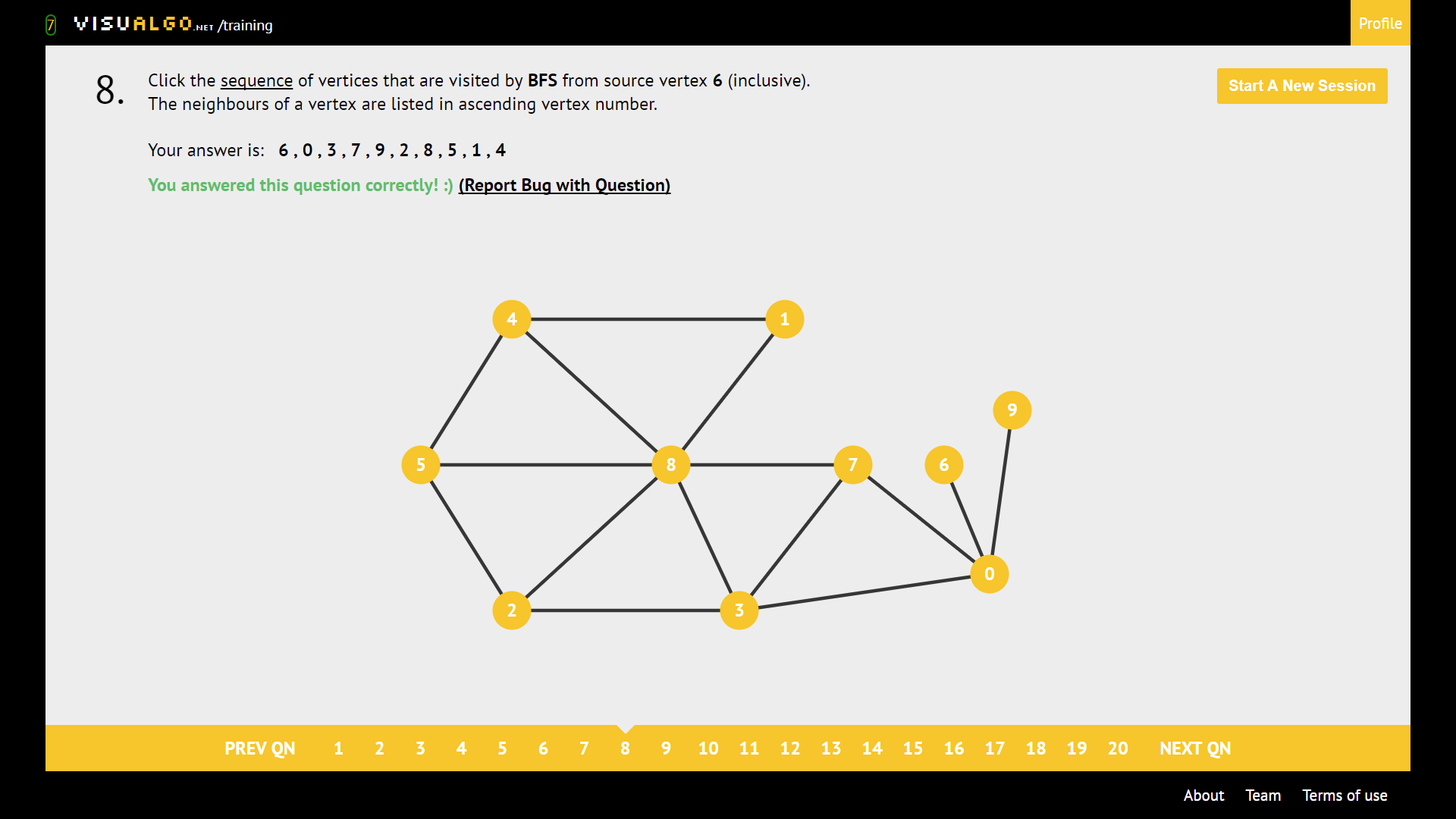
Write down all the paths. (tedious, but idiotproof)



