ASSIGNMENT 6

1 - A - a

Note: The way insertion sort works is that it will record each value, then swap it if needed to sort it with the current stack. Effectively, it takes a new value / position and inserts it into the right spot.

For our problem we will get:

1. **I** L O V E A L G O R I T H M S // recording I
2. **I L** O V E A L G O R I T H M S // recording L → I L is sorted, no swap needed
3. **I L O** V E A L G O R I T H M S // recording O → Sorted, no swap
4. **I L O V** E A L G O R I T H M S // recording V → Sorted, no swap
5. **I L O V E** A L G O R I T H M S // recording E → Not sorted, begin swap
6. **I L O E V** A L G O R I T H M S //
7. **I L E O V** A L G O R I T H M S //
8. **I E L O V** A L G O R I T H M S //
9. **E I L O V** A L G O R I T H M S // Sorted 🙂
10. **E I L O V A** L G O R I T H M S // recording A → Not sorted, begin swap
11. **E I L O A V** L G O R I T H M S //
12. **E I L A O V** L G O R I T H M S //
13. **E I A L O V** L G O R I T H M S //
14. **E A I L O V** L G O R I T H M S //
15. **A E I L O V** L G O R I T H M S // Sorted 🙂
16. **A E I L O V L** G O R I T H M S //
17. **A E I L O L V** G O R I T H M S //
18. **A E I L L O V G O R I T H M S // L processed**
19. **A E I L L O V G** O R I T H M S //
20. **A E I L L O G V** O R I T H M S //
21. **A E I L L G O V** O R I T H M S //
22. **A E I L G L O V** O R I T H M S //
23. **A E I G L L O V** O R I T H M S //
24. **A E G I L L O V O R I T H M S // G processed**
25. **A E G I L L O V O** R I T H M S //
26. **A E G I L L O O V** R I T H M S //

27. **A E G I L L O O V R** I T H M S //
28. **A E G I L L O O R V I T H M S // R processed**
29. **A E G I L L O O R V I** T H M S //
30. **A E G I L L O O R I V** T H M S //
31. **A E G I L L O O I R V** T H M S //
32. **A E G I L L O I O R V** T H M S //
33. **A E G I L L I O O R V** T H M S //
34. **A E G I L I L O O R V** T H M S //
35. **A E G I I L L O O R V T H M S // I processed**
36. **A E G I I L L O O R V T** H M S //
37. **A E G I I L L O O R T V H M S // T processed**
38. **A E G I I L L O O R T V H** M S //
39. **A E G I I L L O O R T H V** M S //
40. **A E G I I L L O O R H T V** M S //
41. **A E G I I L L O O H R T V** M S //
42. **A E G I I L L O H O R T V** M S //
43. **A E G I I L L H O O R T V** M S //
44. **A E G I I L H L O O R T V** M S //
45. **A E G I I H L L O O R T V** M S //
46. **A E G I H I L L O O R T V** M S //
47. **A E G H I I L L O O R T V M S // H processed**
48. **A E G H I I L L O O R T V M** S //
49. **A E G H I I L L O O R T M V** S //
50. **A E G H I I L L O O R M T V** S //
51. **A E G H I I L L O O M R T V** S //
52. **A E G H I I L L O M O R T V** S //
53. **A E G H I I L L M O O R T V S // M processed**
54. **A E G H I I L L M O O R T V S** //
55. **A E G H I I L L M O O R T S V** //

Final sorted list:

56. **A E G H I I L L M O O R S T V // S processed (final element)**

For insertion sort,

- **Best case: O(n)**
    - occurs when the array is already sorted, and no swaps will be made while processing each element
- **Average case: O(n^2)**
    - because on average each element will have to move halfway through the array
- **Worst case: O(n^2)**
    - because then the time taken to sort the list is proportional to the square of the number of elements in the list

Personal algorithm rating: 6/10 🙂

1 - A - b

Note: The way selection sort works from left to right, is that we identify our first non sorted element as our current minimum. Then we iterate until the end, and each number smaller than our initial minimum becomes our new current minimum. Once we reach the end and find the smallest / real minimal value in the unsorted part, we swap the initial minimum and the current minimum, which is now sorted and then we begin to the next element.

