

Part B:

Mason Wilkins

825647427

The screenshot shows a VS Code editor with a C++ file named `sortcar.cpp`. The code implements a sorting function `SortCar::sortCarInventory` that takes a vector of strings `carInventory` and a vector of strings `promotionOrder` as input. The function uses a selection sort algorithm. Handwritten annotations in black ink explain the time complexity of the algorithm.

Handwritten Annotations:

- $5 = 1 \Rightarrow 0$ (near line 32)
- $T(N) = O(N) + 1 + 1 + 1 + 1 + 1 = O(N) + 5 = N$ (near line 38)
- $T(N) = N + O(N) + 5 \Rightarrow O(2N) + 5$ (near line 47)
- $\Rightarrow O(N)$ (near line 58)

Code Snippets:

```
24 syllabus.  
25 Name: Mason Wilkins  
26 Red ID: 825647427  
27 */  
28  
29  
30 vector<string> SortCar::sortCarInventory(vector<string> carInventory, vector<string> promotionOrder) {  
31  
32     unsigned long low = 0; // low pointer to keep track of our beginning order index  
33     string temp; // temp variable to swap  
34     unsigned high = carInventory.size()-1; // high pointer to keep track of our end order index  
35  
36  
37     for(unsigned long i = 0; i < carInventory.size(); i++) { // first for loop in order to iterate through searching for promotionOrder(0)  
38         if(carInventory.at(i) == promotionOrder.at(0)) { // first condition to check if i == promotionOrder[0], if found the low pointer is at 0, :  
39             temp = carInventory.at(i); // sets temp variable to hold value at i  
40             carInventory.at(i) = carInventory.at(low); // then we set i to the value that needs to be pushed to the front which is our low pointer  
41             carInventory.at(low) = temp; // now that the value of low and i are the same, we need to set low to the original i value  
42             low++; // increment low so that everything to the left of low is sorted and we don't go behind the pointer  
43         }  
44     }  
45  
46     for(unsigned long j = carInventory.size()-1; j > 0; j--) { // second for loop is the same as the first, except we want to start from the back  
47         if(carInventory.at(j) == promotionOrder.at(2)) { // we have a new pointer of high index which is set to carInventory.size()-1, and decrease  
48             temp = carInventory.at(j); // sets temp variable to hold the value of j which we are switching  
49             carInventory.at(j) = carInventory.at(high); // since we are decrementing in this loop we are going backwards in the vector. We set car  
50             carInventory.at(high) = temp; // then since the value of j and the high pointer are the same, to avoid value duplication, we set the i  
51             high--;  
52         }  
53     }  
54  
55     // once the vector is sorted, return the vector  
56  
57     return carInventory;  
58  
59  
60  
61
```

Comments in the code:

- The whole for loop iterates for $O(n)$ because it is dependent on the size of `carInventory`.
- The if statement, since it is comparing 2 indices is a constant time operation. Thus, evaluating at $O(1)$.
- The next operations are all constant time. Setting variable values all equating to $O(1)$ for constant time.
- The for loop iterates for $O(n)$ times, because it is dependent on the size of `car`.
- An if loop depending on the operation has a constant time of $O(1)$. If we were comparing function calls, then the function could change the time complexity.
- The last operations are all $O(1)$ time complexity because they are simply setting values and performing in place swap operations.

Terminal Output:

```
Test Passed  
zsh swervo
```

```

23  should this not be the case; I will be subject to penalties as outlined in the course
24  syllabus.
25  Name: Mason Wilkins
26  Red ID: 825647427
27  */
28
29
30  vector<string> SortCar::sortCarInventory(vector<string> carInventory, vector<string> promotionOrder) {
31
32      unsigned long low = 0; // low pointer to keep track of our beginning order index
33      string temp; // temp variable to swap
34      unsigned high = carInventory.size()-1; // high pointer to keep track of our end order index
35
36
37      for(unsigned long i = 0; i < carInventory.size(); i++){ // first for loop in order to iterate through searching for promotionOrder(0)
38          if(carInventory.at(i) == promotionOrder.at(0)){ // first conditon to check if i == promotionOrder(0), if found the low pointer is at 0,
39              temp = carInventory.at(i); //sets temp variable to hold value at i
40              carInventory.at(i) = carInventory.at(low); //then we set i to the value that needs to be pushed to the front which is our low pointer
41              carInventory.at(low) = temp; // now that the value of low and i are the same, we need to set low to the original i value
42              low++; // increment low so that everything to the left of low is sorted and we dont go behind the pointer
43          }
44      }
45      //once the vector is sorted, return the vector
46
47      for(unsigned long j = carInventory.size()-1; j > 0; j--){ //second for loop is the same as the first, except we want to start from the back
48          if(carInventory.at(j) == promotionOrder.at(2)){ // we have a new pointer of high index which is set to carInventory.size()-1, and decrease
49              temp = carInventory.at(j); // sets temp variable to hold the value of j which we are switching
50              carInventory.at(j) = carInventory.at(high); //since we are decrementing in this loop we are going backwards in the vector. We set car
51              carInventory.at(high) = temp; // then since the value of j and the high pointer are the same, to avoid value duplication, we set the
52              high--;
53          }
54      }
55      //once the vector is sorted, return the vector
56
57      Memory(input):
58      (carInventory) vector of N integers
59      (promotionOrder) vector of N integers
60      Memory (Non-input) Variables:
61      low
62      high
63
64      return carInventory;
65  }
66
67

```

Time complexity explanation:

For my algorithm, the two loops used are running at $O(N)$ because they are dependent on the input size of the car inventory vector. The if loops inside each other for loops are constant time operations only making a comparison between indexes; if in the "if" statements I was comparing two function outputs, that could change the whole complexity of the program depending on those functions.

All in all: $N + N = 2N$

In each for loop, there are five constant time operations, which are the comparison and in-place swapping of the indices.

$$T(N) = 2N + 5 + 5;$$

Constants are emitted because they evaluate to zero.

$$T(N) = N$$

Therefore the time complexity for this algorithm is running at $O(N)$;

Space complexity:

For this algorithm, the space complexity evaluates at $O(N)$. To evaluate the space complexity, we first need to consider memory used through our inputs which would be the vectors car inventory and promotion order.

So far:

$$S(N) = N + N$$

Once the inputs are accounted for, we go to the variables used in the program holding values in the loops.

$$S(N) = 2N + 9$$

Therefore with this analysis, the space complexity of the program is $S(N) = 2N + 9 = O(N)$

Auxiliary Space:

To evaluate the auxiliary space of this program, we need to evaluate all memory not related to input which would be the first three variables declared at the beginning of the algorithm.

$$S(N) = 3$$