**Lab 3 Document:**

This document assumes that the List of Dots being passed into each method is a LinkedList.

**Big-O Analysis:**

private static long removeDotsIndex(List<Dot> dots, int numberDesired){  
 final long startTime = System.*nanoTime*();  
 while (numberDesired < dots.size()){  
 double min = Double.*MAX\_VALUE*;  
 int minIndex = -1;  
 int currentSize = dots.size();  
  
 for (int i = 0; i < currentSize; i++){  
 double critValue;  
 if (i == 0){  
 dots.get(i).setCriticalValue(dots.get(currentSize - 1), dots.get(i + 1));  
 critValue = dots.get(i).getCriticalValue();  
 } else if (i == currentSize - 1){  
 dots.get(i).setCriticalValue(dots.get(i - 1), dots.get(0));  
 critValue = dots.get(i).getCriticalValue();  
 } else{  
 dots.get(i).setCriticalValue(dots.get(i - 1), dots.get(i + 1));  
 critValue = dots.get(i).getCriticalValue();  
 }  
  
 if (critValue < min){  
 min = critValue;  
 minIndex = i;  
 }  
 }  
 dots.remove(minIndex);  
 }  
 final long endTime = System.*nanoTime*();  
 return endTime - startTime;  
}

n = number of dots in the list.

T(n) = 1 + n(1 + 1 + 1 + n + 1 + n(1 + 1 + b1(n + n + n + n) + (1 – b1)b2(n + n + n + n) + (1 – b1)(1 – b2)b3(n + n + n + n) + b4(1 + 1) + 1) + 1 + 1) + 1 + 1 = 3 + n(6 + n + n(3 + b1(4n) + (1 – b1)b2(4n) + (1 – b1)(1 – b2)b3(4n) + b4(2)) = 3 + 6n + n2(3 + b1(4n) + (1 – b1)b2(4n) + (1 – b1)(1 – b2)b3(4n) + b4(2)) = 3 + 6n + 3n2 + n2(4n) + n2(b4(2)) = 3 + 6n + 3n2 + 4n3 + n2(b4(2))

3 + 6n + 3n2 + 4n3 + n2(b4(2)) ε O(n3)

* Because the list of dots is assumed to be a LinkedList and getting an element from a LinkedList has a run time of O(n) and the get() methods are within two for loops, the run time of the program is dependent on the number of dots n cubed.

private static long removeDotsIterator(Collection<Dot> dots, int numberDesired){  
 final long startTime = System.*nanoTime*();  
  
 while (numberDesired < dots.size()){  
 double min = Double.*MAX\_VALUE*;  
  
 // Attempting to calculate the critical value for the first dot.  
 Iterator<Dot> firstDotIterator = dots.iterator();  
 Dot firstDot = firstDotIterator.next();  
 Dot secondDot = firstDotIterator.next();  
 Dot lastDot = null;  
 for (Dot dot : dots) {  
 lastDot = dot;  
 }  
 firstDot.setCriticalValue(lastDot, secondDot);  
 double critValue = firstDot.getCriticalValue();  
 if (critValue < min){  
 min = critValue;  
 }  
  
  
 // Moving on to the rest of the dots  
 Iterator<Dot> previous = dots.iterator();  
 Iterator<Dot> dotIterator = dots.iterator();  
 dotIterator.next();  
 Iterator<Dot> next = dots.iterator();  
 next.next();  
 next.next();  
 Dot previousDot = null;  
 Dot currentDot = null;  
 Dot nextDot = null;  
  
 while (dotIterator.hasNext()){  
  
 previousDot = previous.next();  
 currentDot = dotIterator.next();  
 if (next.hasNext()){  
 nextDot = next.next();  
 currentDot.setCriticalValue(previousDot, nextDot);  
 critValue = currentDot.getCriticalValue();  
 }  
  
  
  
 if (critValue < min){  
 min = critValue;  
 }  
 }  
  
 // Calculating the critical value of the last dot  
 Iterator<Dot> lastIterator = dots.iterator();  
 lastDot.setCriticalValue(previousDot, firstDot);  
 critValue = lastDot.getCriticalValue();  
 if (critValue < min){  
 min = critValue;  
 }  
  
 Iterator<Dot> dotRemover = dots.iterator();  
 while (dotRemover.hasNext()){  
 Dot removedDot = dotRemover.next();  
 if (removedDot.getCriticalValue() == min){  
 dotRemover.remove();  
 }  
 }  
 }  
  
 final long endTime = System.*nanoTime*();  
 return endTime - startTime;  
}

n = number of dots in the collection

T(n) = 1 + n(1 + 1 + 1 + 1 + 1 + 1 + n(1 + 1) + 1 + 1 + 1 + b1(1) + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + n(1 + 1 + 1 + b2(1 + 1 + 1) + b3(1)) + 1 + 1 + 1 + 1 + b4(1) + 1 + n(1 + 1 + b5(1)) + 1) + 1 + 1 + 1 = 4 + n(24 + 2n + b1(1) + n(3 + b2(3) + b3(1)) + b4(1) + n(3 + b5(1))) = 4 + 24n + 2n2 + n\*b1(n) + n2(3 + b2(3) + b3(1)) + n\*b4(1) + n2(3 + b5(1)) = 4 + 24n + 2n2 + n\*b1(n) + 3n2 + n2\*b2(3) + n2\*b3(1) + n\*b4(1) +3n2 + n2\*b5(1) = 4 + 24n +8n2 + n2\*b1(n) + n2\*b2(3)+ n2\*b3(1) + n\*b4(1) + n2\*b5(1).

4 + 24n + 8n2 + n2\*b1(n) + n2\*b2(3) + n2\*b3(1) + n\*b4(1) + n2\*b5(1) ε O(n2)

* Because there are loops inside of loops for this method and those loops depend on the number of dots in the collection, the method seems to fit into O(n2).

**Graph of runtime:**

**Interpreting results:**

1. From benchmarking the data, it appears that the runtime increases more when using indices than it does when uses iterators.
2. From the Big-O Analysis, it appears that using indices takes more time than using iterators.
3. It appears that the Big-O Analysis and my actual results are consistent with each other.