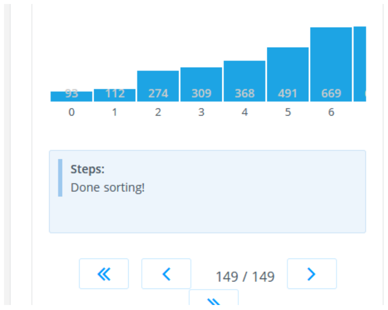
1. I L O V E A L G O R I T H M S // I is our initial minimum and current minimum
2. I L O V E A L G O R I T H M S // L is larger, so I remains our current minimum
3. I L O V E A L G O R I T H M S // O is larger, so I remains our current minimum
4. I L O V E A L G O R I T H M S // V is larger, so I remains our current minimum
5. I L O V E A L G O R I T H M S // E is smaller, and becomes our current minimum
6. I L O V E A L G O R I T H M S // A is smaller, and becomes our current minimum
7. I L O V E A L G O R I T H M S //
8. I L O V E A L G O R I T H M S //
9. I L O V E A L G O R I T H M S //
10. I L O V E A L G O R I T H M S //
11. I L O V E A L G O R I T H M S //
12. I L O V E A L G O R I T H M S //
13. I L O V E A L G O R I T H M S //
14. I L O V E A L G O R I T H M S //
15. I L O V E A L G O R I T H M S // A ends up being our final minimum value
16. A L O V E I L G O R I T H M S // A (final minimum) swaps with I (initial minimum)
17. A L O V E I L G O R I T H M S //
18. A L O V E I L G O R I T H M S //
19. A L O V E I L G O R I T H M S //
20. A L O V E I L G O R I T H M S //
21. A L O V E I L G O R I T H M S //
22. A L O V E I L G O R I T H M S //
23. A L O V E I L G O R I T H M S //
24. A L O V E I L G O R I T H M S //
25. A L O V E I L G O R I T H M S //
26. A L O V E I L G O R I T H M S //
27. A L O V E I L G O R I T H M S //
28. A L O V E I L G O R I T H M S //
29. A L O V E I L G O R I T H M S //
30. A L O V E I L G O R I T H M S //
31. A E O V L I L G O R I T H M S //
32. A E O V L I L G O R I T H M S //
33. A E O V L I L G O R I T H M S //
34. A E O V L I L G O R I T H M S //
35. A E O V L I L G O R I T H M S //

36. A E O V L I L G O R I T H M S //
37. A E O V L I L G O R I T H M S //
38. A E O V L I L G O R I T H M S //
39. A E O V L I L G O R I T H M S //
40. A E O V L I L G O R I T H M S //
41. A E O V L I L G O R I T H M S //
42. A E O V L I L G O R I T H M S //
43. A E O V L I L G O R I T H M S //
44. A E O V L I L G O R I T H M S //
45. A E G V L I L O O R I T H M S //
46. A E G V L I L O O R I T H M S //
47. A E G V L I L O O R I T H M S //
48. A E G V L I L O O R I T H M S //
49. A E G V L I L O O R I T H M S //
50. A E G V L I L O O R I T H M S //
51. A E G V L I L O O R I T H M S //
52. A E G V L I L O O R I T H M S //
53. A E G V L I L O O R I T H M S // Dependent on implementation I believe? But should skip I
54. A E G V L I L O O R I T H M S //
55. A E G V L I L O O R I T H M S //
56. A E G V L I L O O R I T H M S //
57. A E G V L I L O O R I T H M S //
58. A E G H L I L O O R I T V M S //

Okay this will take a while so I shall stop being so verbose:

59. A E G H I L L O O R I T V M S //
60. A E G H I I L O O R L T V M S //
61. A E G H I I L O O R L T V M S // Should skip L and remain in place
62. A E G H I I L L O R O T V M S //
63. A E G H I I L L M R O T V O S //
64. A E G H I I L L M O O T V R S //
65. A E G H I I L L M O O T V R S //
66. A E G H I I L L M O O R V T S //
67. A E G H I I L L M O O R S T V //
68. A E G H I I L L M O O R S T V //

Final sorted list (after well over 100 likely iterations, a calculator that calculates this from right to left gave me 149 steps):



Steps:
Done sorting!

69. A E G H I I L L M O O R S T V //

For selection sort,

- **Best case: O(n^2)**
    - Occurs when array is already sorted, however it still iterates through the entire array and makes no swaps.
- **Average case: O(n^2)**
    - Like the best and worst case, iterates through the entire array and makes no swaps.
- **Worst case: O(n^2)**
    - Occurs when array is completely unsorted, however it still iterates through the entire array and makes swaps which is not much different from the best case.

Personal algorithm rating: 3/10

1 - A - c

Note: The way bubble sort works is that for each iteration, it goes through each pairing that is linked throughout the array, and swaps the positions of the two if they are unsorted. After such iteration occurs, it will continue on until no swaps need to be made at all. The idea is that the largest value will be sorted at the end after each iteration guaranteed.

