On Nov 9th we identified the following 3 cases we should test for the ConvertManifestToGrid function:

1) The grid returned should always have dimensions of 10 rows and 12 columns
2) Each element in the manifest should be trackable to the grid by taking looking at the corresponding row and column
3) The elements on the top two rows of the grid should always be named “UNUSED”

Results (Nov 10th):

1) Passed
2) Passed
3) Passed

Here is our test code for ConvertManifestToGrid. Having traditional unit tests is difficult for this function since the output is large 2D arrays, so instead we compare the grid to the manifest by outputting the grid.

On Nov 13th we identified the following 7 cases we should test for the Container Name Checker function.

1) Prevents “” from being passed in as a container name
2) Prevents the first character in character name from being “ “
3) Prevents non ascii characters from being in container name
4) Prevents containers names from being longer than 255 characters
5) Prevents non strings (such as int or float) from being the container name
6) Prevents “NAN” from being the container name
7) Prevents “UNUSED” from being the container name

Results (Nov 15th):

1) Passed
2) Passed
3) Passed
4) Passed
5) Passed
6) Passed
7) Passed

Here’s the testing code for the Container Name Checker. This can also be found on github under src/functions/tests.

```
import { CheckContainerName } from "./ContainerNameChecker.js";

console.log('Test 1 (regular, only letters): "Walmart"')
console.assert(CheckContainerName("Walmart") === true);

console.log('Test 2 (regular, valid chars):
"DWw?J5-+AjY6EZg2S6@_GdinA]?.RZ0,te2S#hqj,D![FDqt8/Zd](!_#2]gAnnyf(97[@xZbXK:]?XgdQ*tD
kNtigK12[2MzwP%"')
console.assert(CheckContainerName("DWw?J5-+AjY6EZg2S6@_GdinA]?.RZ0,te2S#hqj,D![FDqt8/Z
d](!_#2]gAnnyf(97[@xZbXK:]?XgdQ*tDkNtigK12[2MzwP%") === true);

console.log('Test 3 (empty string): ""')
console.assert(CheckContainerName("") === false);

console.log('Test 4 (begins with space): " Walmart"')
console.assert(CheckContainerName(" Walmart") === false);
```

```
console.log('Test 5 (invalid chars): "Wa😊lart"')
console.assert(CheckContainerName("Wa😊lart") === false);

console.log('Test 6 (valid chars but too long):
"gBPY_S_U8k1J2zp{4-}VCnV;cX9H_qj?49Sig[[0aX&j&xx_@rEg8:ANhL(,#vP?.}FV#A!VqNpK/hxR9UTpb
=.h_u{AnrzW2i,x,Xfu#kgm/qxip/h.c&M+ka}cH?Bb54:bU-+!BZamHkx_.P#8b]/vecjaLPGZ]dQ(%WV/,(U
2W+5yif;y]Ff9rWa$zfwr.ShVJ]86hn,T9a?7(])R&;c3?Y7E3Kp*B.5daZ7ku318.bUv,U+=1-L4SP(!)pMx"
')
console.assert(CheckContainerName("gBPY_S_U8k1J2zp{4-}VCnV;cX9H_qj?49Sig[[0aX&j&xx_@rE
g8:ANhL(,#vP?.}FV#A!VqNpK/hxR9UTpb=.h_u{AnrzW2i,x,Xfu#kgm/qxip/h.c&M+ka}cH?Bb54:bU-+!B
ZamHkx_.P#8b]/vecjaLPGZ]dQ(%WV/,(U2W+5yif;y]Ff9rWa$zfwr.ShVJ]86hn,T9a?7(])R&;c3?Y7E3Kp
*B.5daZ7ku318.bUv,U+=1-L4SP(!)pMx") === false);

console.log('Test 7 (not string): 5')
console.assert(CheckContainerName(5) === false);

console.log('Test 8 (named NAN): "NAN"')
console.assert(CheckContainerName("NAN") === false);

console.log('Test 9 (named UNUSED): "UNUSED"')
console.assert(CheckContainerName("UNUSED") === false);
```

On Nov 14th we identified the following 3 cases we should test for the CostBoxToBox function. This is used as a subroutine by load-unload and balance.