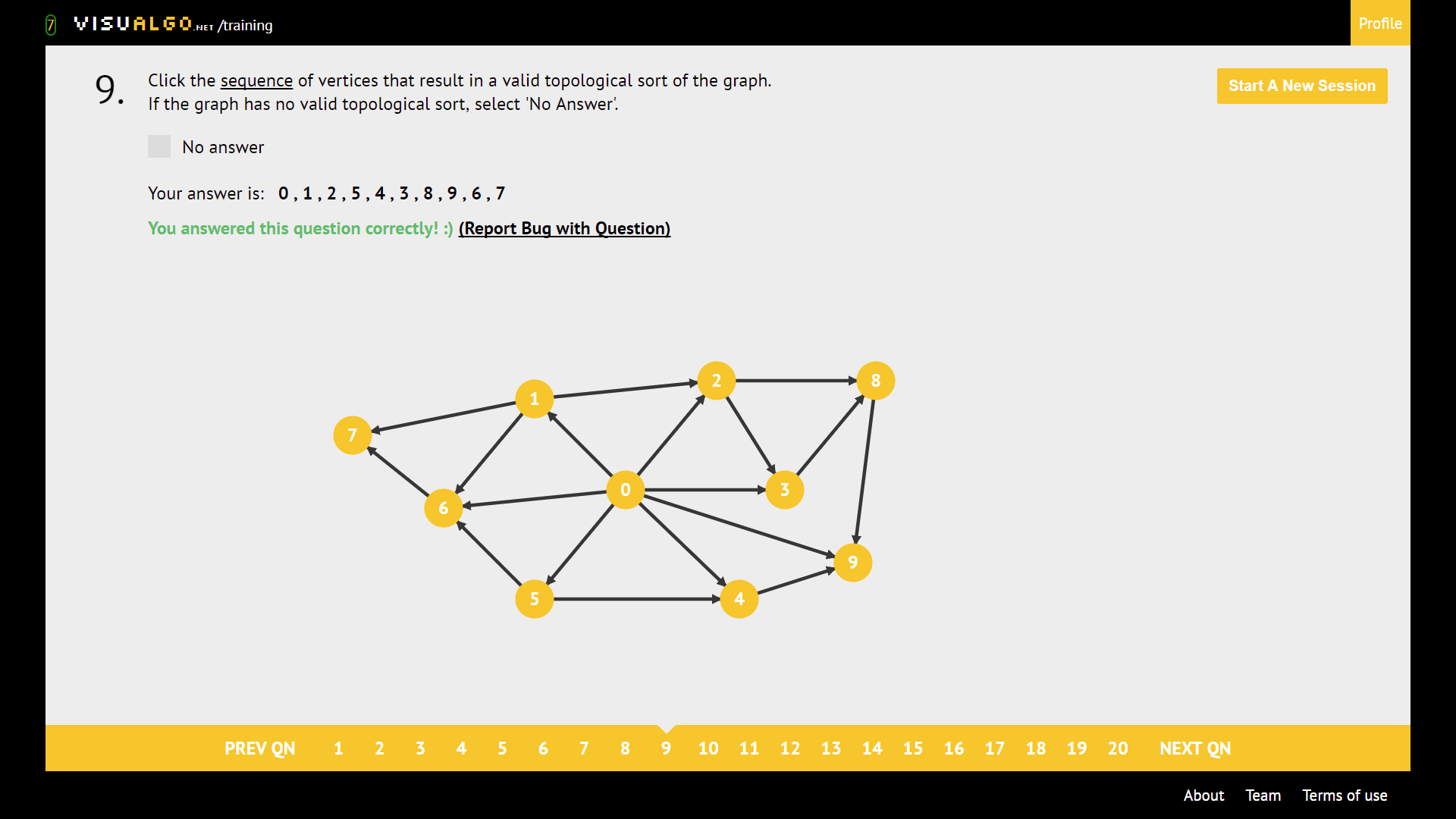
DFS: Go deep, recurse back and go deep for neighbour. Take node of order of neighbours.



Disjoint set members are not connected to one another.



BFS: Visit all neighbours, then all neighbours of neighbours, and so on. Keep track of the order of exploration of neighbours!



