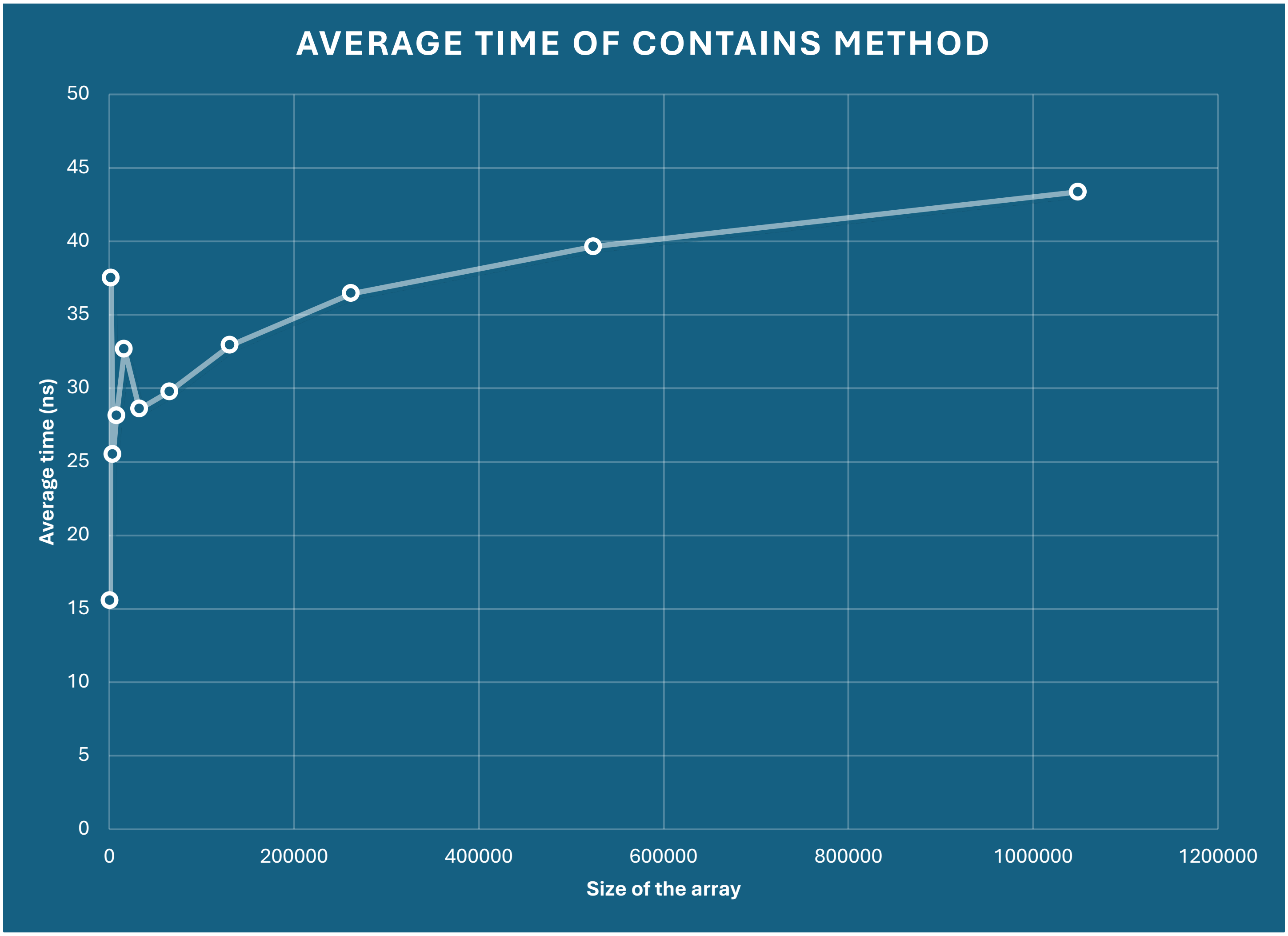
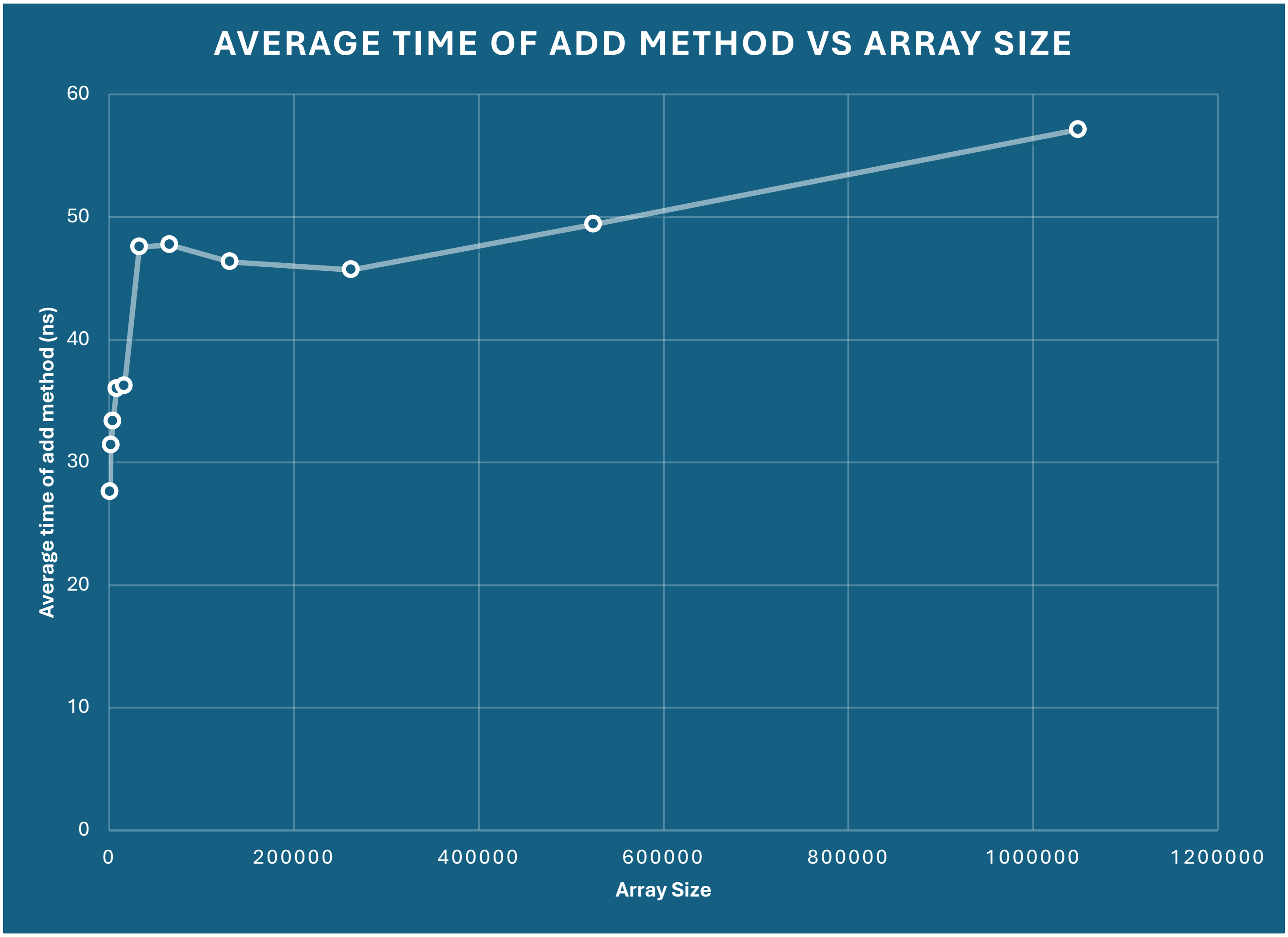
1. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development time.)
   a. Implementation
      i. No manual resizing is needed with a List. ArrayList automatically grows so add method is simplified. Also, there is no checking if array is full.
      ii. Instead of manually shifting elements like we do with System.arraycopy(...), the List handles positioning and shifting internally.
      iii. List has get and set methods to access elements for new binary search method. Method would still need to be manual if we are still not using Collections.binarySearch
      iv. List supports indexed access so Iterator would be a little easier to implement. List does have its own iterator() method but for our Problem, we would still need to customize it to work with sorted insertion.
   b. Efficiency
      i. Running time efficiency can be compared in the following:
         1. Insertion and resizing.
            a. ArrayList: automatic, possible memory overhead (unneeded capacity)
            b. Our array-backed implementation: manual, control how much to grow array
         2. Shifting Elements
            a. O(n) for both, both require shifting elements to the right
         3. Binary Search
            a. Our array-backed method allows us to directly manipulate the array
         4. Memory efficiency
            a. Depends how well we manage the resizing in our array-backed version
      ii. Program development time
         1. Ease of use
            a. List provides built in methods
         2. Error reduction
            a. Managing your own array can increase likelihood of errors
         3. Custom resizing
            a. Our version allows for more granular control over performance