1) Boxes are in columns right next to each other
2) No box in starting column
3) Boxes on the same column

Results (Nov 14th)

1) Passed
2) Passed
3) Passed

Here is the testing code for CostBoxToBox

```
const {MovementCost ,CostBoxtoBox, CostCranetoBox} = require('./CostFunctionBalance');

//CostBox2Box tests
console.log('TestSuite: CostBoxtoBox, Test: 1a (regular)');
console.assert(CostBoxtoBox(4, 9, [3, 7, 1, 10, 6, 0, 4, 9, 8, 2, 5, 7]) === 14);

console.log('TestSuite: CostBoxtoBox, Test: 1b (regular)');
console.assert(CostBoxtoBox(9, 4, [3, 7, 1, 10, 6, 0, 4, 9, 8, 2, 5, 7]) === 14);

console.log('TestSuite: CostBoxtoBox, Test: 2a (regular)');
console.assert(CostBoxtoBox(0, 11, [0, 10, 5, 3, 8, 7, 10, 1, 6, 9, 2, 4]) === 14);

console.log('TestSuite: CostBoxtoBox, Test: 2b (regular)');
console.assert(CostBoxtoBox(11, 0, [0, 10, 5, 3, 8, 7, 10, 1, 6, 9, 2, 4]) === 16);

console.log('TestSuite: CostBoxtoBox, Test: 3 (columns right next to each other)');
console.assert(CostBoxtoBox(4, 5, [2, 8, 0, 5, 9, 10, 6, 3, 10, 1, 4, 7]) === 1);

console.log('TestSuite: CostBoxtoBox, Test: 3 (no box in startcol)');
console.assert(CostBoxtoBox(5, 4, [2, 8, 0, 5, 9, 10, 6, 3, 10, 1, 4, 7]) === 9999);

console.log('TestSuite: CostBoxtoBox, Test: 4 (same column)');
console.assert(CostBoxtoBox(8, 8, [9, 1, 4, 6, 0, 10, 10, 3, 5, 2, 7, 8]) === 0);
```

On Nov 14th we identified the following 3 cases we should test for the CostCraneToBox function. This is used as a subroutine by load-unload and balance.

1) Crane not attached to box
2) Crane attached to box
3) Columns of crane and box right next to each other
4) Crane and box on the same column, but not attached

Results (Nov 14th)

1) Passed
2) Passed
3) Passed
4) Passed

Here is the testing code for CostCraneToBox:

```
//CostCrane2Box tests
console.log('TestSuite: CostCranetoBox, Test: 1 (regular, crane not attached to
box)');
console.assert(CostCranetoBox(8, 4, 4, [3, 7, 1, 10, 6, 0, 4, 9, 8, 2, 5, 7]) === 16);

console.log('TestSuite: CostCranetoBox, Test: 2 (regular, crane not attached to
box)');
console.assert(CostCranetoBox(3, 1, 8, [0, 10, 5, 3, 8, 7, 2, 1, 6, 9, 4, 10]) ===
12);

console.log('TestSuite: CostCranetoBox, Test: 3 (regular, crane attached to box)');
console.assert(CostCranetoBox(0, 2, 11, [2, 8, 0, 5, 9, 10, 6, 3, 10, 1, 4, 7]) ===
22);

console.log('TestSuite: CostCranetoBox, Test: 4 (regular, crane attached to box`)');
console.assert(CostCranetoBox(10, 8, 0, [9, 1, 4, 6, 0, 10, 7, 3, 5, 2, 8, 10]) ===
29);

console.log('TestSuite: CostCranetoBox, Test: 5 (columns right next to each other)');
console.assert(CostCranetoBox(3, 0, 2, [7, 3, 6, 2, 9, 0, 10, 5, 1, 8, 4, 10]) === 7);

console.log('TestSuite: CostCranetoBox, Test: 6 (columns right next to each other)');
console.assert(CostCranetoBox(4, 4, 5, [5, 8, 1, 10, 4, 7, 3, 9, 2, 6, 0, 10]) === 6);

console.log('TestSuite: CostCranetoBox, Test: 7a (same column, not attached)');
console.assert(CostCranetoBox(0, 0, 0, [7, 2, 9, 5, 10, 3, 8, 6, 1, 4, 0, 7]) === 7);

console.log('TestSuite: CostCranetoBox, Test: 7b (same column, attached)');
console.assert(CostCranetoBox(0, 7, 0, [7, 2, 9, 5, 10, 3, 8, 6, 1, 4, 0, 7]) === 0);
```

On Nov 14th we identified the following 3 cases we should test for the MovementCost function. This is used as a subroutine by load-unload and balance, and it incorporates the previous 2 functions (CostBoxtoBox and CostCranetoBox) as subroutines.