For our problem, the first full iteration will look like:

1. I L O V E A L G O R I T H M S //
2. I L O V E A L G O R I T H M S //
3. I L O V E A L G O R I T H M S //
4. I L O V E A L G O R I T H M S //
5. I L O E V A L G O R I T H M S //
6. I L O E V A L G O R I T H M S //
7. I L O E A V L G O R I T H M S //
8. I L O E A V L G O R I T H M S //
9. I L O E A L V G O R I T H M S //
10. I L O E A L V G O R I T H M S //
11. I L O E A L G V O R I T H M S //
12. I L O E A L G V O R I T H M S //
13. I L O E A L G O V R I T H M S //
14. I L O E A L G O V R I T H M S //
15. I L O E A L G O R V I T H M S //
16. I L O E A L G O R V I T H M S //
17. I L O E A L G O R I V T H M S //
18. I L O E A L G O R I V T H M S //
19. I L O E A L G O R I T V H M S //
20. I L O E A L G O R I T V H M S //
21. I L O E A L G O R I T H V M S //
22. I L O E A L G O R I T H V M S //
23. I L O E A L G O R I T H M V S //
24. I L O E A L G O R I T H M V S //
25. I L O E A L G O R I T H M S V //

First iteration complete. We have guaranteed one sorted value at the end.

26. I L O E A L G O R I T H M S V //

Second iteration:

27. I L O E A L G O R I T H M S V //
28. I L E O A L G O R I T H M S V //
29. I L E A O L G O R I T H M S V //
30. I L E A L O G O R I T H M S V //

31. I L E A L G O O R I T H M S V //
32. I L E A L G O O I R T H M S V //
33. I L E A L G O O I R H T M S V //
34. I L E A L G O O I R H M T S V //
35. **I L E A L G O O I R H M S T V //**

Third iteration:

36. I E L A L G O O I R H M S T V //
37. I E A L L G O O I R H M S T V //
38. I E A L G L O O I R H M S T V //
39. I E A L G L O I O R H M S T V //
40. I E A L G L O I O H R M S T V //
41. **I E A L G L O I O H M R S T V //**

Fourth iteration:

42. E I A L G L O I O H M R S T V //
43. E A I L G L O I O H M R S T V //
44. E A I G L L O I O H M R S T V //
45. E A I G L L I O O H M R S T V //
46. E A I G L L I O H O M R S T V //
47. **E A I G L L I O H M O R S T V //**

Fifth iteration:

48. A E I G L L I O H M O R S T V //
49. A E G I L L I O H M O R S T V //
50. A E G I L I L O H M O R S T V //
51. A E G I L I L H O M O R S T V //
52. **A E G I L I L H M O O R S T V //**

Sixth iteration:

53. A E G I I L L H M O O R S T V //
54. **A E G I I L H L M O O R S T V //**

Seventh iteration:

55. **A E G I I H L L M O O R S T V //**

Eigth iteration:

56. **A E G I H I L L M O O R S T V //**

Ninth iteration:

57. **A E G H I I L L M O O R S T V //**

For bubble sort,

- **Best case: O(n)**
    - Occurs when array is already sorted. It just iterates through each pair once and makes no swaps. Effectively, this is n elements of turns.
- **Average case: O(n^2)**
    - On average for each element, there will be n/2 comparisons and swaps
- **Worst case: O(n^2)**
    - Occurs when array is sorted in reverse order or when the largest element occurs at the beginning of the array (or smallest if doing an inverse bubble sort). It will make n-1 passes through the array and perform n-1 comparisons and swaps each pass.

Personal algorithm rating: 5/10 😀 it was kind of fun but doesn’t look as effective to me

1 - B

Note: this algorithm goes through gaps which can be chosen or decided by n/2. Then it will iterate through the two elements between these gaps until it reaches then end. Then the gap decreases. It works similar to insertion sort otherwise.

For this we will choose n/2, but possible selections could be gaps of 5, 3, and 1 for efficiency.

Original Array:
E A S Y S H E L L S O R T Q U E S T I O N

Array iteration with gap 10:
E A S Y S H E L L S O R T Q U E S T I O N
E A S Y S H E L L S O R T Q U E S T I O N
E A S Y S H E L L S O R T Q U E S T I O N
E A S Q S H E L L S O R T Y U E S T I O N
E A S Q S H E L L S O R T Y U E S T I O N
E A S Q S E E L L S O R T Y U H S T I O N
E A S Q S E E L L S O R T Y U H S T I O N
E A S Q S E E L L S O R T Y U H S T I O N
E A S Q S E E L I S O R T Y U H S T L O N
E A S Q S E E L I O O R T Y U H S T L S N
E A S Q S E E L I O N R T Y U H S T L S O

