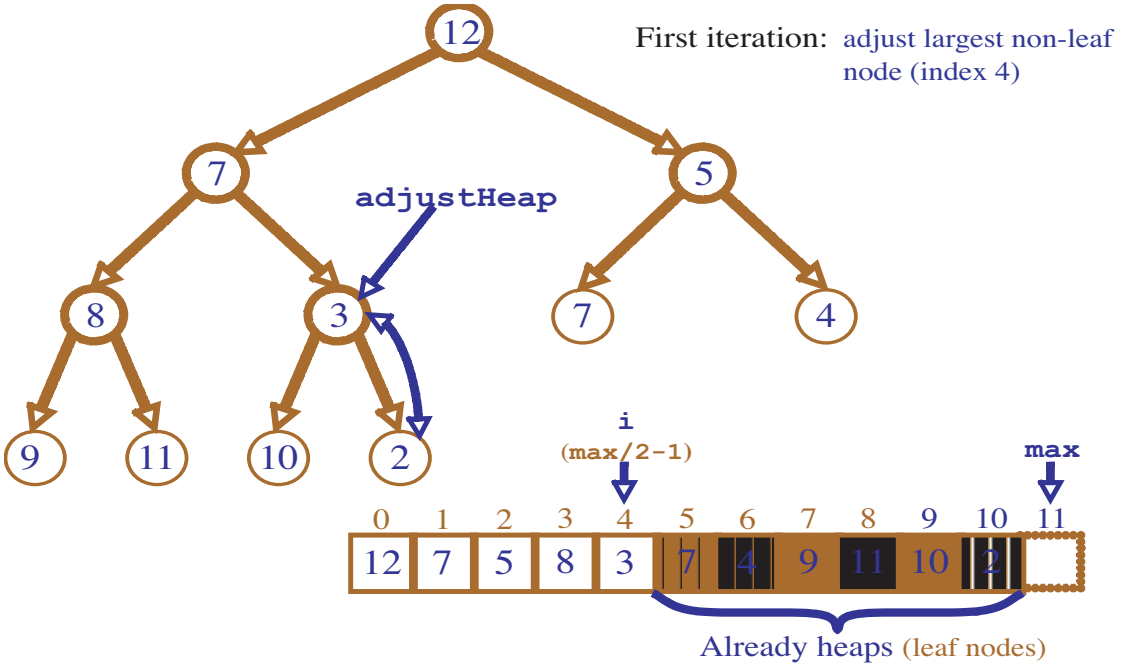
Worksheet 34: BuildHeap and Heap Sort

In preparation: If you have not done so already, you should complete Worksheet 33 to learn more about the heap data structure.

In some applications it is useful to initialize a Heap with an existing vector of values. The values are not assumed to be organized into a heap, and so a routine named buildHeap is invoked for this purpose.

```
void buildHeap (struct dyArray *heap) {
    int max = dyArraySize(heap);  int i;
    for ( i = max/2-1; i >= 0; i--)
      _adjustHeap(heap, max, i);
}
```

To understand the buildHeap algorithm, consider the following diagram:



All values indexed after max/2 are leaves, and are therefore already a heap. The first value that could potentially not be a heap is found at max/2. Walking backwards from this value until the root is reached eventually makes all nodes into a heap.

The heap data structure provides an elegant technique for sorting a vector. First form the vector into a heap. To sort the vector, the top of the heap (the smallest element) is swapped with the last element, and the size of the heap is reduced by 1 and readjusted. Repeat until all elements have been processed.