1) Crane and box are in columns right next to each other
2) Crane attached to box
3) Crane not attached to box
4) Crane and box on the same column, but not attached

Results (Nov 14th)

1) Passed
2) Passed
3) Passed
4) Passed

Here is the testing code for MovementCost:

```
//MovementCost tests
console.log('TestSuite: MovementCost, Test: 1');
console.assert(MovementCost(6, 1, 10, 0, [4, 9, 1, 7, 3, 5, 10, 0, 2, 8, 6, 10]) ===
34);

console.log('TestSuite: MovementCost, Test: 2');
console.assert(MovementCost(3, 1, 5, 4, [2, 8, 6, 1, 10, 3, 0, 5, 7, 4, 9, 3]) ===
13);

console.log('TestSuite: MovementCost, Test: 3');
console.assert(MovementCost(11, 9, 1, 11, [5, 0, 9, 2, 7, 10, 3, 8, 4, 1, 6, 10]) ===
40);

console.log('TestSuite: MovementCost, Test: 4');
console.assert(MovementCost(0, 0, 6, 7, [6, 4, 10, 1, 8, 3, 7, 2, 9, 0, 5, 6]) ===
20);

console.log('TestSuite: MovementCost, Test: 5');
console.assert(MovementCost(0, 0, 0, 1, [10, 3, 7, 0, 9, 5, 1, 4, 8, 6, 2, 10]) >=
9999);
```

On Nov 15th we identified the following 7 cases we should test for the balance operation with no buffer:

1) Balance an empty ship
2) Balance a ship with only 1 container that needs to be moved
3) Balance a ship with only 1 container that's in the correct place
4) Balance a full ship that is already balanced
5) Balance a full ship that is out of balance
6) Balance a ship with containers of duplicate weight
7) Balance a ship with all containers having duplicate weight

Results (Nov 23rd):

1) Passed
2) Passed
3) Passed
4) Passed
5) Fail (Will pass on cases where only a few containers need to move, but in general requires implementing the buffer)
6) Passed
7) Passed

Written tests are at end of document

On November 21st we identified the following 6 cases we should test for the load-unload operation with no buffer:

1) Run the load-unload operation on a ship that needs 1 container to be loaded
2) Run the load-unload operation on a ship that needs 1 container to be unloaded
3) Run the load-unload operation on a ship that needs no containers to be loaded or unloaded
4) Run the load-unload operation on a full ship
5) Run the load-unload operation on a ship with duplicate containers
6) Run the load-unload operation on a ship with all duplicate containers

Results (November 26th):

1) Passed
2) Passed
3) Passed
4) Fail (Will pass on cases where only a few containers need to move, but in general requires implementing the buffer)
5) Passed
6) Passed

Written tests are at end of document

On December 5th we identified the following 3 cases we should test for the SIFT operation with no buffer:

1) Run SIFT on a ship with 1 container that needs to be moved
2) Run SIFT on a ship with 1 container that's in the correct place
3) Run SIFT on a full ship

Results (December 9th):

1) Passed
2) Passed
3) Fail (Will pass on cases where only a few containers need to move, but in general requires implementing the buffer)

Written tests are at end of document

On December 9th we identified the following 7 cases we should test for the balance operation with the buffer:

1) Balance a ship with less than half its capacity filled with containers
2) Balance a ship with more than half its capacity filled with containers
3) Balance an empty ship
4) Balance a ship with only 1 container that needs to be moved
5) Balance a ship with only 1 container that's in the correct place
6) Balance a full ship that is already balanced
7) Balance a full ship that is out of balance
8) Balance a ship with duplicate weighted containers

Results (December 10th):

1) Passed
2) Passed
3) Passed
4) Passed
5) Passed
6) Passed
7) Passed
8) Passed

Written tests are at end of document

On December 9th we identified the following cases we should test for the MoveContainerMessage subroutine that's used by the frontend to give directions:

1) Create message for movement of container from buffer to ship
2) Create message for movement of container from ship to ship
3) Create message for movement of container from ship to buffer
4) Create message for movement of container from buffer to buffer
5) Throw error for when trying to move container to NAN cell
6) Throw error for when trying to move a NAN cell like its a container