A picture containing chart

Description automatically generated

Chart

Description automatically generated

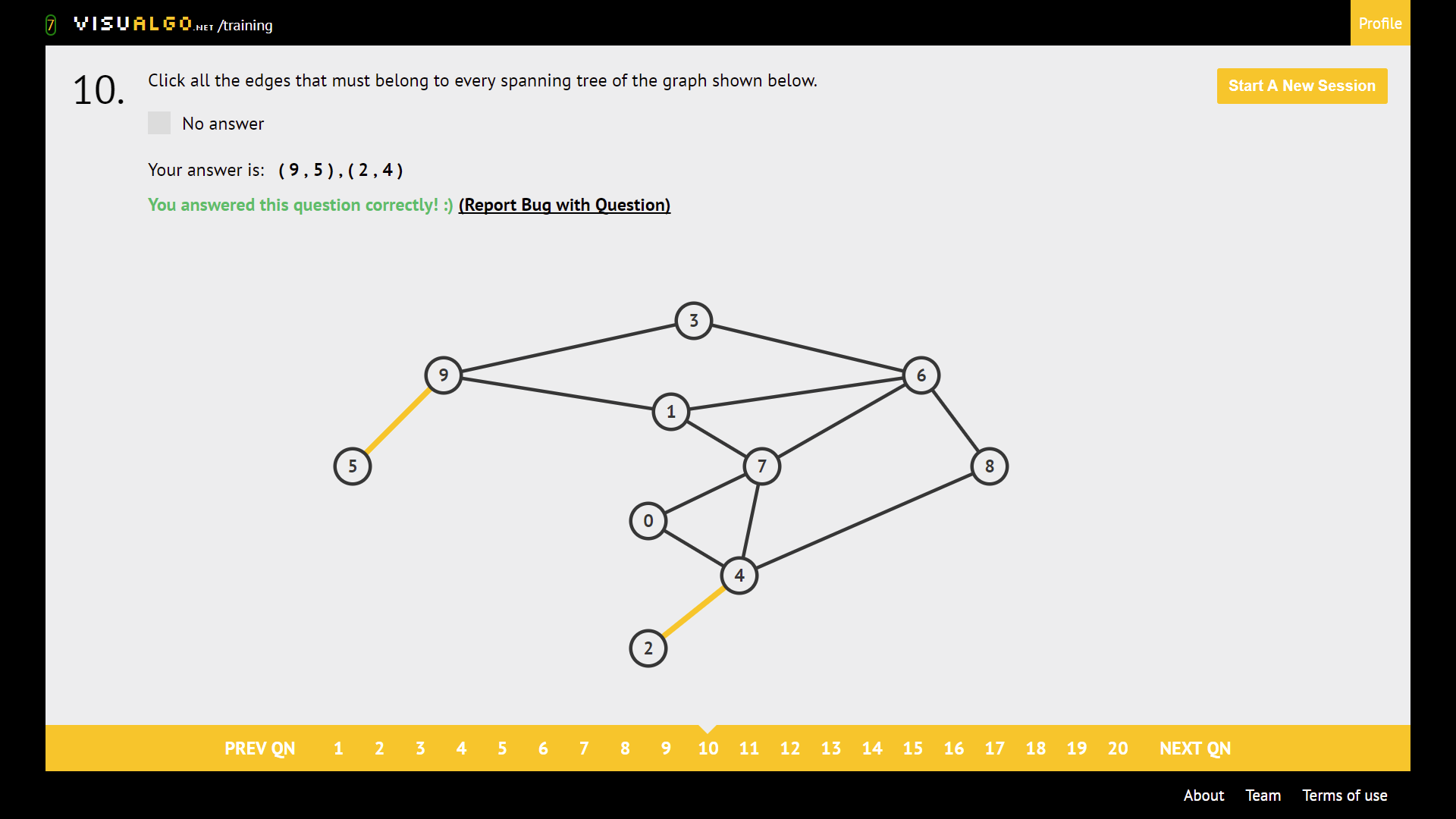
Topological sort

* Go backwards from the node with no outgoing edges but only incoming edges
* If you encounter another node with another outgoing edge, that node comes before the node its attached to
* ie: 6, 1, 9, as 6 must come before 1

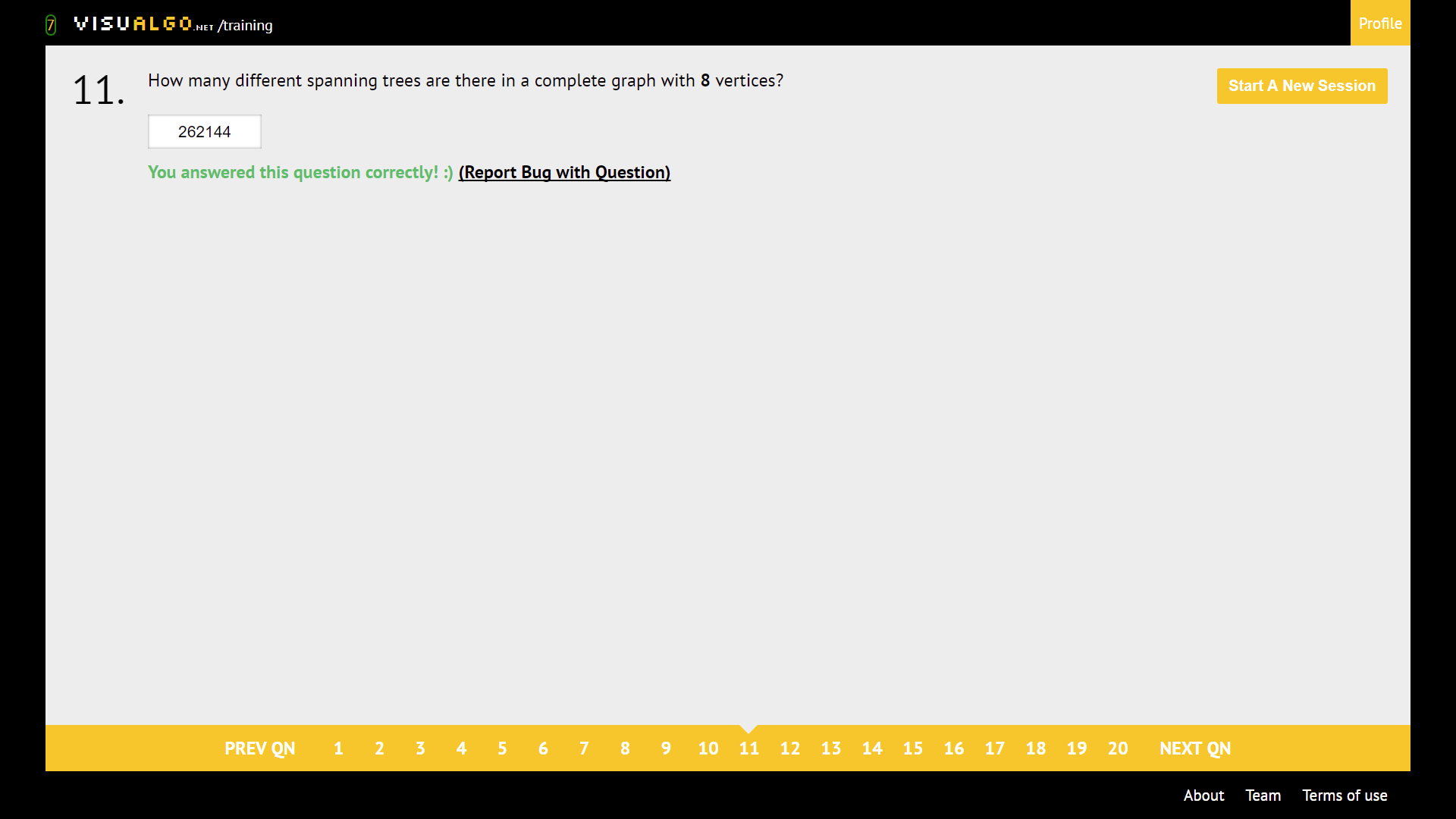
belong to every spanning tree.