Array iteration with gap 5:
E A S Q S E E L I O N R T Y U H S T L S O
E A S Q S E E L I O N R T Y U H S T L S O
E A L Q S E E S I O N R T Y U H S T L S O
E A L I S E E S Q O N R T Y U H S T L S O
E A L I O E E S Q S N R T Y U H S T L S O
E A L I O E E S Q S N R T Y U H S T L S O
E A L I O E E S Q S N R T Y U H S T L S O
E A L I O E E S Q S N R T Y U H S T L S O
E A L I O E E S Q S N R T Y U H S T L S O
E A L I O E E S Q S N R T Y U H S T L S O
E A L I O E E S Q S H R T Y U N S T L S O
E A L I O E E S Q S H R T Y U N S T L S O
E A L I O E E S Q S H R T Y U N S T L S O
E A L I O E E S L S H R T Q U N S T Y S O
E A L I O E E S L S H R T Q S N S T Y U O

```
E A L I O E E S L S H R T Q S N S T Y U O
```

Array iteration with gap 2 (could fix rounding to change this to 3):

```
E A L I O E E S L S H R T Q S N S T Y U O
E A L I O E E S L S H R T Q S N S T Y U O
E A L I O E E S L S H R T Q S N S T Y U O
E A L E O I E S L S H R T Q S N S T Y U O
E A E E L I O S L S H R T Q S N S T Y U O
E A E E L I O S L S H R T Q S N S T Y U O
E A E E L I L S O S H R T Q S N S T Y U O
E A E E L I L S O S H R T Q S N S T Y U O
E A E E H I L S L S O R T Q S N S T Y U O
E A E E H I L R L S O S T Q S N S T Y U O
E A E E H I L R L S O S T Q S N S T Y U O
E A E E H I L Q L R O S T S S N S T Y U O
E A E E H I L Q L R O S S S T N S T Y U O
E A E E H I L N L Q O R S S T S S T Y U O
E A E E H I L N L Q O R S S S S T T Y U O
E A E E H I L N L Q O R S S S S T T Y U O
E A E E H I L N L Q O R S S S S T T Y U O
E A E E H I L N L Q O R S S S S T T Y U O
E A E E H I L N L Q O R O S S S S T T U Y
```

Array iteration with gap 1 (essentially becomes insertion sort here):

```
A E E E H I L N L Q O R O S S S S T T U Y
A E E E H I L N L Q O R O S S S S T T U Y
A E E E H I L N L Q O R O S S S S T T U Y
A E E E H I L N L Q O R O S S S S T T U Y
A E E E H I L N L Q O R O S S S S T T U Y
A E E E H I L N L Q O R O S S S S T T U Y
A E E E H I L N L Q O R O S S S S T T U Y
A E E E H I L L N Q O R O S S S S T T U Y
A E E E H I L L N Q O R O S S S S T T U Y
A E E E H I L L N O Q R O S S S S T T U Y
A E E E H I L L N O Q R O S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
A E E E H I L L N O O Q R S S S S T T U Y
```

Array is now sorted:
A E E E H I L L N O O Q R S S S S T T U Y

For shellsort,
I suppose it depends on chosen gaps, but for n/2 lets say:

- **Best case: O(n log(n))**
    - Occurs when array is already sorted. Normally you'd expect it to be O(n), but it is O(n^2) because it makes n-1 passes through the array where each pass requires n-1 comparisons. However no swaps will be made.
- **Average case: O(n log(n)^2)**
    - On average for each element, it can range between nlogn and n^2 due to the chosen gaps.
- **Worst case: O(n log(n)^2)**
    - Occurs when array is sorted in reverse order. Closer to O(n^2) than n log(n).

Personal algorithm rating: 1.5/10 🙁 looks inefficient and also very tedious to learn

2 - A - a

Top down merge sort is when it keeps pairing off and then it reconvenes.

I L O V E A L G O R I T H M S
// 15 characters, so the split gives odd part to the first subarray.

```
I L O V E A L G                  |                 O R I T H M S
I L O V       |      E A L G     |      O R I T       |      H M S
I L  |  O V  |  E A  |  L G      |      O R  |  I T  |  H M  |  S
I|L  |O|V  |E|A  |L|G            |      O|R  |I|T|  H|M  |S
I L  |  O V  |  A E  |  G L      |      O R  |  I T  |  H M  |  S
I L  |  O V  |  A E  |  G L      |      O R  |  I T  |  H M  |  S
I L O V       |      A E G L     |      I O R T       |      H M S
A E G I L L O V                  |      H I M O R S T
```
A E G H I I L L M O O R S T V

Personal rating: 8/10 🙂

2 - A - b

Bottom up merge sort is when the whole array pairs off and then merges together.

I L O V E A L G O R I T H M S
// 15 characters, so the last element is left unpaired.

```
I L O V E A L G O R I T H M S
I  |  L  |  O  |  V  |  E  |  A  |  L  |  G  |  O  |  R  |  I  |  T  |  H  |  M  |  S
I L  |  O V  |  A E  |  G L  |  O R  |  I T  |  H M  |  S
I L O V  |  A E G L  |  I O R T  |  H M S
A E G I L L O V  |  H I M O R S T
```
A E G H I I L L M O O R S T V

Personal rating: 8.5/10 🙂