Results (December 9th):

1) Passed
2) Passed
3) Passed
4) Passed
5) Passed
6) Passed

This was the code for the MoveContainerMessage test cases (You can also find this in our github repo under :

```
//TEST PROGRAM for MoveContainerMessage()

let manifest = new ManifestGridTranslator()
import fs from 'fs';

// Your code using the `fs` module
const manifestString = fs.readFileSync("../../data/ShipCase1.txt", "utf8");   //Stores the file into a string
let grid = manifest.ConvertManifestToGrid(manifestString)
let x = new BalanceOperation(grid)
let craneX = 27
let craneY = 0
let boxCol = 23

//MoveContainerMessage(originalY, i, finalY, j)

console.log("TEST1")
let testString = x.MoveContainerMessage(5, 23, 9, 30);
let expectedString = "Move the container on the buffer at (1, 24) to the ship at (1, 4)"
if (!(testString === expectedString)) {
console.log("Error:\t" + testString + "\t!=\t" + expectedString);
```

```
}

console.log("TEST2")
testString = x.MoveContainerMessage(9, 31, 9, 30);
expectedString = "Move the container on the ship at (1, 5) to (1, 4)"
if (!(testString === expectedString)) {
console.log("Error:\t" + testString + "\t!=\t" + expectedString);
}

console.log("TEST3")
testString = x.MoveContainerMessage(9, 31, 4, 0);
expectedString = "Move the container on the ship at (1, 5) to the buffer at (2, 1)"
if (!(testString === expectedString)) {
console.log("Error:\t" + testString + "\t!=\t" + expectedString);
}

console.log("TEST4")
testString = x.MoveContainerMessage(0, 25, 4, 0); //This should throw an error

console.log("TEST5")
testString = x.MoveContainerMessage(4, 0, 0, 25); //This should throw an error
```

On December 9th we identified the following 5 cases we should test for the SIFT operation with the buffer:

1) Run SIFT on a ship with less than half its capacity filled with containers
2) Run SIFT on a ship with more than half its capacity filled with containers
3) Run SIFT on a ship with 1 container that needs to be moved
4) Run SIFT on a ship with 1 container that's in the correct place
5) Run SIFT on a full ship

Results (December 11th):

1) Passed
2) Sometimes passes (there are some cases where SIFT will run out of memory. However, if more memory is allocated for the operation, it will finish with the correct operations list)
3) Passed
4) Passed
5) Likely fails (Only passes if the ship is already in SIFT position. Otherwise, it will use run out of memory, unless allocated enough.

On December 9th we identified the following 5 cases we should test for the Load-Unload operation with the buffer:

1) Run load-unload on a ship with less than half its capacity filled with containers
2) Run load-unload on a ship with more than half its capacity filled with containers
3) Run the load-unload operation on a ship that needs 1 container to be loaded
4) Run the load-unload operation on a ship that needs 1 container to be unloaded
5) Run the load-unload operation on a ship that needs no containers to be loaded or unloaded
6) Run the load-unload operation on a full ship
7) Run the load-unload operation on a ship with duplicate containers
8) Run the load-unload operation on a ship with all duplicate containers

Results (December 12th):

1) Passed
2) Passed
3) Passed
4) Passed
5) Passed
6) Passed
7) Passed
8) Passed

Extra Test Evidence

Because the balance, load-unload, and SIFT operations are such large modules, they are difficult to test, its difficult to write test programs for them. Instead, we tested these modules by hand, by calculating ourselves the optimal path on paper and compare this to the path we get from the program. Here are some of these tests

Personal Example 1: Cost = 15

	0	1	2	3	4	5	6	7	8	9	10	11	
0													0
1	O												1
2													2
3													3
4													4
5													5
6													6
7													7
8					10		15	5					8
9			10	15	20	~~15~~	10	15					9
	0	1	2	3	4	5	6	7	8	9	10	11	

Personal Example 2: Cost = 15

	0	1	2	3	4	5	6	7	8	9	10	11	
0													0
1	O												1
2													2
3													3
4													4
5													5
6													6

7													**7**
8							20						**8**
9	X	X	10	~~20~~	5	35	20	10			X	X	**9**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	