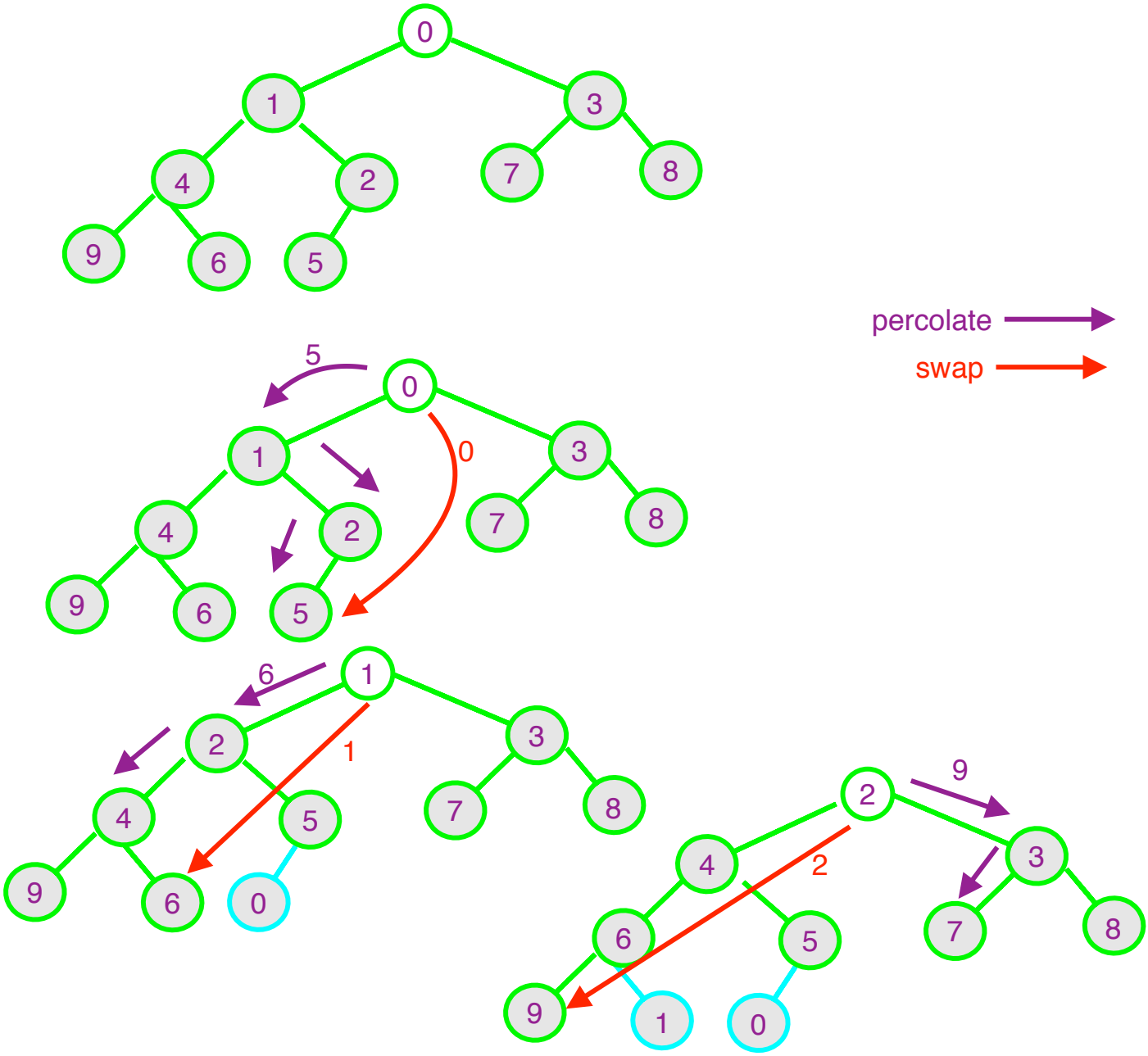
```
void heapsort (struct dyArray * v) { int i;
  buildHeap(v);
  for (i = dyArraySize(v) – 1; i > 0; i--)  {
    dyArraySwap(v, 0, i);
    _adjustHeap (v, i, 0);
  }
}
```