2. What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?
   a. O(log n) because were using binary search
   b. Contains method checks if specific element is contained in the set(backed by a sorted array)
   c. The method is likely to be called a lot and doing a binary search on a sorted array is more efficient than unsorted.
3. Plot the running time of BinarySearchSet's contains method, using the timing techniques demonstrated in previous labs. Be sure to use a decent iteration count to get a reasonable average of running times. Include your plot in your analysis document. Does the growth rate of these running times match the Big-oh behavior you predicted in question 2?
   a. Yes the graph has a classic O(log N) shape



4. Consider your add method. For an element not already contained in the set, how long does it take to locate the correct position at which to insert the element? Create a plot of running times. Pay close attention to the problem size for which you are collecting running times. Beware that if you simply add N items, the size of the sorted set is always changing. A good strategy is to fill a sorted set with N items and time how long it takes to add one additional item. To do this repeatedly (i.e., iteration count), remove the item and add it again, being careful not to include the time required to call remove() in your total. In the worst-case, how much time does it take to locate the position to add an element (give your answer using Big-oh)?
    a. Locating the correct position to insert an element takes O(log N) time in worst case
        i. d/t need to maintain sorted property of set while keeping it balanced
    b. The addTiming class measures time taken to add a single element repeatedly.
        i. Each iteration adds the targetElement then removes it right after, so that the set size (N) remains constant throughout each iteration
    c. Time taken to remove the element is not included in measurement. The compensationIteration simulates inherit costs with a contains check


AVERAGE TIME OF ADD METHOD VS ARRAY SIZE
Average time of add method (ns)
60
50
40
30
20
10
0
0
200000
400000
600000
800000
1000000
1200000
Array Size