[**Closed book and notes.**] Show all of your work clearly in the space provided or on the additional page at the end of the exam. If the additional page is used, clearly identify to which exam question it is related. Be sure to **read each problem carefully**. Note that the exam is double sided.

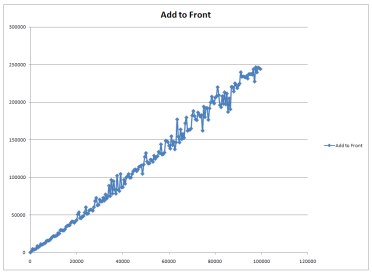
**1. (a)** (15 points) Implement the Exam<E> class that is used below. You may extend the ArrayList<E> class to assist in your implementation (strongly recommended). Your implementation must make the following statements work properly:

```
// Following line creates a collection of words: honk, if, you, ...
List<String> words = Arrays.asList("honk if you do not have a horn".split(" "));
Set<String> wordSet = new Exam<String>();
wordSet.add("honk");
wordSet.add("honk");            // not added because "honk" already in the set
wordSet.add("now");
wordSet.addAll(words);          // Adds the collection of words to the set
wordSet.contains("note");       // returns false
```

**(b)** (5 points) Using big-oh notation, describe the overall worst case time complexity for the contains method for your Exam class. Suppose that the size of the your Exam object is $N$. Be sure to explain your reasoning and state any additional assumptions that you make.

**(c)** (5 points) Using big-oh notation, describe the overall worst case time complexity for the addAll method for your Exam class. Suppose that the size of the your Exam object is $N$ and the size of the container passed to addAll is $M$. Be sure to explain your reasoning and state any additional assumptions that you make.

**2.** (10 points) In the second lab, many of you found that the test program produced a graph similar to the following for adding elements to the front of a LinkedList. However, adding one element to the front of a LinkedList of size $N$ should take $O(1)$ time. Explain what's going on here.




**3.** (10 points) Compare and contrast the internal structures of the ArrayList and LinkedList containers.

**4.** Consider the following method:

```
public static double getMedian(List<Double> nums)
{
  int N = nums.size();
  if(N<1)
  {
    throw new Exception("Grip a get");
  }
  double median = nums.get(0);
  int index = 0;
  for(int i=0; i<N/2; ++i)
  {
    for(int j=1; j<N-i; ++j)
    {
      if(nums.get(j)<median)
      {
        median = nums.get(j);
        index = j;
      }
    }
    swap(nums, j-1, index);
  }
  return median;
}
```

**(a)** (10 points) Suppose that the List passed to the method is an ArrayList and assume that swap swaps the $(N-i)^{th}$ and $index^{th}$ values in nums and runs in $O(N^2)$ time. Using big-oh notation, describe the overall worst case time complexity for the getMedian method. Be sure to explain your reasoning and state any additional assumptions that you make.

**(b)** (10 points) Suppose that the List passed to the method is a LinkedList and assume that swap swaps the $(N-i)^{th}$ and $index^{th}$ values in nums and runs in $O(N)$ time. Using big-oh notation, describe the overall worst case time complexity for the getMedian method. Be sure to explain your reasoning and state any additional assumptions that you make.

**(c)** (10 points) Suppose that the List passed to the method is the SinglyLinkedList developed in lecture and assume that swap swaps the $(N-i)^{th}$ and $index^{th}$ values in nums and runs in $O(1)$ time. Using big-oh notation, describe the overall worst case time complexity for the getMedian method. Be sure to explain your reasoning and state any additional assumptions that you make.

**5.** In lecture, we implemented a portion a CircularQueue<E> class (see page 9).
**(a)** (5 points) Implement the isEmpty() method.

**(b)** (5 points) Implement the size() method.

**(c)** (15 points) Implement the dequeue() method.

Additional work area for any problem. Clearly identify which problem is associated with the work on this page.

```
public class CircularQueue<E> implements PureQueue<E> {

  private static final int CAPACITY = 32;
  private E[] queue;
  private int front;
  private int back;
  private boolean isFull;

  public CircularQueue() {
    queue = new E[CAPACITY];
    front = 0;
    back = 0;
    isFull = false;
  }

  public boolean enqueue(E val) {
    boolean added = false;
    if(!isFull) {
      queue[back] = val;
      added = true;
      back = (back+1) % CAPACITY;
      if(back==front) {
        isFull = true;
      }
    }
    return added;
  }
}
```