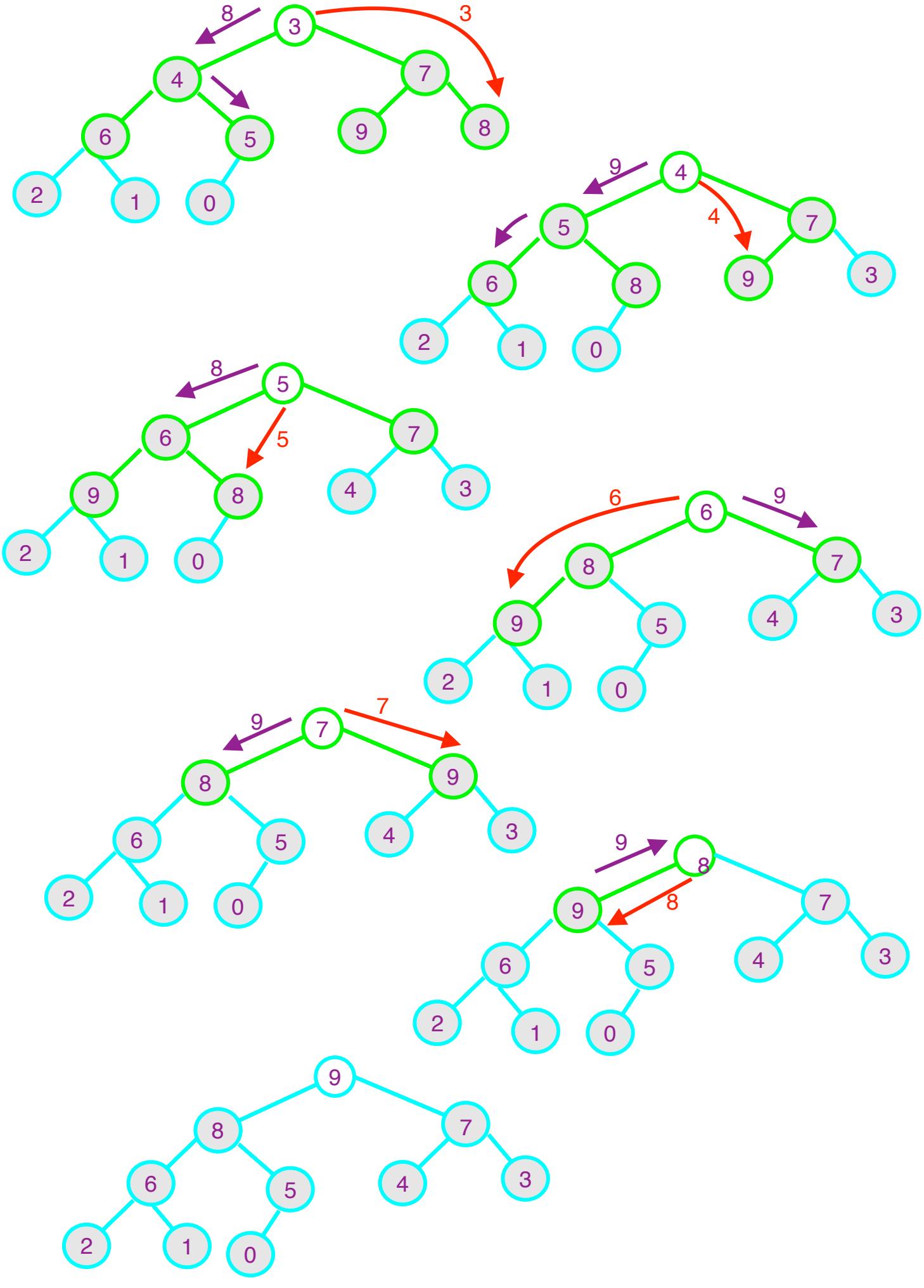
worksheet 34: BuildHeap and Heap Sort Name:

Simulate execution of the Heap sort algorithm on the following values:

9 3 2 4 5 7 8 6 1 0

First make the values into a heap (the graphical representation is probably easier to work with than the vector form). Then repeatedly remove the smallest value, and rebuild the heap.




8
3
4
7
6
5
9
8
2
1
0
9
4
4
5
7
6
8
9
3
2
1
0
8
5
6
5
7
9
8
4
3
2
1
0
6
6
9
8
7
9
5
4
3
2
1
0
9
7
7
8
9
6
5
4
3
2
1
0
9
8
8
9
7
6
5
4
3
2
1
0
9
8
7
6
5
4
3
2
1
0