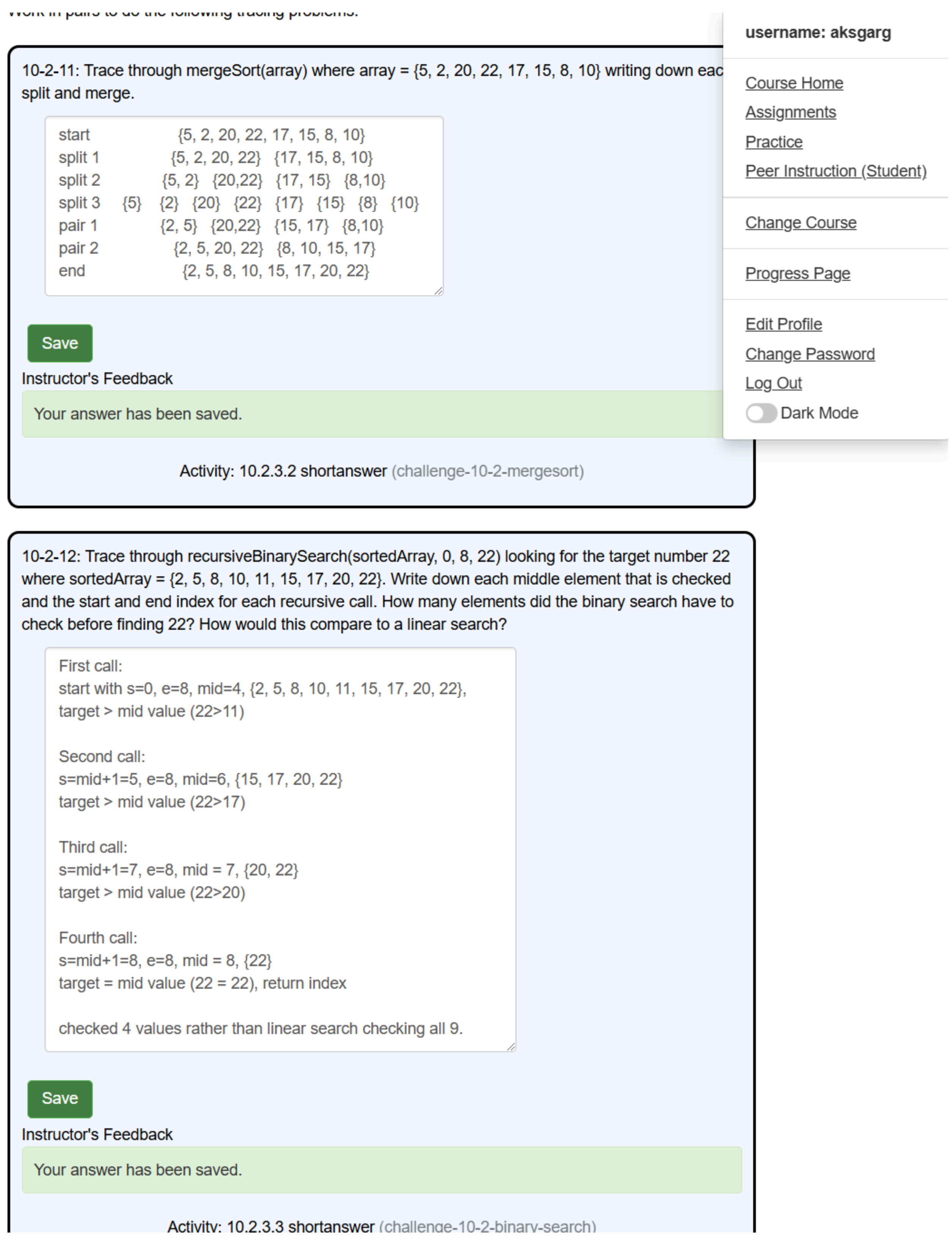
Work in pairs to do the following tracing problems.

10-2-11: Trace through mergeSort(array) where array = {5, 2, 20, 22, 17, 15, 8, 10} writing down eac split and merge.

```
start              {5, 2, 20, 22, 17, 15, 8, 10}
split 1           {5, 2, 20, 22}  {17, 15, 8, 10}
split 2          {5, 2}  {20,22}  {17, 15}  {8,10}
split 3   {5}   {2}  {20}  {22}  {17}  {15}  {8}  {10}
pair 1           {2, 5}  {20,22}  {15, 17}  {8,10}
pair 2             {2, 5, 20, 22}  {8, 10, 15, 17}
end                 {2, 5, 8, 10, 15, 17, 20, 22}
```

Instructor's Feedback

Your answer has been saved.

Activity: 10.2.3.2 shortanswer (challenge-10-2-mergesort)

10-2-12: Trace through recursiveBinarySearch(sortedArray, 0, 8, 22) looking for the target number 22 where sortedArray = {2, 5, 8, 10, 11, 15, 17, 20, 22}. Write down each middle element that is checked and the start and end index for each recursive call. How many elements did the binary search have to check before finding 22? How would this compare to a linear search?

```
First call:
start with s=0, e=8, mid=4, {2, 5, 8, 10, 11, 15, 17, 20, 22},
target > mid value (22>11)

Second call:
s=mid+1=5, e=8, mid=6, {15, 17, 20, 22}
target > mid value (22>17)

Third call:
s=mid+1=7, e=8, mid = 7, {20, 22}
target > mid value (22>20)

Fourth call:
s=mid+1=8, e=8, mid = 8, {22}
target = mid value (22 = 22), return index

checked 4 values rather than linear search checking all 9.
```

Instructor's Feedback

Your answer has been saved.

Activity: 10.2.3.3 shortanswer (challenge-10-2-binary-search)

Quite smooth to see how each method works and to do it by hand. Will do this in the future to map out my algorithms. Very clear to see the small intricacies with indexes in the binary search as well. This was a great way to see how the increments occur to make sure there are no skipped values or infinite loops.