Mr. Keogh Example 1 (ShipCase1.txt): Cost = 14

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	
0													**0**
1	O												**1**
2													**2**
3													**3**
4													**4**
5													**5**
6													**6**
7													**7**
8													**8**
9	X	99	~~100~~				100					X	**9**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	

Mr. Keogh Example 2 (ShipCase2.txt): Cost = 23

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	
0													**0**
1	O												**1**
2													**2**
3													**3**
4													**4**
5													**5**
6													**6**

7	40												**7**
8	X	50										X	**8**
9	X	X	X	~~120~~		35	120		~~35~~	X	X	X	**9**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	

Mr. Keogh Example 3 (ShipCase3.txt): Cost = 25

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	
0													**0**
1	O												**1**
2													**2**
3													**3**
4													**4**
5													**5**
6													**6**
7													**7**
8	~~9041~~	10					100						**8**
9	10001	500	600	~~100~~			9041						**9**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	

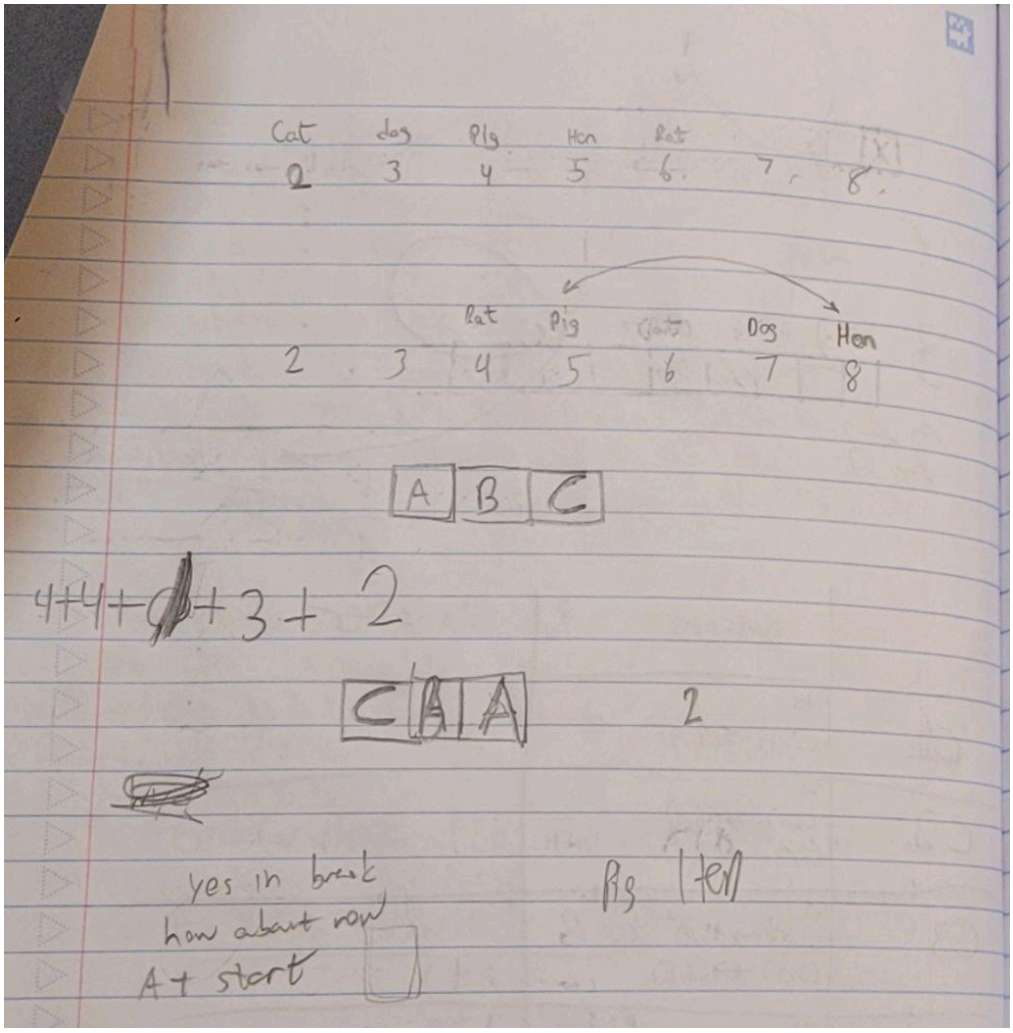
Mr. Keogh Example 4 (ShipCase4.txt): Cost = 43

