A. REQUIREMENTS (50 points)

A1. Complex application description (10 points) Explain in which application your library will be used. What the application is doing and why the library is needed. You must include at lest one paragraph with 7 or more lines describing the requirements.

The coding library for this assignment will be used for testing the sorting of the tips acquired from "passengers" using the autonomous vehicle (AV) service. The library will require a way to store tips and hold tips so the data structure of a Bag will be used. Since this application is for testing the sorting of tips we know/control the total number of tips received to sort. Therefore, we will use a fixed capacity bag. We also must have a way to randomly generate tips so an array that can generate doubles will be needed. Since the main focus of this test application is to sort the tips, we need a way to sort the tips using an elementary sorting method. Since shell sort is faster than both insertion sort and selection sort, shell short will be used to sort the array of tips. Since we want to analyze the effectiveness of the sorting we will implement the use of a stopwatch to track time in milliseconds. We will then also need a way to record the statistics of the time so a time analysis class will be implemented to give us statistics on the effectiveness of the sort.

A2. Relevance to topic area (10 points): Why the complex application is relevant for the semester topic. Be specific. You must include at lest one paragraph with 5 or more lines.

This is relevant to the topic area of this semester's project topic of autonomous vehicles by being about sorting and analyzing the tips that customers would provide after their ride with the AV service. Tips or any money that a user would send to the AV service at some point would need to be sorted for clean analysis of the revenue produced. I am using tips for this portion since tips a genuinely smaller or a percentage of the price of the service; Therefore, I can have the randomly generated amounts be lower values($0.00 to $20.99).

A3. Library role description (10 points) Explain the role of your library for your complex applications.Imagine a different application for which the library will be useful. You must include at lest one paragraph with 7 or more lines describing the requirements.

The role of this library needs to be able to generate, store, sort, and analyze the tips from customers using the AV service. A data structure of a bag will need to be used for storing the data, since we know/control the amount of tips being generated and stored we can have the bag be fixed capacity. We also need a way to generate tips randomly for the testing so a doubles array generator will be created, as well as a toString for printing the tips. The tips need to be sorted in so will we need a sorting method to do so. A shell sort will work best because of its speed compared to selection and insertion. Then we need a way to track and analyse time in milliseconds, so a stopwatch and a time analysis class will be needed. We also need a way to analyze the sort so everything will come together in a Shell sort analysis class that can give us the statistics of the sorting of the tips.This library although used for generating, storing, sorting, and analyzing tips, this can also be used for analyzing ride revenues, refunds, and other monetarily based items.

A4. Library requirements description (20 points) Explain the API (interface) or your library. Each method in the API must be explained. Be sure that your API will cover the normal use of the library as described above.

All of the features in the complex application have "m2bc…" infront of it, so I will just say the basic name of the feature. An example of this would be me referring to m2bcShellsort as Shellsort, but in the actual library it would be m2bcShellsort. Now that is cleared up I can discuss the library. I first created a Bag interface that gives us the ability to use a Fixed Capacity Bag class whose main function for this library is its size() and the getCapacity() methods. I then created an ArrayUtility that can randomly generate doubles rounded to the nearest hundredth place, (like tip money). The ArrayUtility class also has a toString for doubles (the tips) that adds a "$" to the doubles after being printed. I have a Shellsort class that implements the use of a shell sort to sort the tips at extremely fast rates. I then have a testing method for the randomly generated array and the shell sort sorting method called Shellsort test, this demonstrates what the array is generating and how the shell sort is working. I then created a Stopwatch class that is able to record time in milliseconds, this is then used in a TimeAnalysis class that records the temporal stats of the sorting of the arrays. This is then used in the ShellSortAnalysis class that which has a meanTime method using features in the Time analysis class and the stopwatch class. Then everything comes together for to be tested, analyzed and properly structured in the printMeanExecutionTimeGrowthTable.

B. IMPLEMENTATION (50 points)

B1. Method Requirements (10 points) Explain in your own words detailed requirements for that method

This method takes any double[] of tip amounts and sorts it in place from smallest to largest. It behaves correctly for all sizes, including the edge cases of an empty array or a single element. Because these numbers represent money, the values are already rounded to two decimal places when they're created; the sort compares the numeric values directly (not their string or "$"-formatted versions), so $10.00 correctly comes after $9.99. Shell sort isn't stable, so equal tips might not keep their original relative order, and that's acceptable for this use case. It's designed for in-memory batches (thousands up to the low hundreds of thousands of items) with typical sub-quadratic performance on random data. The algorithm is in-place, using only constant extra space, and it's deterministic: the same input will always produce the same sorted result.

B2. Method Implementation explanation (15 points) Explain how you implemented in the method the requirements, be specific with references to the code.

