Turner Trowbridge
827959204
10/11/22

Assignment 2 - Time and Auxilary Space Complexity Analysis

Code	Time Complexity Analysis	Auxilary Space Complexity Analysis
void PromotedCarModelStack::push(string model, int price) { PromotedModel latestModel = PromotedModel(model, price);	• 1 operation	• 1 fixed-size variable **S(N) = O(1)**
latestPromotedModelStack.push_front(latestModel);	• 1 operation	
if (maxMinList.empty()){	• 2 operations	
maxMinList.push_front(make_pair(latestModel, latestModel));	• 2 operations (not worse case)	
}else if (price ≥ maxMinList.front().first.getPromotedPrice()) {	• 4 operations	
maxMinList.push_front(make_pair(latestModel, maxMinList.front().second));	• 4 operations (not worst case)	
} else if (price ≤ maxMinList.front().second.getPromotedPrice()) {	• 4 operations	
maxMinList.push_front(make_pair(maxMinList.front().first, latestModel));	• 4 operations (not worst case)	

} else {
maxMinList.push_front(make_pair(maxMinList.front().first, maxMinList.front().second));
}
	• 6 operations	

}	**f(N)** = 18 * O(1) **= O(1)**	

PromotedModel
PromotedCarModelStack::pop() {
  if (latestPromotedModelStack.empty()) {
  throw logic_error("Promoted car model stack is empty");
  }
 	
  PromotedModel latestModel = latestPromotedModelStack.front();
 	
  latestPromotedModelStack.pop_front();
  maxMinList.pop_front();
 	
  return latestModel;
}
	• 3 operations
	
	• 2 operations
	
	• 2 operations
	
	• 1 operation
	
	f(N) = 8 * O(1)
	= O(1)
		• 1 fixed-size variable
		S(N) = O(1)

PromotedModel
PromotedCarModelStack::peek() {
  if (latestPromotedModelStack.empty()) {
  throw logic_error("Promoted car model stack is empty");
  }
 	
  PromotedModel latestModel = latestPromotedModelStack.front();
 	
  return latestModel;
}
	• 3 operations
	
	• 2 operations
	
	• 1 operation
	
	f(N) = 6 * O(1)
	= O(1)
		• 1 fixed-size variable
		
		S(N) = O(1)

PromotedModel
PromotedCarModelStack::getHighestPricedPromotedModel() {
  if (latestPromotedModelStack.empty()) {
  throw logic_error("Promoted car model stack is empty");
  }
 	
  PromotedModel highestPricedModel = maxMinList.front().first;
 	
  return highestPricedModel;
}
	• 3 operations
	
	• 3 operations
	
	• 1 operation
	
	f(N) = 7 * O(1)
	= O(1)
		• 1 fixed-size variable
		
		S(N) = O(1)

PromotedModel
PromotedCarModelStack::getLowestPricedPromotedModel() {
 	
  if (latestPromotedModelStack.empty()) {
  throw logic_error("Promoted car model stack is empty");
  }
 	
  PromotedModel lowestPricedModel = maxMinList.front().second;
 	
  return lowestPricedModel;
}
	• 3 operations
	
	• 3 operations
	
	• 1 operation
	
	f(N) = 7 * O(1)
	= O(1)
		• 1 fixed-size variable
		
		S(N) = O(1)