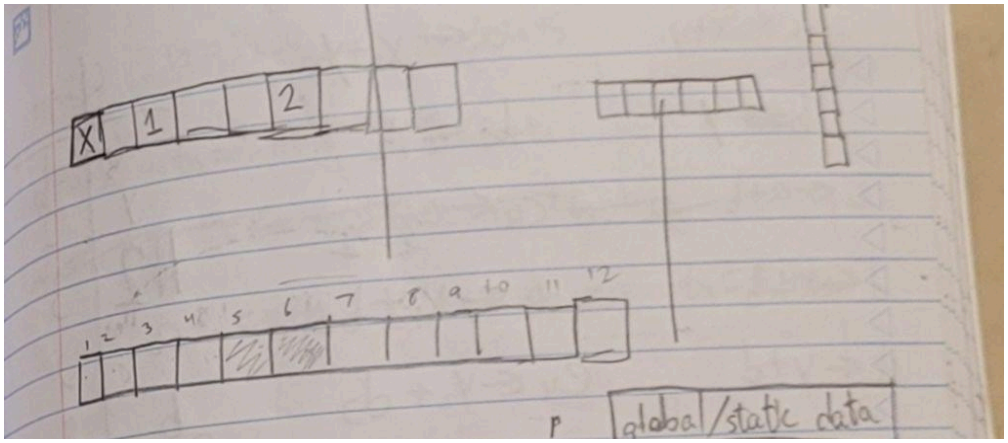
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	
0													**0**
1	O												**1**
2					~~3044~~								**2**
3					~~1100~~								**3**
4					~~2020~~								**4**
5					~~10000~~								**5**
6				2020	2011								**6**

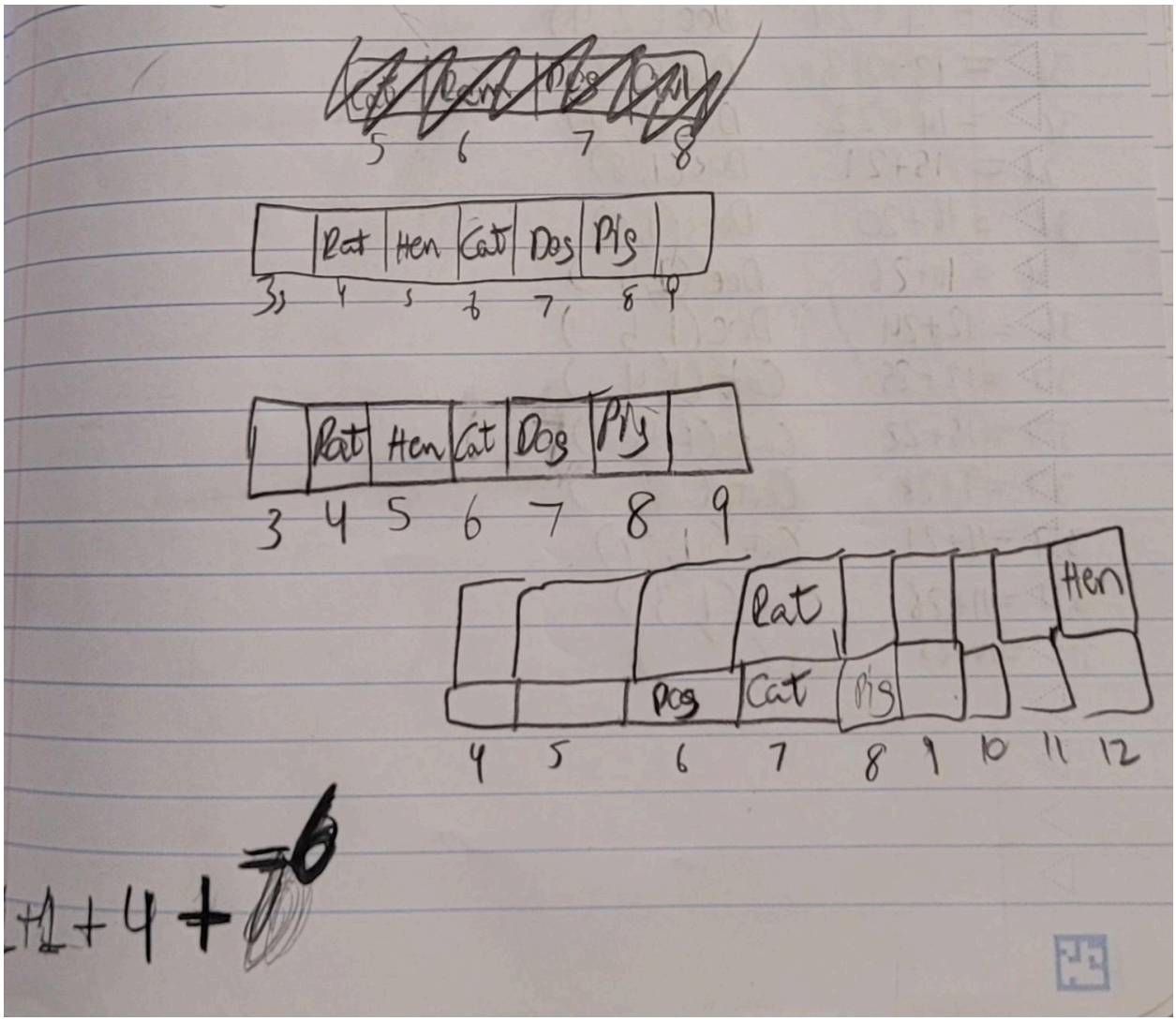
7				1100	2007								**7**
8	X			3044	2000		10000					X	**8**
9	X	X	X	X	X	X	X	X	X	X	X	X	**9**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	

