1) Exercise 1

1. The basic operations performed by the algorithm that counts toward the running time is: printing a star, printing a space, and printing a newline.
2. The following algorithm will print 15 stars, 10 blank spaces, and 5 newlines.
3. As a function of n, there are n\*(n+1)/2 operations to print stars, n\*(n-1)/2 operations to print spaces, and n operations to print new lines.
4. The Big O runtime will be O(n^2).

2) Exercise 2

1. The basic operations performed by the algorithm that counts toward the running time is: adding the scores and dividing them to find the average class score.
2. The worst-case running time as a total count of these basic operations will be the following: all scores for each student is added c times. Because there are r student cumulative scores, there are (c-1)\*r additions for the totals. Then, these totals are added up (r-1) times to find the total score in the class. The final average will be the result of dividing exactly once by the number of students.
3. The Big O running time is O(rc).
4. This algorithm is linear. (because the runtime is proportionally linear to the input scores)

3) Exercise 3

1. By unshuffling a deck, you would need to create 4 new arrays with length 13 each, representing the suit and rank of each card. Next, you would have to iterate through the cards and place them in their respective spot within these arrays. This always has a Big O runtime of O(1) as the function’s inputs never change, which should known as the worst-case runtime.
2. The basic operations are: finding the card in the deck array and putting into its place in the arrays.
3. Since input sizes never change, the worst-case runtime is the same as the average runtime: O(1).

4) Exercise 4

1. It should be an algorithm that involves no sorting that would find common songs between the lists would compare every song in the first list to every song in the second.
   1. In the worst case, there would be no matching songs from either lists so Big O runtime would be O(n\*m). The basic operations that contributes to the runtime are all the comparisons (each song from the first playlist with the second) that would have to be done.
   2. In the best case, the songs from each list would match immediately that causes a runtime of O(min(m,n)^2). The bookkeeping that would be necessary is storing each of these songs that are present in both lists.
2. Using mergesort to sort each array and then combine the two arrays into a singular array.
   1. The worst case is O(mlog m + nlog n) which accounts for not finding a single matched pair from the entire array.
   2. The best case is O(mlog m + nlog n) which accounts for the runtime and finding only the matched pairs from the array.

5) Exercise 5

1. My algorithm creates a new array that has the cumulative distances for each exit from the starting point. To do this, I would create a new array. For each exit, I would add its own distance from the previous exit and the accumulated distance from that previous exit to the beginning. For this part, the runtime becomes O(n).

After that, when the algorithm is called to find the distance between two exits, it’s necessary to subtract the associated respective cumulative distances between the two input exit points to find the distance between those exits. This would result the runtime of O(1) since you are only calculating the difference between two exits.