The sorter uses Knuth's 3x+1 gap sequence (1, 4, 13, 40…etc) growing up to about a third of the array length, then stepping down by dividing the gap by three each round. That sequence is a classic pick for Shell sort because it behaves well in practice on random data. For each gap size, the algorithm runs an insertion-style pass over the interleaved elements: starting at index h, it takes the current value, shifts earlier items that are larger by h positions, and drops the value into its proper spot. The big win is that large out-of-place elements get moved closer to where they belong while the gap is large, so the final pass with h = 1 is essentially a quick clean-up on an almost sorted array. Once the gap drops to zero, we stop, then the array is sorted. We compare doubles numerically (not as strings), which is exactly what we want for money. Since the

generator already rounds to two decimals, you avoid floating-points and when you print with toStringDouble(...) you get consistent "$%.2f" output.

B3. Design a test for your method (10 points)

For testing, start with m2bcShellSortTest.testShellSortDouble() and then add a few focused checks. Hit the edge cases first whic is an empty array, a single element, and an array where every value is the same to make sure nothing weird happens at the boundaries. Next, verify small, easy-to-reason-about inputs: a two-element array that's already sorted, the same pair in reverse, and a short mixed example (say five tips with a couple of duplicates) to confirm ties sort correctly. For a more realistic run, generate a random 10-element array of tips in the range [0.00, 20.99] and print it before and after using m2bcArrayUtility.toStringDouble(...) so you can see the currency formatting. After each run, automatically check that the result is non-decreasing—write a tiny isSorted(double[]) so you're not eyeballing it. Finally, run the suite multiple times to cycle through different random samples; you shouldn't see any exceptions, and the output should always come back in non-decreasing order.

B4. Explain the results of your test (include screenshot here and at the end) (15 points) be specific with references to the values obtained.

```
Creative Assignment - by Matthew Novak
Executed on: Wed Sep 24 21:23:26 EDT 2025
_____
Generating and sorting varying amounts of tips recieved
------------------------------------------------------------
Random array of 0 tips, from $0.00 to $20.99:
{}
Sorted array of tips :
{}
------------------------------------------------------------
Random array of 1 tip, from $0.00 to $20.99:
{$3.66}
Sorted array :
{$3.66}
------------------------------------------------------------
Random array of 2 tips, from $0.00 to $20.99:
{$18.19, $4.16}
Sorted array of tips :
{$4.16, $18.19}
------------------------------------------------------------
Random array of 5 tips, from $0.00 to $20.99:
{$12.06, $15.62, $2.02, $19.09, $4.00}
Sorted array of tips :
{$2.02, $4.00, $12.06, $15.62, $19.09}
------------------------------------------------------------
Random array of 10 tips, from $0.00 to $20.99:
{$0.10, $9.61, $4.68, $12.00, $16.87, $4.76, $3.65, $7.78, $2.45, $19.83}
Sorted array of tips :
{$0.10, $2.45, $3.65, $4.68, $4.76, $7.78, $9.61, $12.00, $16.87, $19.83}
------------------------------------------------------------
```

As discussed in the previous section you can see that the arrays of tips go from 0 to 10 elements all going from $0.00 to $20.99. You for first see the sorting of the shell work in the third row or the 2 tip array, where $18.19 is before $4.16 and then switched to the correct order of $4.16, $18.19. Then you can see the overall works of the shell sort with the 10 tip array where although the lowest and highest values are actually in the correct spot in the random array the middle portion of the array is then sorted. You can see this with the second element in the array $9.61 which is moved to the 7th position of the array and $2.45 is then moved to the second position of the array.

C. ANALYSIS (50 points)

C1. Experimental time analysis (30 points) Perform an experimental time analysis for your method. Include the raw data and a graph. Infer the experimental order of growth.

```
Creative Assignment- Main Test Analysis - by Matthew Novak
Executed on: Wed Sep 24 21:25:45 EDT 2025

_____________________________________
Mean execution time growth table for sorting tips
  - Method: Selection sort of N doubles (tips)
  - Sample size for time estimation: 10 tips
|-----------|--------|------|------|------------------|
|         N |   Mean |  Min |  Max |         CI 99.9% |
|-----------|--------|------|------|------------------|
|     10000 |    1.6 |    1 |    3 | (   0.3,    2.9) |
|     15000 |    1.8 |    1 |    2 | (   1.2,    2.4) |
|     20000 |    2.2 |    2 |    3 | (   1.6,    2.8) |
|     25000 |    2.5 |    2 |    3 | (   1.7,    3.3) |
|     30000 |    3.2 |    3 |    4 | (   2.6,    3.8) |
|     35000 |    4.0 |    4 |    4 | (   4.0,    4.0) |
|     40000 |    4.8 |    4 |    5 | (   4.2,    5.4) |
|     45000 |    5.2 |    5 |    6 | (   4.6,    5.8) |
|     50000 |    6.0 |    5 |    7 | (   5.3,    6.7) |
|     55000 |    6.7 |    6 |    7 | (   6.0,    7.4) |
|     60000 |    7.7 |    7 |    8 | (   7.0,    8.4) |
|     65000 |    8.0 |    8 |    8 | (   8.0,    8.0) |
|     70000 |    8.8 |    8 |    9 | (   8.2,    9.4) |
|     75000 |    9.6 |    9 |   10 | (   8.8,   10.4) |
|     80000 |   11.2 |   10 |   12 | (  10.0,   12.4) |
|     85000 |   12.7 |   12 |   16 | (  10.4,   15.0) |
|     90000 |   12.1 |   11 |   14 | (  10.4,   13.8) |
|     95000 |   12.6 |   12 |   13 | (  11.8,   13.4) |
|    100000 |   14.0 |   13 |   15 | (  13.0,   15.0) |
|-----------|--------|------|------|------------------|
```