Ans: pick the hanging edges

Vertex with only 1 edge. “Burn all the bridges”



How many different spanning trees are there in a complete graph with n vertices?



, n^n-2

Print DFS



Chart, line chart

Description automatically generated

Print comes after exploring all neighbours -> Post-order traversal

1. root is always last
2. go left then right
3. left child, then right child, then go to subtree root

Is the graph bipartite?

Ans: **use BFS algo, if 2 colour clashes, orange suppose to blue and vice versa, it’s a NOT bipartite**

**Usually, a non-tree or 3 vertex cyclic graph is NOT bipartite**

**Bipartite Ex:**

Chart, shape, line chart, polygon

Description automatically generated

**Non-bipartite**

A picture containing watch

Description automatically generated

**GRAPH DRAWING**

DAMN BRIDGES

Draw a simple connected graph with n vertices and m edges such that the graph contains t bridges

Diagram

Description automatically generated

Bridge: Edge of undirected graph which removal disconnects the graph.

**Bridge edge tip: 3 bridges are 3 hanging edges next to a SCC**

Cut vertex/Articulation point: Vertex of undirected graph which removal disconnects the graph.

**Cut vertex tip: the 2-cut vertex would 2 hanging vertexes next to a SCC But not the end vertex of the hanging edge**

**Ans: below, the cut vertex are 7 & 8, BUT NOT 9!!!**

Draw the Dipper Graph!

Diagram

Description automatically generated

Connected component/ SCC

Ans: Draw a cycle + X hanging edges, cycle is one SCC, each hanging edge is a SCC

**Note: if the hanging edges have paths back and forth to them, its one SCC**

Diagram

Description automatically generated

**Above is 3 SCC**

**Below is 1 SCC, as 7 & 8 are part of the cycle**

Chart, line chart

Description automatically generated

DFS & BFS can run O(V^2) in

­­­­­­Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**SSSP**

* Negative Weight Cycle Definition: if there a cycle that has a negative edge, **it’s a negative weight cycle**