Mr. Keogh Example 5 (ShipCase5.txt): Cost = 45

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	
0													**0**
1	O												**1**
2													**2**
3													**3**
4													**4**
5													**5**
6													**6**
7													**7**
8				1		96							**8**
9	X	~~96~~	~~8~~	~~4~~	4	~~4~~	8	4				X	**9**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	


Cat dog Pig Hen Rat
2 3 4 5 6 7 8
Rat Pig Dog Hen
2 3 4 5 6 7 8
A B C
4+4+3+2
C A A
2
yes in break
how about row
A+ start
Pig Hen


X 1 2
1 2 3 4 5 6 7 8 9 10 11 12
global/static data


Rat
Hen
Cat
Dog
Pig
3 4 5 6 7 8 9
Rat
Hen
Cat
Dog
Pig
3 4 5 6 7 8 9
Rat
Hen
Dog
Cat
Pig
4 5 6 7 8 9 10 11 12

+4+

currState.operationList is undefined

	g	h
(1,4) → (1,7)	14	3
(1,4) → (1,8)	15	3

Case 1

- Move crane to (1, 3) and move container to (1,7)
  (Estimate 14 minutes)

Move crane back to starting location at (1,9)

- Constructor initialize

Case 3 falls

1	2	3
4	5	6
7		8

Looking for
(1,9) → (1,6)
(owl) 3 to left

(3,1) → (1,23)

1+11 = 32

8+28+4 = 12+28 = 40+9 = 49

9+2+9

4+32+8 = 44+4+2 = 50

+4 +2

4+4+2

3+4+2

8+(32-5)+4

8+32-1 = 39

(1,5) → (1,6)

12+8+(32-6)+4

20+36-6 = 50

When initially at (1,5) on ship moves to (1,6) on buffer

9+

8+(32-6)+

20 29 30

4+(28-6) = 22+11 = 33

= 38

12

Acceptance testing

We proposed the following tests during our project pitch:

1) The system will produce the best order of operations list for any load/unload job in under 15 minutes. To test this, 25 different manifests are input into the system. If the system can produce a load/unload job in under 15 minutes for each of the 25 manifests, the test is accepted.

2) The system will produce the best order of operations list for any balance job in under 15 minutes. To test this, 25 different manifests are input into the system. If the system can produce a balance job in under 15 minutes for each of the 25 manifests, the test is accepted.

3) The system will save its previous state before a simulated power outage during 50/50 test trials.

4) The log file will update within 2.5 seconds after an event that needs to be logged occurs (manifest being opened, operator signing in/out, etc). If each of 50 test log updates occur within 2.5 seconds after an event, the test is accepted

Results:

1) Passed
2) Passed, except for when a SIFT job is required. In that case, if there are a lot of containers, the time may take more than 15 minutes.
3) Unfortunately, we were not able to implement the power outage
4) We ended up having to manually save the log file, so this test is not applicable