You can see the as more doubles or tips are introduced to the array then the amount of time also increases. You can see this with and array of 10,000 taking 1.6 milliseconds and 100,000 array taking 14 milliseconds. Watching the arrays increment 5,000 in length you can see the slow increase of time. It appears to be roughly a little less than 1 millisecond per 10,000 elements.

C2. Theoretical time analysis (10 points) Perform a naive theoretical time analysis trying to guess which will be the order of growth.

Shell sort's running time really depends on the gap sequence you choose. Using Knuth's 3x+1 gaps (1, 4, 13, 40, …), it typically behaves better than quadratic on random data; in practice you'll often see growth close to N^1.5. If the array is already nearly sorted, the last pass with gap 1 is cheap, so it can be quicker still. The flip side is that Shell sort isn't stable, and the exact worst-case analysis is tricky and not something most courses pin down tightly. For an "elementary methods" write-up, a fair summary is: faster than $O(N^2)$ in general, and commonly around N^1.5 with Knuth gaps.

C3. Space analysis (10 points) Perform a theoretical space analysis for your method assuming the textbook space values for elementary structures.

Space-wise, this sorter works entirely in place. It keeps only a few scalars, the current gap h, some loop counters, and a single temp slot while shifting elements, and it never allocates any helper arrays. So the extra memory stays constant ($\Theta(1)$). The only thing that scales is the data you give it: the input array itself is $\Theta(N)$ simply because it holds N values. There's no recursion, either, so the call stack doesn't grow; you just have a small, fixed stack frame the whole time.

D. CODE AND EXECUTION (50 points)

```
Creative Assignment - by Matthew Novak
Executed on: Wed Sep 24 21:23:26 EDT 2025
_____________________________________________
Generating and sorting varying amounts of tips recieved
-----------------------------------------------------------
Random array of 0 tips, from $0.00 to $20.99:
{}
Sorted array of tips :
{}
-----------------------------------------------------------
Random array of 1 tip, from $0.00 to $20.99:
{$3.66}
Sorted array :
{$3.66}
-----------------------------------------------------------
Random array of 2 tips, from $0.00 to $20.99:
{$18.19, $4.16}
Sorted array of tips :
{$4.16, $18.19}
-----------------------------------------------------------
Random array of 5 tips, from $0.00 to $20.99:
{$12.06, $15.62, $2.02, $19.09, $4.00}
Sorted array of tips :
{$2.02, $4.00, $12.06, $15.62, $19.09}
-----------------------------------------------------------
Random array of 10 tips, from $0.00 to $20.99:
{$0.10, $9.61, $4.68, $12.00, $16.87, $4.76, $3.65, $7.78, $2.45, $19.83}
Sorted array of tips :
{$0.10, $2.45, $3.65, $4.68, $4.76, $7.78, $9.61, $12.00, $16.87, $19.83}
-----------------------------------------------------------

Creative Assignment- Main Test Analysis - by Matthew Novak
Executed on: Wed Sep 24 21:25:45 EDT 2025
_____________________________________________
Mean execution time growth table for sorting tips
  - Method: Selection sort of N doubles (tips)
  - Sample size for time estimation: 10 tips
```

N	Mean	Min	Max	CI 99.9%
10000	1.6	1	3	(0.3, 2.9)
15000	1.8	1	2	(1.2, 2.4)
20000	2.2	2	3	(1.6, 2.8)
25000	2.5	2	3	(1.7, 3.3)
30000	3.2	3	4	(2.6, 3.8)
35000	4.0	4	4	(4.0, 4.0)
40000	4.8	4	5	(4.2, 5.4)
45000	5.2	5	6	(4.6, 5.8)
50000	6.0	5	7	(5.3, 6.7)
55000	6.7	6	7	(6.0, 7.4)
60000	7.7	7	8	(7.0, 8.4)
65000	8.0	8	8	(8.0, 8.0)
70000	8.8	8	9	(8.2, 9.4)
75000	9.6	9	10	(8.8, 10.4)
80000	11.2	10	12	(10.0, 12.4)
85000	12.7	12	16	(10.4, 15.0)
90000	12.1	11	14	(10.4, 13.8)
95000	12.6	12	13	(11.8, 13.4)
100000	14.0	13	15	(13.0, 15.0)