**Note that this can exist for 2 vertex with 2 edges, tricky to spot these kinds of NWC**

Run Indefinitely

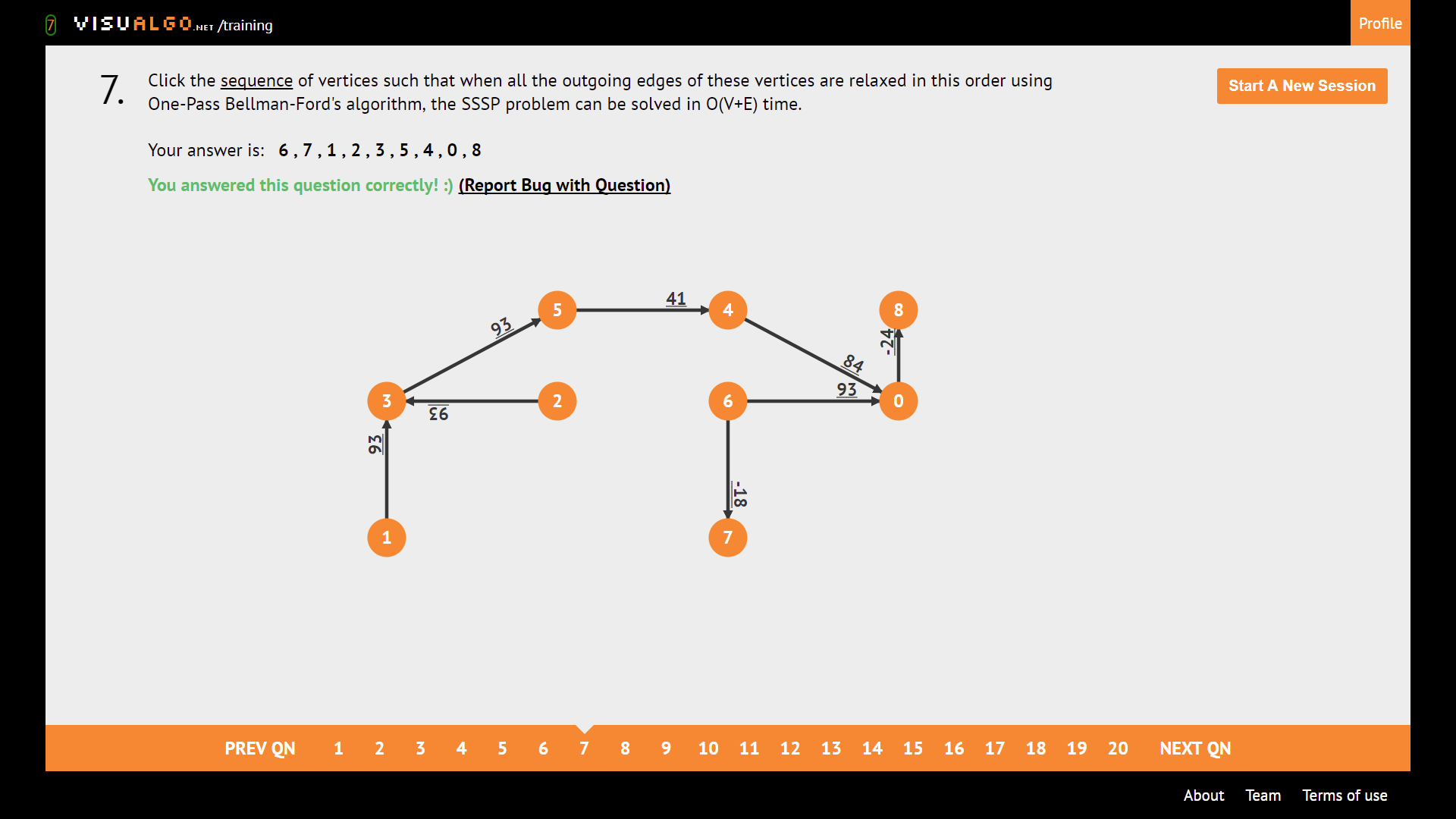


Diagram

Description automatically generated with medium confidence

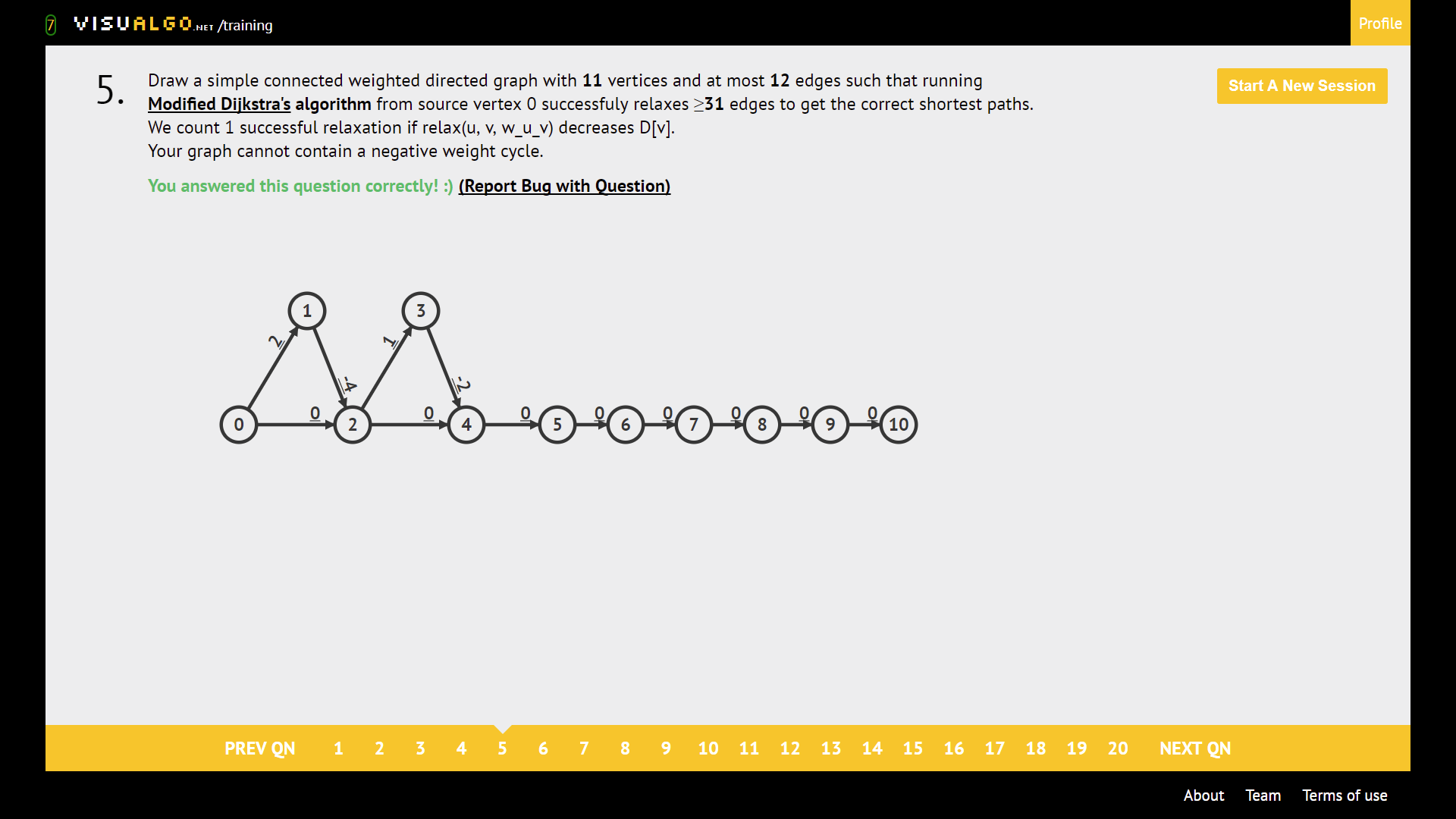
Drawing a terminating Djikstra

Create a negative cycle. **MAKE SURE THE WEIGHTS ARE DISTINCT**

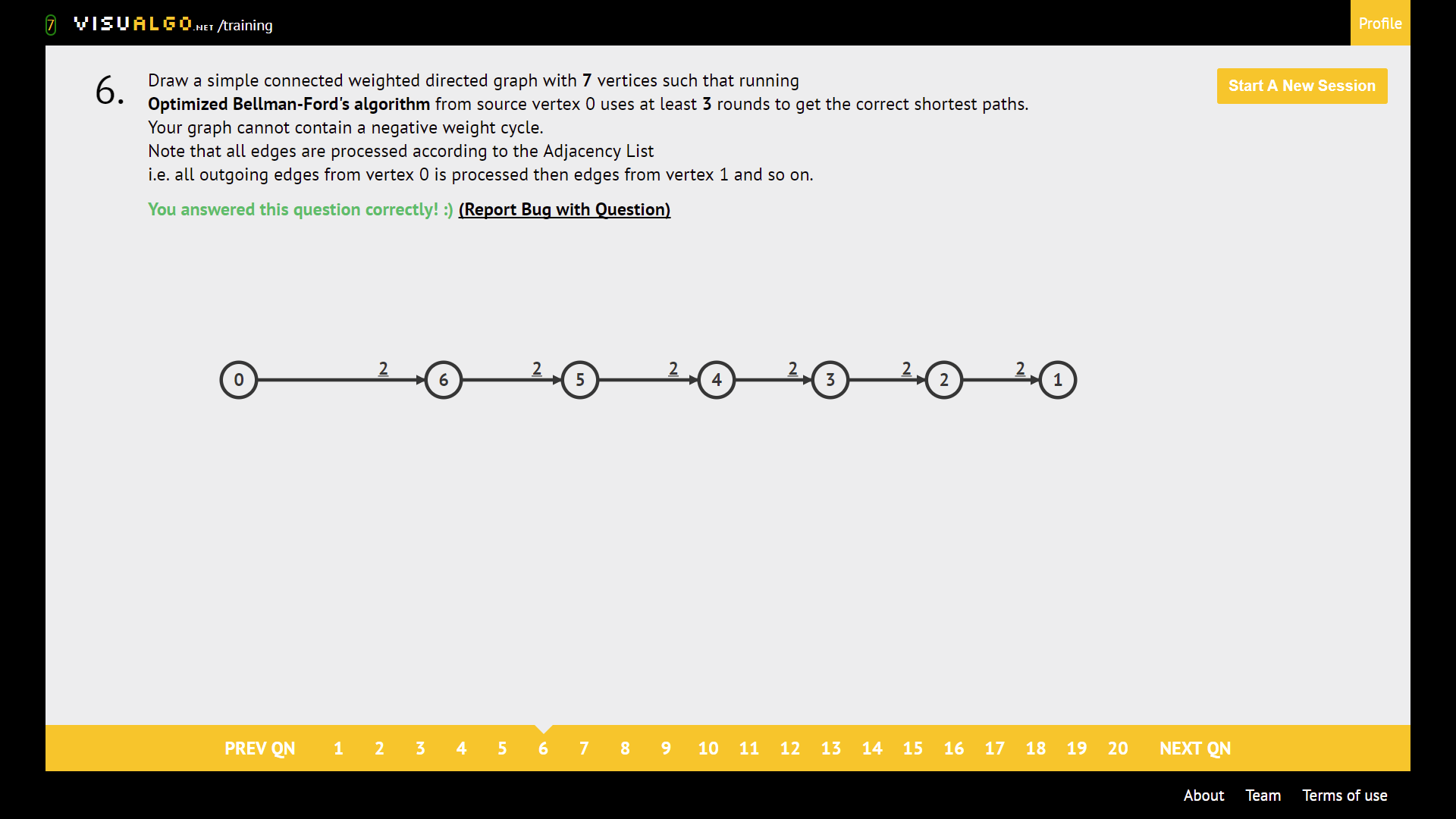


Topological sort. For relaxed edges

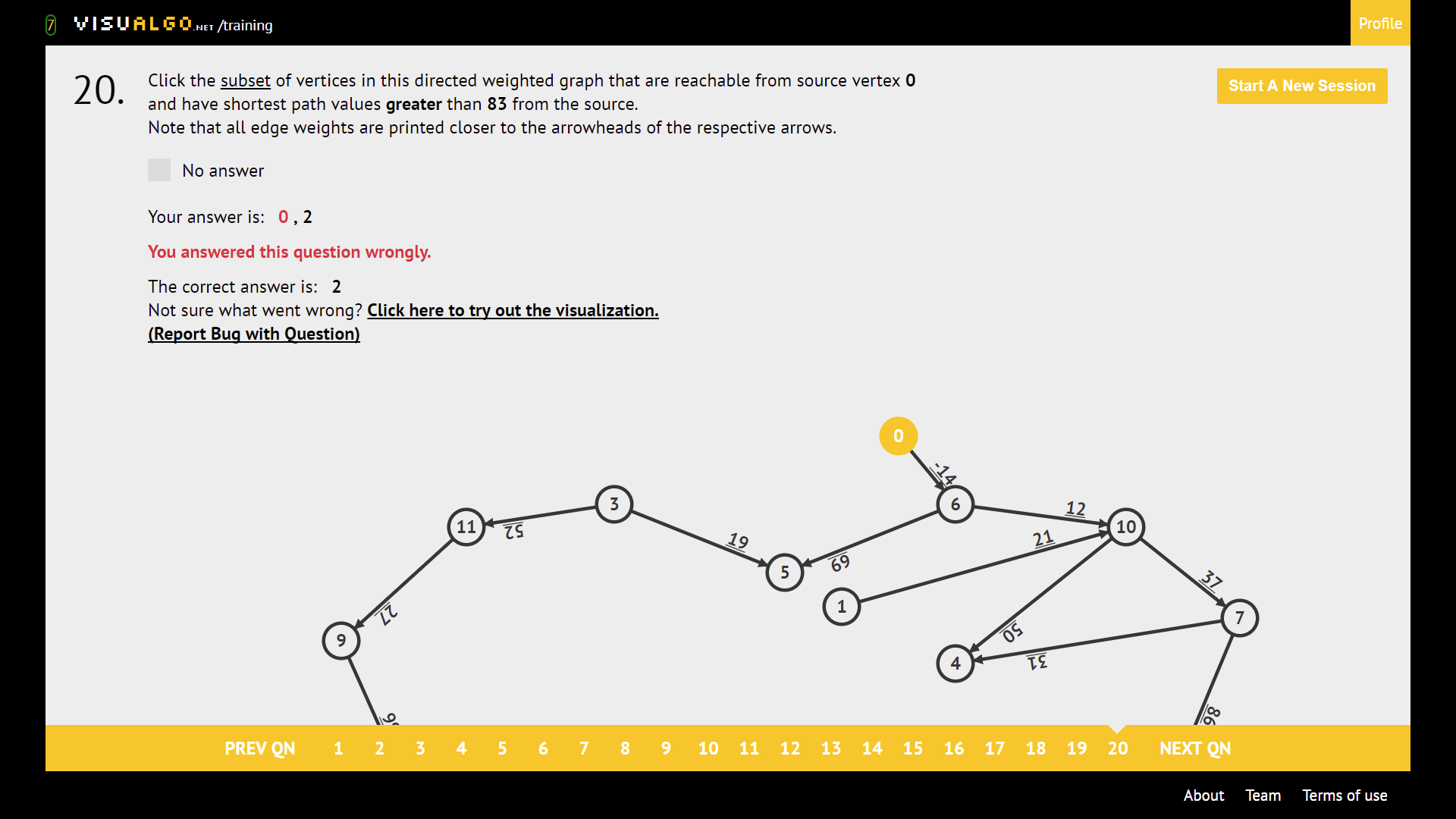
Sequence outgoing edges relaxed



Dijkstra Killer. **THERE ARE (E-V+1) TRIANGLES.**



Bellman-Ford Killer. Optimised Bellman-Ford algorithm



If greater than DON’T click source node. If lesser than CLICK source node. This is because source node dist is = 0.

**SOURCE VERTEX must have clicked on source!!!!**

**Run BFS on the source**

Diagram

Description automatically generated

**NOTE:\*\*\* if the there is ONLY 1 SSSP possible, BFS will work, as seen below**

**The only way to solve this is to do BFS, then see if all the distances are correct**

**Usually nodes with more than 2 incoming edges + more than or equal to 2 hops are wrong**

Diagram

Description automatically generated

Diagram

Description automatically generated

A picture containing diagram

Description automatically generated

Diagram

Description automatically generated

TIP:

If Greater -> **DO NOT CLICK ON source node**

If Less -> **CLICK ON source node**

Note path weight criteria.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Bellman-Ford | Original Dijkstra | Modified Dijkstra |
| Terminate | Always | Always | **Does not terminate when there’s negative weight cycle** |
| Wrong when there’s | Negative weight cycle | Negative weight  And Negative weight cycle | Negative weight cycle (it doesn’t even terminate in the first place ☹ ) |

NOTE EXAMPLE BELOW:

A picture containing diagram

Description automatically generated

After n pass(es) of the Bellman-Ford algorithm

If D[x] is INFINITY, select ‘No Answer’

Ans: **See that that there are only a few predetermined paths to the answer, just calculate that path**

* **\*\* Take note of the source node, you can ignore edges if you find the source node! See image 2**
* **BECAREFUL of negative weight cycles! If you look at image 3, the ans is NO ANSWER! Due to a negative weight cycle**

**Steps to take**

* **Check for negative weight cycle**
* **Find source node**
* **Compute SSSP to destination**

Diagram

Description automatically generated

Diagram

Description automatically generated

* The blue circled edge shows a negative weight cycle, leading to a **wrong answer!**

Diagram

Description automatically generated

Diagram

Description automatically generated

**MST**

* Kruskal: sort all edges in EL, use a UFDS to prevent cycles
* Prim’s: use an AL and Priority Queue, if else to prevent cycles

Note: Prim or Kruskal’s MAXIMUM Spanning Tree

Chart, line chart

Description automatically generated

Draw a graph for Kruskal’s algo such that all are visited.

Chart

Description automatically generated with medium confidence

**NOTE WEIGHTS MUST BE UNIQUE!!!!**

Only has one unique Minimum Spanning Tree.

* If each edge has a distinct weight then there will be only one, unique minimum spanning tree.

Diagram

Description automatically generated

Road network of Country A.

* 1) You must click on the **roads that cannot be demolished. These edges are fixed**
* **2) Now you must click the rest of the edges for Min ST or Max ST to COMPLETE THE MST with the fixed edges.** The rest of the edges must connect to ALL vertexes
* Note if MIN or MAX Spanning Tree.

A picture containing chart

Description automatically generated

Select the edges (in any order) that do not belong

* NOTE: if MIN or MAX Spanning Tree
* **1) First complete the MIN or MAX ST using Prim’s or Kruskal’s**
* **2) Next click on the odd edges out/ edges that do not belong to the MST**

Diagram

Description automatically generated with low confidence

MINiMAX

MINiMAX stands for: **MIN path, MAX edge**

* **Find the MIN PATH (SSSP) to the destination given that you can only travel on the MST**
* Now find the **MAXIMUM EDGE** on that SSLP
* Note, the edge is **biggest to the path to destination, if there’s bigger edge not on the path, ignore it**
* \*\* find the MIN Spanning Tree first then find **MAX edge**
* **\*\*\* Your selected path must be on the way to your desired node from source**

Diagram

Description automatically generated

MIN or MAXI-MIN: **MAXimizes MINimum edge**

**MAXiMIN** stands for: MAX path, MIN edge

* **Find the MAXIMUM PATH (SSLP single source Longest Path) to the destination**
* Now find the **MINIMUM EDGE** on that SSLP

**\*\* Find the MAX Spanning Tree First, then pick MIN edge**

**\*\*\* Your selected path must be on the way to your desired node from source**

Diagram

Description automatically generated with low confidence

form X connected components (in any order)

* 1) Draw the Minimum Spanning Tree
* 2) remove k largest to make k + 1 components, in this case, we remove k = 1, to get 1 + 1 = 2 connected components
* 3) **It doesn’t matter if the MST is broken or not, just follow this formula**
* **4) \*\* MST must be SORTED EDGES**

Diagram

Description automatically generated with medium confidence

Second Best Minimum Spanning Tree

* First **create an MST, and take note of all the edges that were rejected (because they could form a cycle)**
* And then now check for every cycle, find **two edges in that cycle a and b**, a in MST but with maximum edge, b not in MST but also with maximum edge, **that gives the smallest difference, then you can replace a with b**
* Basically, find the **closest <accepted edge, rejected edge> pair there is**
* Below we see that **the closest pair was <13, 14>**, so we should pic **(3,5) 🡪 14**

Diagram

Description automatically generated with medium confidence

* Below we see that the **closest pair was <14, 23>, so we pick (3,7) 🡪 23**

Chart

Description automatically generated

Diagram

Description automatically generated

# **Tables**

Max Heap max swaps (min swaps = 0. what if it’s already a heap?)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Elements | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Swaps | 1 | 3 | 3 | 4 | 4 | 7 | 7 | 8 | 8 | 10 | 10 | 11 |

Max Heap Max comparisons in O(n) time (querying 2 nodes are 2 comparisons)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Elements | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Compare | 2 | 6 | 6 | 7 | 8 | 11 | 14 | 15 | 16 | 18 | 20 | 21 |

Max Heap Min Comparisons: N-1 (min comparisons is NOT ZERO)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Elements | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Compare | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

# **Binary Search Tree: how many permutations**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Elements | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Catalan # | 1 | 2 | 5 | 14 | 42 | 132 | 429 | 1430 | 4862 |

**AVL Tree: Minimum # of vertices (Height(n) = height(n-1) + height(n-2) + 1)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Height | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Vertices | 1 | 2 | 4 | 7 | 12 | 20 | 33 | 54 | 88 | 143 | 232 | 376 | 609 | 986 | 1596 | 2583 |

# **Graph Structures**

Neighbours Frequently Enumerated AND **HAS ENOUGH MEMORY: AL + AM** (We need V^2 memory for AM)

Neighbours Frequently Enumerated AND **NOT ENOUGH MEMORY:** **AL ONLY** (memory < V^2 )

Existence of edge(u,v): **AM ONLY**

Edges need to be sorted: **EDGE LIST ONLY**

**Number of spanning trees in complete graph n^(n-2)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vertices | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Ans | 3 | 16 | 453 | 1296 | 16807 | 262144 | 4782969 | 100000000 | 2357947691 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Bellman-Ford | Original Dijkstra | Modified Dijkstra |
| Terminate | Always | Always | Does not terminate when there’s negative weight cycle |
| Wrong when there’s | Negative weight cycle | Negative weight | Negative weight cycle |

**Bitmasking/UFDS tallest tree**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| h | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| N = 2^h | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 |

**Hash Tables**

T = table value after first getting value from table

QP 🡪 (T + 1 \* 1) % 13 🡪 (T + 2 \* 2) % 13 🡪 (T + (n+1) \* (n+1)) % m, **T stays after first round**

LP 🡪 (T + 1 \* 1) % 13 🡪 (T + 2 \* 1) % 13 🡪 (T + (n+1) \* 1) % m

DP: let’s say second hash is 11 – k%11 and primary hash is k%13

Ex: if 75 % 13 = 10, and 75 % 11 = 9, **where m = 9, then 11 – m = 2,**

**So, its (T + (n+1) \* 2) % 13**

🡪(T + 1 \* (11 – m)) % 13 🡪(T + 2 \* (11 – m)) % 13 🡪 (T \* (n+1) \* (11 -m)) % 13

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **%11** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  |  |  |
|  | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |  |  |  |
|  | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |  |  |  |
|  | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 |  |  |  |
|  | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |  |  |  |
|  | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |  |  |  |
|  | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 |  |  |  |
|  | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 |  |  |  |
|  | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 |  |  |  |
|  | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **%13** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  |
|  | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |  |
|  | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |  |
|  | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 |  |
|  | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |  |
|  | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |  |
|  | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |  |
|  | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 |  |
|  | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **%12** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |  |  |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |  |  |
|  | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |  |  |
|  | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |  |  |
|  | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |  |  |
|  | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |  |  |
|  | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |  |  |
|  | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |  |  |
|  | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |  |  |
|  | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 |  |  |
|  | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **%14** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|  | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|  | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
|  | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
|  | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 |
|  | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
|  | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 |
|  | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |