Computer Science Master's Project Report

Rensselaer Polytechnic Institute

275 Windsor Street
Hartford CT, 06120-2991

A Comparison of Data Storage Techniques for Linear Linked Lists and List Arrays and Their Performance Utilizing the Google Android Operating System in a Handheld Configuration

Submitted by

James K. Falconer

S# 660383236

In partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

December 2009

(For Graduation April 2010)

December 16, 2009

Approved by:

Project Advisor

______________________ Date: ______________________

Eugene Eberbach, PhD.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Presented herein is a study of two data storage techniques. The problem is presented in the context of Google Android Operating System and developed using Java in an emulated Eclipse environment. This work performs a study of which data storage technique ADT (Abstract Data Type) List Array or Linked List implementation is more effective in delivering data from a handheld computer.

The nature of the Google Android handheld operating system is that it utilizes unique memory and file management techniques that may yield different results when compared to commonly held wisdom.

The first method, ADT List Array, is generally considered faster than Linked List when randomly accessing an element within a list. A Linked List has to traverse every node preceding the chosen element to reach the element.

The second method, a linear Linked List, is generally considered faster when inserting and deleting elements within a list. A linked list only has to modify the pointers between two elements to add or delete an element. An ADT List Array may have to modify (shift) half, if not more, elements for the insertion or deletion of a new element.

An application to study and compare both data storage techniques were then installed and run on a genuine Google Android handheld device, the Motorola Droid by Verizon. This application performs the 4 most common tasks performed by arrays (Add, Remove, Read, and Insert) and determines the nodes randomly, and in different sized arrays, to try to provide a fair study. From this data the project will try to derive recommendations and conclusions.

ACKNOWLEDGEMENTS

I would like to thank my project advisor, Dr. Eugene Eberbach, RPI Hartford, for assisting and guiding me on this project. I feel extremely grateful for the opportunity to spend a semester working on a project in a field of great personal interest under his direction.

I would also like to thank my fiancée Seanne Hanke, Yale MBA, who understood that I needed to work on this project, work full time, and take an Operating Systems course at the same time as purchasing a home and moving to Southie without getting too mad.

I would also like to thank my father James K Falconer Sr, whose masters thesis is the size of a dictionary, and due to unfortunate political turmoil had to lose much of what he earned working and studying thousands of hours in our motherland of Chile.

1. INTRODUCTION

1.1 Introduction

Handheld Operating systems, such as Google Android have very limited resources in both speed and storage due to its small size. It is quite common for applications to be quite sluggish and have very low data storage and access capabilities due to the limitations of the handheld device. To improve this circumstance software applications, that have extensive data access routines, would need to process data quickly and efficiently.

Google provides most standard Sun Java classes as the developmental language for creating applications. Java includes some very powerful classes that store and organize data. Two of the most popular methods are Linked List arrays and ADT List arrays. These two structures provide simple and powerful methods to access data. By wisely deciding on which data structure to utilize, a developer can provide a product that can be faster and more powerful by simply taking the right decision on what type of data structure to use from the beginning.

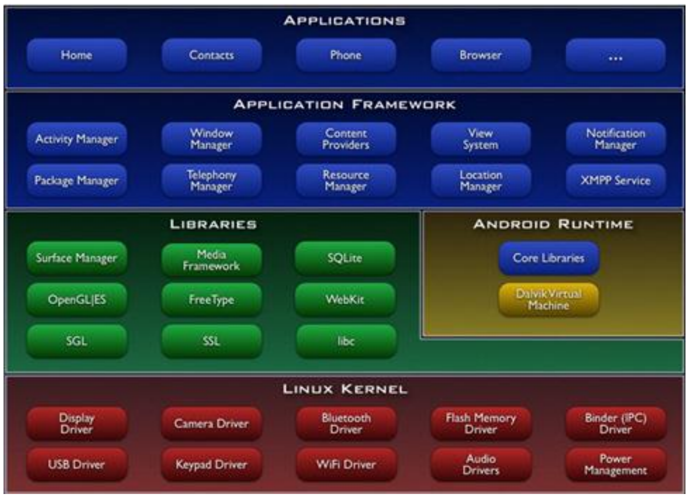
1.2 Background

Handheld computers provide rich graphical experiences and highly intuitive interfaces for users. Beginning from the first PDA, the Apple Newton in 1993, to today's interconnected mobile phones the handheld computer offers the power to process rich and complex applications. These handheld devices employ impressive GUIs which also allow users to access large amounts of information; From dynamic and quickly changing internet data sources to small static databases.

This study will use the Android Operating System which had its debut on November 5, 2007. [1] It is a mobile operating system originally developed by a small firm named Android, which was then purchased by Google, that offers developers the ability to write Java applications using standard and android centric Java libraries. The operating system was spun off from Google and now exists as an entity named the Open Handset Alliance (OHA).

Although only capturing 11% of the Mobile Operating System market, when compared to Apple IPhone’s 50% share, it has increased substantially from the 4% it had only one year ago [2]. The Android Operating System has several features that make it quite interesting in comparison to other popular mobile operating systems.

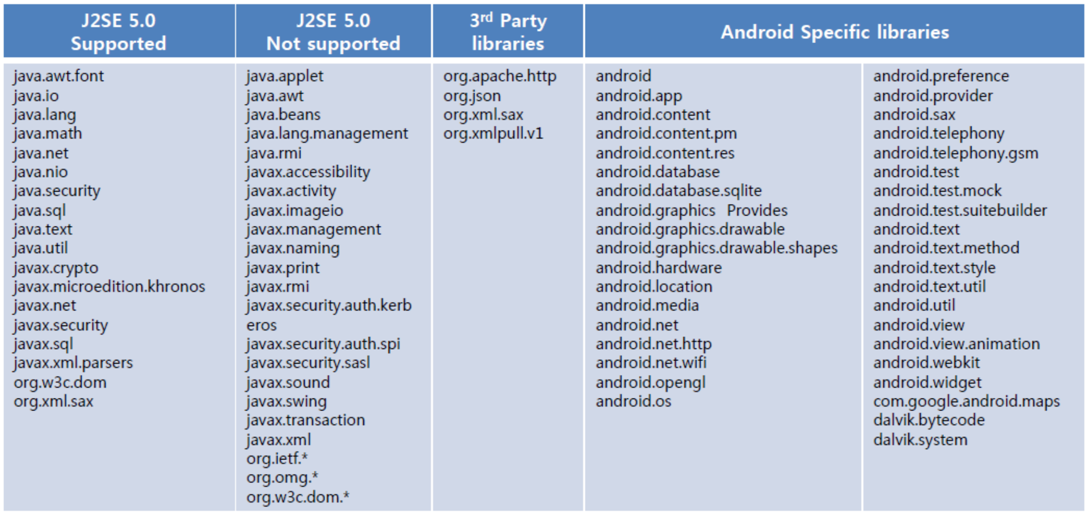
1. It uses an open source Apache license.
2. It uses Java as the development language
3. It has the backing of a very large corporation, Google, and many other software and hardware manufacturers under the name OHA (Open Handset Alliance)
4. The kernel utilized for the operating system is a trimmed down Linux 2.6. Android utilizes the Android developed Dalvik virtual machine during runtime to convert java code to compiled DEX files.



At left is shown the architecture of the Android Operating System. Android is built on top of the Linux 2.6 kernel. The Linux distribution that is part of the Android OS is not a full version, but a modified and partial distribution.[3]

1.3 Dalvik Virtual Machine

The Dalvik Virtual Machine is Android’s proprietary runtime component for running applications. Dalvik, named after a town in Iceland that is the ancestral home of an Android engineer[4], is an interpreter-based virtual machine. Dalvik runs the Java code that was compiled as DEX files (Dalvik executables.) Android supports most basic Java libraries and add several of its own. The chart below illustrates the Java libraries that are available, and also the libraries that are not available, to developers[5].



J2SE 5.0 Supported	J2SE 5.0 Not supported	3rd Party libraries	Android Specific libraries	
java.awt.font	java.applet	org.apache.http	android	android.preference
java.io	java.awt	org.json	android.app	android.provider
java.lang	java.beans	org.xml.sax	android.content	android.sax
java.math	java.lang.management	org.xmlpull.v1	android.content.pm	android.telephony
java.net	java.rmi		android.content.res	android.telephony.gsm
java.nio	javax.accessibility		android.database	android.test
java.security	javax.activity		android.database.sqlite	android.test.mock
java.sql	javax.imageio		android.graphics Provides	android.test.suitebuilder
java.text	javax.management		android.graphics.drawable	android.text
java.util	javax.naming		android.graphics.drawable.shapes	android.text.method
javax.crypto	javax.print		android.hardware	android.text.style
javax.microedition.khronos	javax.rmi		android.location	android.text.util
javax.net	javax.security.auth.kerberos		android.media	android.util
javax.security	javax.security.auth.spi		android.net	android.view
javax.sql	javax.security.sasl		android.net.http	android.view.animation
javax.xml.parsers	javax.sound		android.net.wifi	android.webkit
org.w3c.dom	javax.swing		android.opengl	android.widget
org.xml.sax	javax.transaction		android.os	com.google.android.maps
	javax.xml			dalvik.bytecode
	org.ietf.*			dalvik.system
	org.omg.*			
	org.w3c.dom.*			

Also unique to the virtual machine is that it is register-based rather than stack based due to the limitations of the hardware. In general register based machines require less instructions than stack based machines to load and process data. But register based machines must then encode the source and destination registers which increases the size of the instructions.

Motorola Droid for Verizon.
115.8 x 60 x 13.7mm and
weighs 165gm
iphone 3GS measures
115.5x62.1x12.3mm and weighs
135gm[6]

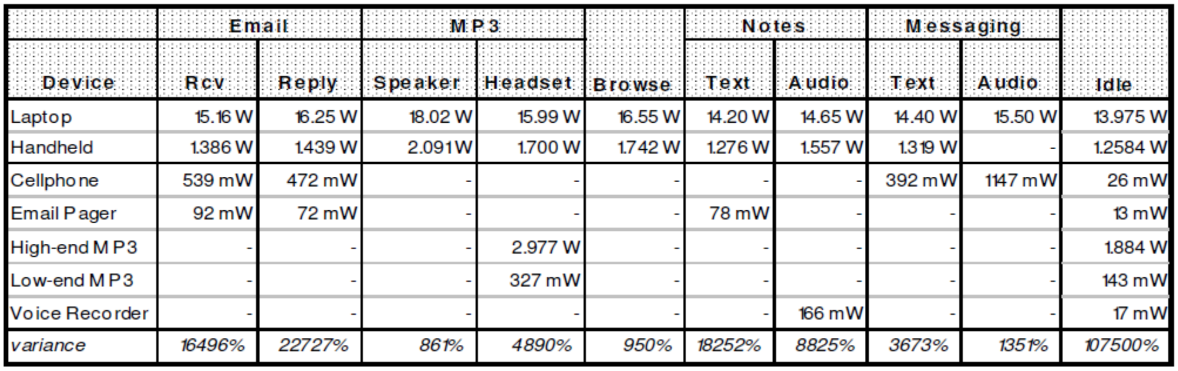
By utilizing an operating system that is quickly becoming one of the most popular platforms to use there is an opportunity to study and deliver solutions to a new and exciting platform in software development.

1.4 Hardware Limitations

Handheld computers running the Android Operating System utilize hardware that is physically much smaller than desktop or laptop computers. The Apple Newton, regarded as one of the largest handheld computers ever made measured 1 inch thick, much smaller than the newest desktops and comparable to the newest laptops. The newly released Verizon Droid, that utilizes the Google Android OS, measures 2.3 inches x 4.6 inches x 0.6 inches thick. This small size, in comparison with nearly anything ever released that has a user interface, creates some challenges in both hardware and software design.

Note Size of first handheld device, the Apple Newton, in comparison to 2 modern phones, the Apple iPhone and the Palm Treo[7].

The term “handheld device” is usually regarded as having pocket size proportions and is a limiting factor for providing computational power for three reasons. The first reason is that a small form factor limits the capability of a power source to effectively provide energy for high performance hardware for a long duration. Since batteries on handheld devices need to perform adequately for several hours of use, and also be small in stature, a compromise needed to be made. Handheld devices, in general, utilize 1/10th of the energy used by a laptop[8]. This dramatic difference in scale equates to lowering CPU performance to conserve energy.



	Email		MP3			Notes		Messaging		
Device	Rcv	Reply	Speaker	Headset	Browse	Text	Audio	Text	Audio	Idle
Laptop	15.16 W	16.25 W	18.02 W	15.99 W	16.55 W	14.20 W	14.65 W	14.40 W	15.50 W	13.975 W
Handheld	1.386 W	1.439 W	2.091 W	1.700 W	1.742 W	1.276 W	1.557 W	1.319 W	-	1.2584 W
Cellphone	539 mW	472 mW	-	-	-	-	-	392 mW	1147 mW	26 mW
Email Pager	92 mW	72 mW	-	-	-	78 mW	-	-	-	13 mW
High-end MP3	-	-	-	2.977 W	-	-	-	-	-	1.884 W
Low-end MP3	-	-	-	327 mW	-	-	-	-	-	143 mW
Voice Recorder	-	-	-	-	-	-	166 mW	-	-	17 mW
variance	*16496%*	*22727%*	*861%*	*4890%*	*950%*	*18252%*	*8825%*	*3673%*	*1351%*	*107500%*

Illustration above shows power consumption for various small devices. Note power consumption of handheld devices in comparison to other devices[9].

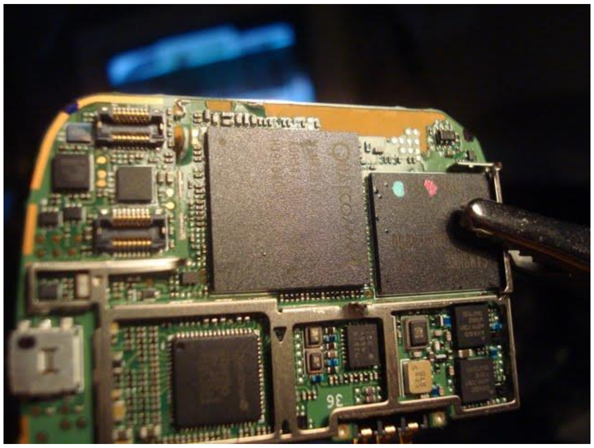
In addition to lower power consumption, handheld devices also are limited by the size of the hardware that will fit in the small form device. To fit all the components of a computer into a small form factor requires some adjustments when compared to a full size PC. As an example these are specifications of the Android capable Qualcomm MSM7201A[10] chipset in comparison with a desktop system featuring an Intel Core 2 Duo processor[11].

1. CPU Speed: 528 MHz (Compared to 2-4 GHz in x86 computers)

2. Power Consumption 500mW (Compared to 90 W in x86 computers)

3. 65 nm small form factor (Compared to 45 nm, Socket P in x86 computers)

The limited CPU capability, in comparison to larger x86 based computers, is due to multiple factors which are mostly derived from simply being smaller. A slower CPU, ½ of 1% of power usage, and utilizing a much smaller footprint in comparison to a Socket P architecture Intel Core 2 Duo.



Motherboard of the Google Android enabled HTC Dream phone[12].

In comparison to modern desktop machines, the Android based handheld computers also have much less RAM to utilize for processing. A basic comparison is shown as follows:

1. Level 1 Cache: 16 KB (Compared to 32-64 KB in newer x86 computers)
2. System RAM: 642-256 MB (Compared to 2-8 GB in new x86 computers)

After low level and high level processes have started, if the system contains 64 MB, the system RAM left over would be barely 20 MB. This is when memory management becomes very important for such resource bare machines.

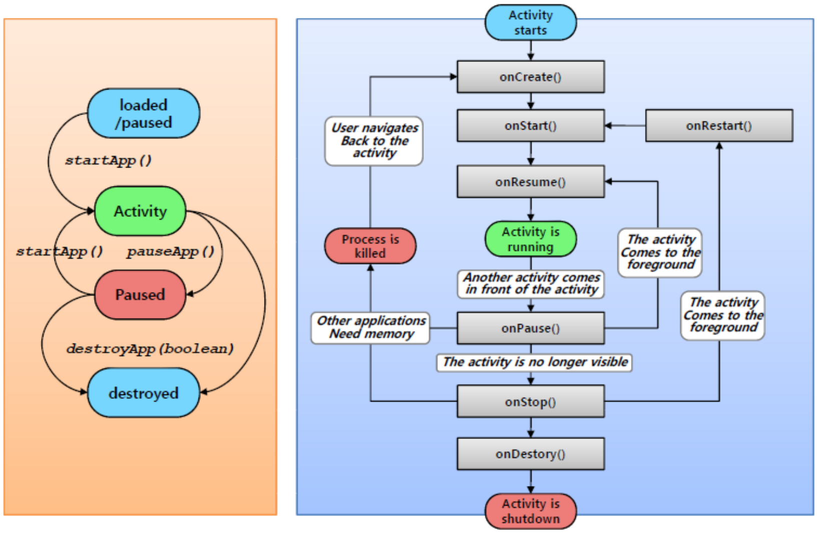
1.5 Memory Management

Memory management is actualized in two places in the Android OS. Memory allocation is performed by the Dalvik Virtual machine and memory management is handled by the Linux kernel[13].

Due to the relatively small amount of RAM available for mobile computers there are several techniques employed by Android to effectively extend RAM. Android equipped mobile computers have the capability of utilizing virtual memory to expand its capabilities. The Linux kernel inside Android uses a ‘page out’ Least Recently Used (LRU)[15] method to swap information from RAM to a secondary storage, such as flash memory.

Android applications are also limited to a very small heap size, 16MB. If an application exceeds this resource the OS will send out a crash message. Application developers must be careful to manage RAM resources to not exceed limitations.

Application swapping is also unique in the Android OS. If the new application needs more memory the currently running activity will be saved, then closed. This assumes that a user prioritizes fast running applications but will accept longer application swapping and startup times. A diagram of the application swap is shown below[15]:



1.6 File Management

Handheld Operating systems, such as Google Android have very limited resources to process data. Due to the relatively small amount of RAM and flash memory based storage available for handheld computers there are several techniques employed by Android to effectively extend its capabilities.

Android equipped mobile computers have the capability of utilizing virtual memory to expand its memory and file management capabilities. Similar to Linux, Android uses a ‘page out’ method to swap information from RAM to a secondary storage, such as flash memory.

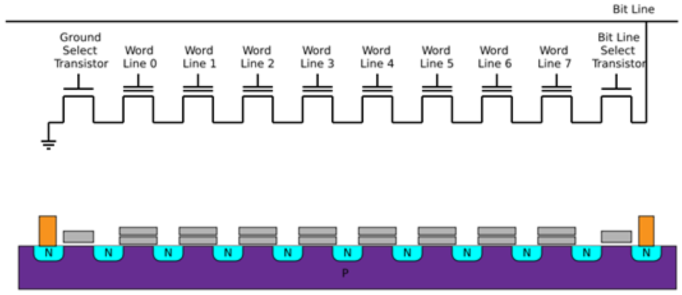
Google Android utilizes the YAFFS file system to manage its flash memory storage. YAFFS (Yet Another Flash File System) is an optimized file management system that is written especially for NAND flash based systems. YAFFS in not OS specific, and it is an open source project that is available to other operating systems as well[16]. For most handheld devices hard drives are much too large and consume too much power to be efficient. In contrast flash memory is able to provide similar capabilities with much less power consumption. NAND memory offers extremely high storage density and has fast writes.

Flash based file management can be quite slow and problematic due to several reasons.

1. Flash memory cannot be rewritten a byte at a time. The file system must first erase the memory, one block at a time, and then write, one block at a time. NAND memory can be randomly access pages, but it cannot process random access rewrite or erase[17]. To overcome this problem, memory segments are marked “dirty” when they are no longer needed. When the entire block is dirty, then YAFFS allows the block to be erased.

2. Flash memory has memory wear[18]. There is a hardware lifetime, numbering in the hundreds of thousands of rewrites, that eventually will stop working. If this area of memory is being utilized for virtual memory, a high amount of rewriting is entirely possible.

The following illustration is an image of a NAND memory structure and wiring on a piece of silicon. The block structure displays the area that would need to be erased during a write operation[19]:



Modern NAND flash utilize large pages. YAFFS takes advantage of large page sizes; 2048 bytes + 64 bytes for spare areas. A page would be 64 blocks of 2048 bytes. Strict write requirements still exist for the newer NAND chips. Each page within each block must be written sequentially, and can only be written once.

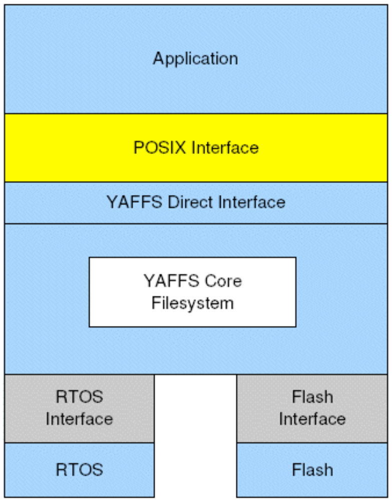


Image of the YAFFS embedded structure[21].

YAFFS does not require formatting. A blank flash chip is a formatted flash chip. On written blocks, it marks the 5th byte in the spare 64 byte area for bad blocks. When writing, YAFFS writes a whole page at a time. This page includes the file metadata, which includes timestamps, path, and name. Each new file is assigned a unique object ID number. This ID number will exist on every page within the spare area. The following is a list of a typical YAFFS page structure:

0-511	Data
512-515	Tags
516	Data status
517	Block Status
518-519	Tags
520-522	ECC on 2nd 256 bytes
523-524	Tags
525-527	ECC on 1st 256 bytes

YAFFS Page construction[21].

YAFFS features the following capabilities:

1. Journaling: YAFFS features a log system which helps maintain data stability during power failures[22].

2. Garbage collection: YAFFS uses a highly optimized garbage collection strategy. Collection is performed when free space starts becoming low. When space becomes scarce a block with a few dirty pages and some good pages is chosen. YAFFS will then move the good pages to another block and the block will be marked as dirty and then erased[23].

3. Portability: YAFFS was originally developed for Linux, but its modular design made it easy and portable to port to many other operating systems, such as WinCE and eCos.

4. POSIX Support: YAFFS supports POSIX style file systems through file system interface calls.

YAFFS is quite similar to the Linux Ext3 file system, but it maintains it differences due to differences in storage media. The differences can be summarized as follows:

1. File Access: Flash memory devices have no seek latency, unlike hard drives, and can randomly access files.

2. Block Erasure: Flash devices have much more work to perform during erasures when compared to hard drive based systems.

3. Wear Leveling: YAFFS employs several wear leveling techniques to minimize wear on the flash chip. Hard drive based systems are much more durable in this regard.

YAFFS is a very powerful utility that provides enhancements when compared to traditional hard-disk based systems. In contrast to these enhancements, YAFFS also has to work around several problems inherent in flash based memory systems. The combination of enhancements and workarounds create an system environment that may yield interesting data and study.

1.7 Purpose

By studying Java implementations for data storage techniques this project can provide programmers scientific data and recommendations to provide more effective data retrieval schemes. It is the purpose of this study to guide a prospective programmer when they need to choose a particular method to effectively gather data using the new technology of Google Android. While similar studies have been made using numerous other operating systems and architectures it is the intent of this study the effectiveness of the new operating system. To summarize, this study should provide insightful conclusions due to the following factors:

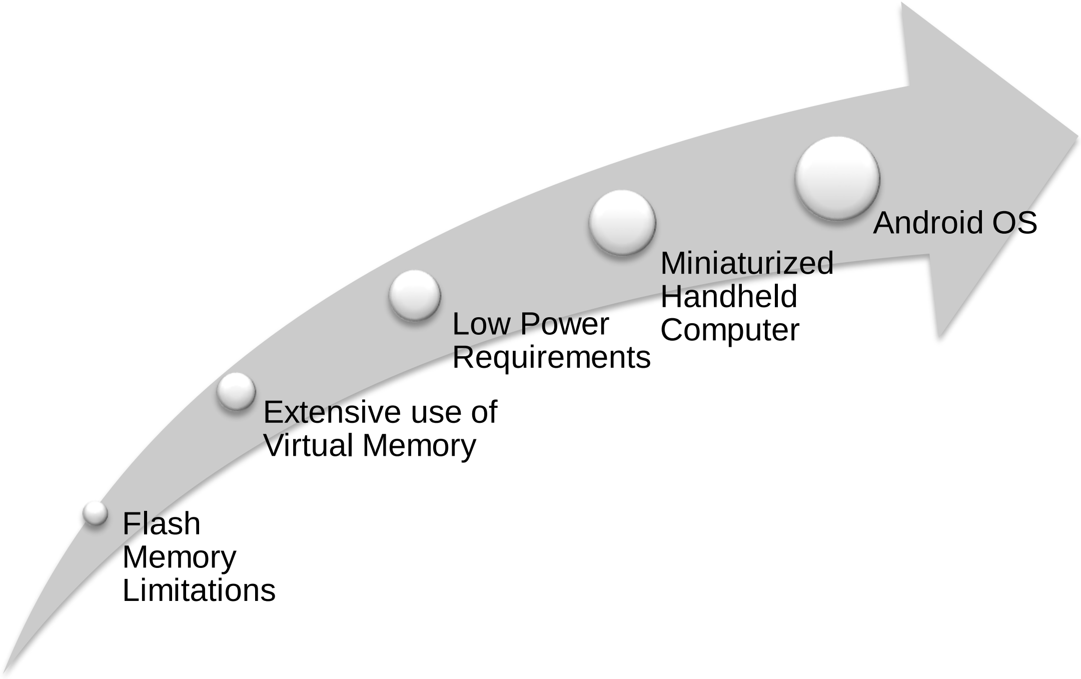
1. This project shall be working with a new and innovative operating system that has not been utilized or tested as extensively as other well known operating systems

2. This project shall be working with a small form factor ‘miniaturized’ computer that has limited capabilities in comparison to typical desktop or laptop architectures.

3. This project shall be working with an operating system that limits its power use to conserve energy

4. This project shall be working with an operating system that extensively uses virtual memory that may impact speed and throughput of some processes

5. This project shall be testing an operating system that uses flash memory that has distinctive write characteristics.


Flash Memory Limitations
Extensive use of Virtual Memory
Low Power Requirements
Miniaturized Handheld Computer
Android OS

The demand, and assumption, that applications need to be fast and powerful, means that programmers must use the most efficient and intelligent choices when programming their device. As powerful as desktops from 10 years ago, but accessing data and providing features from present day means that programmers have a difficult challenge; how do they provide powerful features while maintaining the constraints imposed by the small form factor and the capabilities of a new and relatively unknown operating system.

1.8 Objectives

This research proposal will try to give us a better understanding of the question; Which data storage technique ADT (Abstract Data Type) List Array or Linked List implementation, is more effective in delivering data from Google Android handheld device? This paper will discuss methods to test the effectiveness of the two data storage techniques. The methods will be tested using object oriented JAVA code.

The two data storage techniques, ADT List Array and Linked List implementations, will undergo time measurement based on the five following methods: add, read, remove, insert and an overall combination of the four stated methods.

The test data will be a random array of 5 different sizes (500, 1000, 2500, 4000, and 7500) which will be accessed by the application. It is the intention of this study to discover which data storage method is most effective for a data vault of these sizes on a handheld device.

1.9 Hypothesis

This project will be testing five null hypotheses to determine which data storage technique ADT (Abstract Data Type) List Array or Linked List implementation is more effective in delivering data from a handheld device using the Android operating system.

Null Hypotheses:

1. There is no statistically significant difference between the speed at which ADT List Array or Linked List implementation when adding random data in a dataset using object oriented Java code.
2. There is no statistically significant difference between the speed at which ADT List Array or Linked List implementation when randomly retrieving data in a dataset using object oriented Java code.
3. There is no statistically significant difference between the speed at which ADT List Array or Linked List implementation when randomly removing data in a dataset using object oriented Java code.
4. There is no statistically significant difference between the speed at which ADT List Array or Linked List implementation when inserting data in a dataset using object oriented Java code.
5. There is no statistically significant difference between the speed at which ADT List Array or Linked List implementation when performing all four of the most common functions in a dataset using object oriented Java code.

Alternate Hypotheses:

1. There is a statistically significant difference when using ADT List Array than when using Linked List implementation when adding random data in a dataset using object oriented Java code.
2. There is a statistically significant difference when using ADT List Array than when using Linked List implementation when randomly retrieving data in a dataset in a small dataset using object oriented Java code.
3. There is a statistically significant difference when using ADT List Array than when using Linked List implementation when randomly removing data in a dataset in a small dataset using object oriented Java code.
4. There is a statistically significant difference when using ADT List Array than when using Linked List implementation when inserting data in a dataset in a small dataset using object oriented Java code.
5. There is a statistically significant difference when using ADT List Array than when using Linked List implementation when performing all four of the most common functions in a dataset in a small dataset using object oriented Java code.

2. DATA STRUCTURES

2.1 Definition

A data structure is defined as follows by the US National Institute of Standards and Technology: *“An organization of information, usually in memory, for better algorithm efficiency, such as queue, stack, linked list, heap, dictionary, and tree, or conceptual unity, such as the name and address of a person. It may include redundant information, such as length of the list or number of nodes in a subtree.”*[24]

Within the computer science field, a data structure is considered to be a method of organizing and storing data. There are several types of data structures, two of which are discussed in this paper. Different data structures are suited for different tasks and implementations. And some data structures excel in different tasks than others.

Data structures are utilized in most software applications because they enable us to process data with great efficiency and speed. Depending on which type of data structure this project uses there can be a significant difference in efficiency when choosing a particular data structure.

2.2 Linked Lists

As defined by the US National Institute of standards a linked list is; *“A list, a collection of items accessible one after another beginning at the head and ending at the tail, implemented by each item having a link (a reference, pointer, or access handle to another part of the data structure) to the next item.”*[25] From this definition the array can be visualized as a chained link, with pointers marking from where each link comes from and where it goes. For a node to be reached, the link must be traversed to the pointer that is connected to the node.

2.3 ADT List Array

As defined by the US National Institute of standards a linked list is; *“An assemblage of items that are randomly accessible by integers, the index.”*[26]

This type of list is can be randomly accessible, in contrast to a linked list. For a node to be reached, we simply need to know where the node is stored, then jump to the position of the node.

2.4 Comparisons

In general the linked list data structure is considered more efficient where nodes will be added or removed at the beginning or middle of a list. In an array the same operations would require moving many of its nodes up or down in the array to accomodate new nodes or to fill in nodes that were deleted. Arrays would run at $\Theta(n)$ when performing moves in the beginning or middle of an array. For linked list the run time for the same action would be $\Theta(1)$, which is much faster if n (the number of nodes in a list) is larger than 1.

The advantage of the array list data structure is that it is generally faster when randomly accessing items in a list. When reading a k-th item on a list the run time is $\Theta(1)$. For linked list the run time is $\Theta(n)$ because it must start at the beginning of the array and traverse the list to the k-th node. If n is larger than 1, then array list should be much faster.[27]

3. APPLICATION

The application this project used to test the hypothesis was written in Java, with Android API extensions, within the Eclipse 3.5 (Galileo) framework. It was decided that this project will use the latest API version available, 2.0, codenamed Éclair which was released on October 26, 2009. This version is compatible to run on the Verizon Droid which also runs version 2.0 of the Google operating system.

It was decided that the project will write one application that would run through five randomly populated array sizes. The project will use array sizes of 500, 1000, 2500, 4000, and 7500 for several reasons.

1. Multiple Array sizes will give us the opportunity to graphically and statistically view trends and increases.
2. The array of 7500 was chosen to be the maximum size before encountering difficulties through testing on the device. Array sizes of 10,000 and above were tried, but for unforeseen reasons failed to produce data.
3. Random numbers were chosen to populate the fields so the project can provide generalized results not skewed by specialized data.

The project shall also offer data output for both List Array and Linked List implementations with only one run to correlate runs with the same random arrays.

3.1 Initialization

Importation of standard Java libraries to implement the two data structures utilizes the standard java.util libraries. The application also import libraries for creating the Android activity, since the application is sending the output to the screen. The import of scroll view, linear layout, and text view are also simply to implement a long vertical message sent to a screen that can scroll.

The project is to perform 25 tests for 5 different sizes of arrays. It is important to see if there is correlation between array sizes and the performance of the application. The project will use list sizes of 500, 1000, 2500, 4000, and 7500 units. The project will populate each list randomly using the Random class which returns a new psudo-random int value which is uniformly distributed between 0 and the high value of each array size.

```
package list.test.com;

import android.app.Activity;     //Import tools to begin Activity thread
import android.os.Bundle;
import android.widget.TextView;  //Programmatic output to screen

import java.util.ArrayList;      //Java library for implementing ADT list arrays
import java.util.LinkedList;     //Java library for implementing linked lists
import java.util.List;           //Java library for implementing an array
import java.util.Random;         //Java library for implementing random numbers

public final class ListTestAndroid extends Activity {

        private static final int TESTS = 5;   //# of tests to be performed this run
        private static final int[] LIST_SIZES = new int[] {500, 1000, 2000, 4000, 
7500}; //Size of arrays
        private static final Random RANDOM = new Random(); //Initialize random number 
generator
```

3.2 Populating the Array

To populate the array the application will sequentially add random integers to the array using the add method. The application timed the performance of populating the whole array for both list array and linked list implementations.

```
private static long performAdds(final List<Integer> list, final int adds) {

    long startAdds = System.currentTimeMillis();//Initialize timer in milliseconds
                 for (int i = 0; i < adds; i++) {
                   list.add(RANDOM.nextInt(Integer.MAX_VALUE));    //Add random #s
                 }
                 long endAdds = System.currentTimeMillis();        //Initialize end
                 long addTime = endAdds - startAdds;               //Perform math
                 return addTime;                                   //Returns value
               }
```

3.3 Reading the Array Randomly

To read the array the application performed random reads of the array using the get method. The application timed the performance of reading the whole array for both list array and linked list implementations.

```
private static long performRandomGets(final List<Integer> list) {
      long startGets = System.currentTimeMillis();   //Initialize timer
```

```
       int size = list.size();                 //Initialize size to size of array
              for (int i = 0; i < size; i++) {
                    int x = list.get(RANDOM.nextInt(size));//Get random nodes
                  }
                  long endGets = System.currentTimeMillis();        //Initialize
                  long time = endGets - startGets;                  //Perform math
                  return time;                 //Returns value of time operation
                }
```

3.4 Random Deletion of Nodes in the Array

To delete all of the elements of the array the application performed random deletions of array elements using the remove method. The application timed the performance of removing the whole array for both list array and linked list implementations.

```
private static long performRemove(final List<Integer> list, final int rems) {
    long startAdds = System.currentTimeMillis();//Initialize timer in milliseconds
       for (int i = 0; i < rems; i++) {
              list.remove(RANDOM.nextInt(rems));//Remove random nodes from whole array
                         }
       long endAdds = System.currentTimeMillis();     //Initialize end of timer
       long addTime = endAdds - startAdds;            //Perform math
       return addTime;                                //Returns value of time operation
                       }
```

3.5 Insertion into the Array

To recreate all of the elements of the array the application performed an insertion of a new array into the first node of the array until all the elements were populated using the add method. The applicaation timed the performance of recreating the whole array for both list array and linked list implementations.

```
private static long timeList(final List<Integer> list) {
      long startInsert = System.currentTimeMillis();  //Initialize timer
      int size = list.size();       //Initialize size to size of array
      for (int i = 0; i < size; i++) {
            list.add(0,(Integer.MAX_VALUE));//Insert value to populate array
                  }
      long endInsert = System.currentTimeMillis();    //Initialize
      long insertTime = endInsert - startInsert;      //Perform math
      return insertTime;      //Returns value of time operation
              }
```

3.6 Creation of the Arrays

Instead of a main method the application initializes an activity in the Android OS. The application then initializes the view structure and initializes the arrays for each array size

```
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        long[] arrayListAddTimes = new long[TESTS];
        long[] linkedListAddTimes = new long[TESTS];

        long[] arrayListRandomGetTimes = new long[TESTS];
        long[] linkedListRandomGetTimes = new long[TESTS];


        long[] arrayListRemove = new long[TESTS];
        long[] linkedListRemove = new long[TESTS];

        long[] arrayListInsert = new long[TESTS];
        long[] linkedListInsert = new long[TESTS];


        for (int listSize : LIST_SIZES) {
          for (int i = 0; i < TESTS; i++) {
              List<Integer> arrayList = new ArrayList<Integer>();
      //Instantiate ADT List Array
              List<Integer> linkedList = new LinkedList<Integer>();
      //Instantiate linked List
              arrayListAddTimes[i] = performAdds(arrayList, listSize);
              linkedListAddTimes[i] = performAdds(linkedList, listSize);

              arrayListRandomGetTimes[i] = performRandomGets(arrayList);
              linkedListRandomGetTimes[i] = performRandomGets(linkedList);

              arrayListRemove[i] = performRemove(arrayList, listSize);
              linkedListRemove[i] = performRemove(linkedList, listSize);

              arrayListInsert[i] = timeList(arrayList);
              linkedListInsert[i] = timeList(linkedList);
          }
        }
    }
```

3.7 Screen Output

```
TextView tv01 = new TextView(this);
tv01.setText(String.format("\n+ Size of list : %s+\n", listSize));
          ll.addView(tv01);
```

```
TextView tv02 = new TextView(this);
tv02.setText(String.format("Array list add: %s ms",
mean(arrayListAddTimes)));
          ll.addView(tv02);

TextView tv03 = new TextView(this);
tv03.setText(String.format("Linked list add: %s ms\n",
mean(linkedListAddTimes)));
          ll.addView(tv03);

TextView tv06 = new TextView(this);
tv06.setText(String.format("Array list random read: %s
ms",mean(arrayListRandomGetTimes)));
          ll.addView(tv06);

TextView tv07 = new TextView(this);
tv07.setText(String.format("Linked list random gets mean time: %s
ms\n",mean(linkedListRandomGetTimes)));
          ll.addView(tv07);

TextView tv08 = new TextView(this);
tv08.setText(String.format("Array list remove: %s
ms",mean(arrayListRemove)));
          ll.addView(tv08);

TextView tv09 = new TextView(this);
tv09.setText(String.format("Linked list remove: %s
ms\n",mean(linkedListRemove)));
          ll.addView(tv09);

TextView tv10 = new TextView(this);
tv10.setText(String.format("time to insert for ArrayList: %s ms",
mean(arrayListInsert))); // do timing for LinkedList
          ll.addView(tv10);

TextView tv11 = new TextView(this);
tv11.setText(String.format("time to insert for LinkedList: %s ms\n",
mean(linkedListInsert)));
          ll.addView(tv11);


this.setContentView(sv);

        }
    }
```

3.8 Timing Output

```
        private static long mean(final long[] times) {

            long sum = 0;
            for (long x : times) {
              sum = sum + x;
            }
            int n = times.length;
            return sum / n; }}
```

4. PROCEDURES

4.1 ADT List Array and Linked List Operational Procedure

The sequential list below summarizes the steps that the operations will take. The project should end up with 1000 time determinations. (40 data results x 25 times)

1. Open application to be tested by tapping on application icon on main screen. Application will automatically run.

2. Read results for each array size and confirm that it is the one that was requested.

3. Read time result for each array size and write into the column corresponding to the particular data structure and method being used.

4. Close Application by clicking on return home button and wait 5 seconds.

5. Repeat steps 1-4 25 times

The project will obtain tests for 1000 time tests derived from 25 separate randomly created datasets tested in the applications. This is a reasonably high number which should satisfy the need to adequately attain a good range of values. These values should also give us a recognizably significant visual indicator when graphing results.

The inspection will give us the following information:

1. Time it took to add values in an ADT List Array implementation.
2. Time it took to remove values using an ADT List Array implementation.
3. Time it took to read values in an ADT List Array implementation.
4. Time it took to insert values using an ADT List Array implementation.
5. Time it took to add values in a Linked List implementation.
6. Time it took to remove values using a Linked List implementation.
7. Time it took to read values using a Linked List implementation.
8. Time it took to insert values using Quick Sort in a Linked List implementation.

4.2 Quantitative Study

By averaging the time data per algorithm the project can find the following:

- Average time it takes to find data using ADT List Array implementation data storage technique.
- Average time it takes to find data using Linked List implementation data storage technique.

The project will also have the information to compute the following:

- The standard deviation of adding, removing, reading and inserting by using both data management techniques.
- The mean value of adding, removing, reading and inserting by using both data management techniques.

T-Test:

The project will also use the simplified T-Test to prove or disprove the null hypothesis. For the T test the project will pursue a significance level of 0.05. The 5% level is a traditional value which is going to be useful in accepting or rejecting the null hypothesis.

$\overline{X}_1$= Mean of the samples for ADT List Array

$\overline{X}_2$ = Mean of the samples for Linked List Implementation

SS_1 = Sum of the square for the samples of ADT List Array

SS_2 = Sum of the square for the samples of Linked List Implementation

N_1 = Number of samples of ADT List Array (25)

N_2 = Number of samples of Linked List Implementation (25)

$$t = \frac{(\overline{X}_1 - \overline{X}_2)}{\sqrt{\left[\frac{SS_1 + SS_2}{N_1 + N_2 - 2}\right]\left[\frac{1}{N_1} + \frac{1}{N_2}\right]}}$$ [28]

For the T test the project will pursue a significance level of 0.05. The 5% level is a traditional number which is going to be useful in rejecting the null hypothesis.

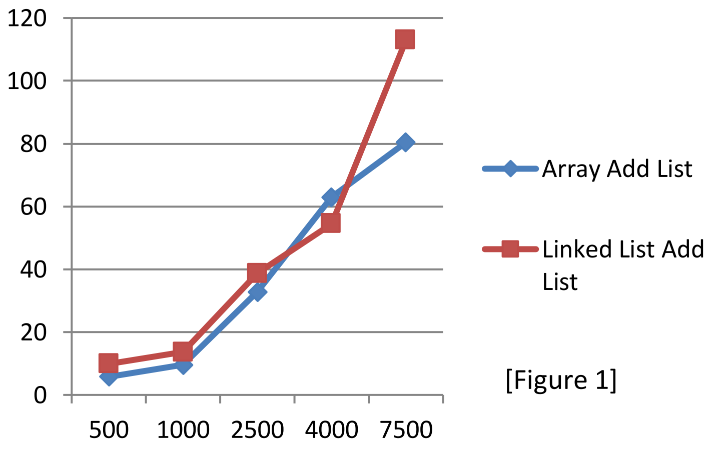
With a degree of freedom of 48 ((N1+N2-2) = 25+25-2), and a significance level of 0.05, we look at the value on the Table of Critical Values[29] and derive a number of 2.01 (for value 45). The analysis shall compare this value (2.01) with the results for each method.

5. EXPERIMENT AND ANALYSIS

5.1 Results for Adding Data to List

[Table 1] t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	0.455034496	0.544966
Variance	0.001883491	0.001883
Observations	25	25
Pooled Variance	0.001883491	
Hypothesized Mean Difference	0	
df	48	
t Stat	7.326265839	
P(T<=t) one-tail	1.16419E-09	
t Critical one-tail	1.677224197	
P(T<=t) two-tail	2.32838E-09	
t Critical two-tail	2.010634722	



[Figure 1]

Calculated t: 7.326

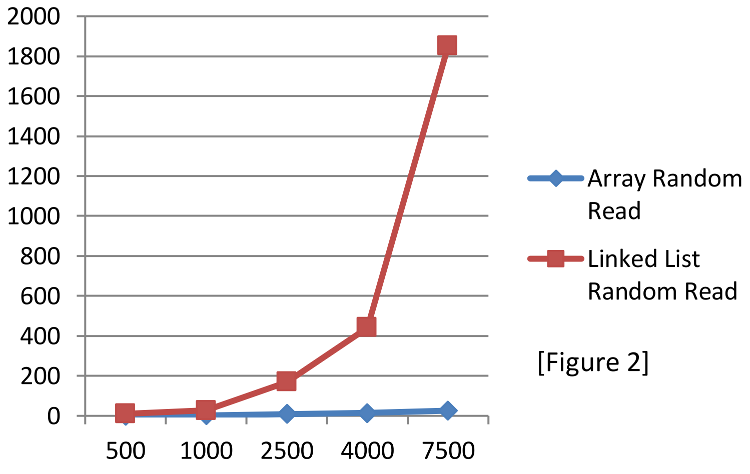
Significance level: 2.011

Calculated t > Significance level (7.326 > 2.011). From this calculation the Null Hypothesis #1 is rejected. In conclusion this study accepts that there is a statistically significant difference when using ADT List Array than when a Linked List implementation when you are adding data to a list in object oriented java code. As shown in table 1, the T calculation rejects the null hypothesis and shows that ADT List array is more effective in adding data to the array. Figure 1 demonstrates visually the quantitative conclusion that ADT List Array is more effective than Linked List in adding data to an array.

5.2 Results for Randomly Reading Data from List

[Table 2] t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	0.020450712	0.979549
Variance	3.86176E-06	3.86E-06
Observations	25	25
Pooled Variance	3.86176E-06	
Hypothesized Mean Difference	0	
df	48	
t Stat	1725.542529	
P(T<=t) one-tail	5.4353E-117	
t Critical one-tail	1.677224197	
P(T<=t) two-tail	1.0871E-116	
t Critical two-tail	2.010634722	



[Figure 2]

Calculated t: 1725.543

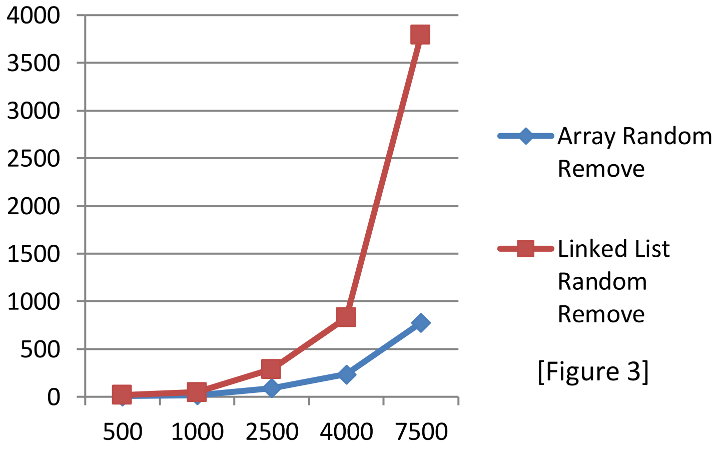
Significance level: 2.011

Calculated t > Significance level (1725.543> 2.011). From this calculation the Null Hypothesis #2 is rejected. In conclusion this study accepts that there is a statistically significant difference when using ADT List Array than when using a Linked List implementation when you are randomly reading data from a list in object oriented java code. As shown in table 2, the T calculation rejects the null hypothesis and shows that ADT List Array is more effective in randomly reading data from an array. Figure 2 demonstrates visually the quantitative conclusion that ADT List Array is more effective than Linked Lists in randomly reading data from an array.

5.3 Results for Randomly Removing Data from List

[Table 3] t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	0.185000854	0.814999
Variance	4.61862E-05	4.62E-05
Observations	25	25
Pooled Variance	4.61862E-05	
Hypothesized Mean Difference	0	
df	48	
t Stat	327.7468145	
P(T<=t) one-tail	2.27723E-82	
t Critical one-tail	1.677224197	
P(T<=t) two-tail	4.55446E-82	
t Critical two-tail	2.010634722	



[Figure 3]

Calculated t: 327.747

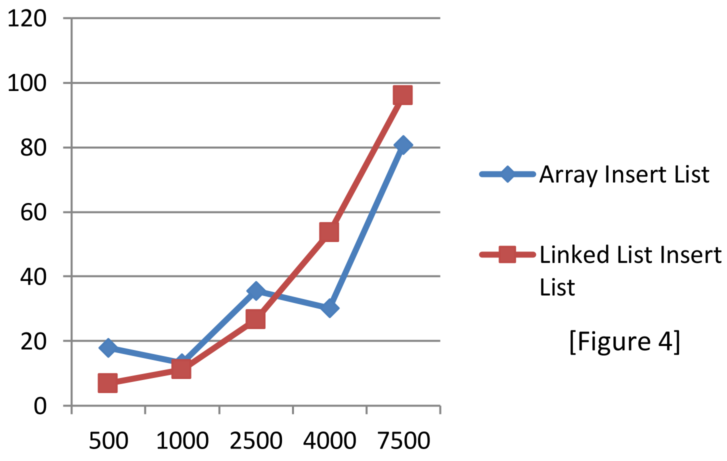
Significance level: 2.011

Calculated t > Significance level (327.747 > 2.011). From this calculation the Null Hypothesis #3 is rejected. In conclusion this study accepts that there is a statistically significant difference when using ADT List Array than when using a Linked List implementation when you are randomly removing data from a list in object oriented java code. As shown in table 3, the T calculation rejects the null hypothesis and shows that ADT List Array is more effective in randomly removing data from an array. Figure 3 demonstrates visually the quantitative conclusion that ADT List Array is more effective than Linked Lists in randomly removing data from an array.

5.4 Results for Inserting Data to List

[Table 4] t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	0.476344069	0.523656
Variance	0.00279319	0.002793
Observations	25	25
Pooled Variance	0.00279319	
Hypothesized Mean Difference	0	
df	48	
t Stat	3.165007991	
P(T<=t) one-tail	0.00134611	
t Critical one-tail	1.677224197	
P(T<=t) two-tail	0.00269222	
t Critical two-tail	2.010634722	



[Figure 4]

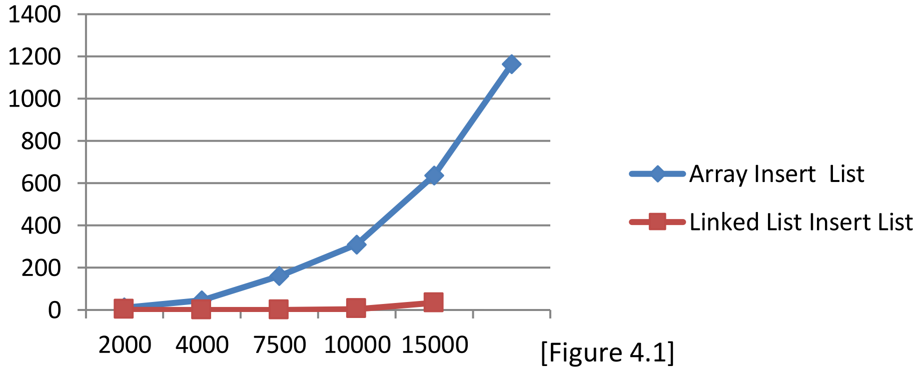
Calculated t: 3.165

Significance level: 2.011

Calculated t > Significance level (3.165 > 2.011). From this calculation the Null Hypothesis #4 is rejected. In conclusion this study accepts that there is a statistically significant difference when using ADT List Array than when using a Linked List implementation when you are inserting data into a list in object oriented java code. As shown in table 4, the T calculation rejects the null hypothesis and shows that ADT List Array is more effective in inserting data into an array. Figure 4 demonstrates visually the quantitative conclusion that ADT List Array is more effective than Linked List in inserting data into an array.

Special Note:

Running the same code on a 2 GHz PC running Windows XP resulted in very different preliminary results. Linked list inserts were much faster than ADT list array as shown below in the chart, graph, and t-test. Different array sizes were used and only 3 tests were performed.



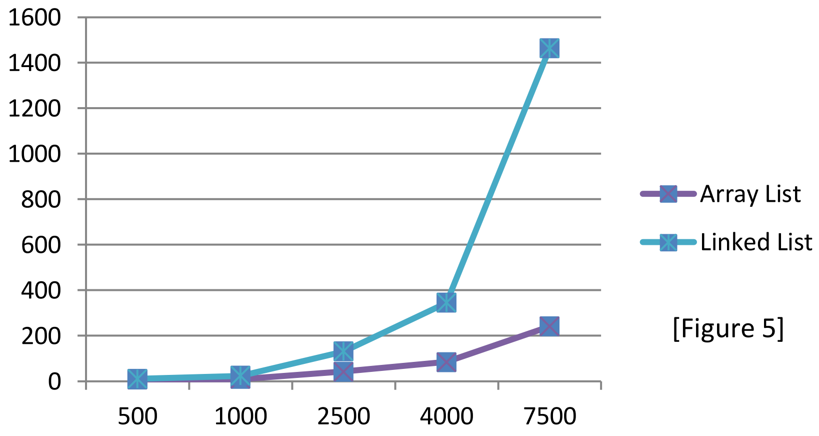
[Figure 4.1]

[Table 4.1] t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	0.966703	0.033297
Variance	5.9E-06	5.9E-06
Observations	3	3
Pooled Variance	5.9E-06	
Hypothesized Mean Difference	0	
Df	4	
t Stat	470.7299	
P(T<=t) one-tail	6.11E-11	
t Critical one-tail	2.131847	
P(T<=t) two-tail	1.22E-10	
t Critical two-tail	2.776445	

These results may be due to the differences in how Android accesses memory and virtual memory. There may be disadvantages to using a linked list because it has to do as much work as an array during times that a linked list should prove superior. There may be slowdowns when writing to blocks when using either linked lists or arrays. This project hypothesizes that the flash based virtual memory is encumbered when writing to the flash drive. The flash drive may prove to be so slow that any improvements exhibited by linked lists is simply lost when the memory logjam impedes processing. Eventually linked lists and ADT list arrays meet similar fates when processing a linked list favored process. These roadblocks seem to slow down processing for linked lists, thus giving much poorer results when running the same process on an Android device.

5.5 Overall Results and Values



[Figure 5]

[Table 5] t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	0.284207533	0.715792
Variance	0.00032332	0.000323
Observations	25	25
Pooled Variance	0.00032332	
Hypothesized Mean Difference	0	
df	48	
t Stat	-84.8603998	
P(T<=t) one-tail	2.89811E-54	
t Critical one-tail	1.677224197	
P(T<=t) two-tail	5.79621E-54	
t Critical two-tail	2.010634722	

Calculated t: 84.860
Significance level: 2.011
Calculated t > Significance level (84.860 > 2.011). From this calculation the Null Hypothesis #5 is rejected. In conclusion this study accepts that there is a statistically significant difference when using ADT List Array than when using a Linked List implementation when you are using overall methods in a list in object oriented java code. As shown in table 5, the T calculation rejects the null hypothesis and shows that ADT List Array is more effective in overall data processes in an array. Figure 5 demonstrates visually the quantitative conclusion that ADT List Array is more effective than Linked List in overall data processing in an array.

5.6 Average Values per Method

Average Values		Array Size					
		500	1000	2500	4000	7500	Test #
Array List	Add List	5.76	9.6	32.76	62.92	80.4	25 Tests
	Random Read	1.56	2.76	8.6	13.24	25.92	
	Random Remove	8.32	17.36	92.24	234.88	776.88	
	Insert List	17.96	13.24	35.56	30.24	80.84	
Linked List	Add List	9.88	13.64	38.72	54.52	112.92	
	Random Read	10.12	27.36	169.6	440.92	1850.76	
	Random Remove	17.76	46.36	287.88	831.64	3792.16	
	Insert List	6.96	11.24	26.64	53.6	96	

[Table 6]

5.7 Average Values per Data Structure

				Size		
Combined	500	1000	2500	4000	7500	Test #
Array List	8.4	10.74	42.29	85.32	241.01	25 Tests
Linked List	11.18	24.65	130.71	345.17	1462.96	

[Table 7]

Table 6 illustrates the average values of each method for all 25 tests for Array List and Linked Lists data structures. Table 7 illustrates the average values of all four combined methods for each data structure. From these results it is easy to compare values to note that ADT Array List is faster in each method that was tested.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

Testing the data provided insight which would not have been derived if there was not stringent statistical testing. Since the data was appropriate for using a T test the project was able to satisfactorily test each one of the five hypotheses.

Through testing it was discovered that ADT Lists Arrays are more effective than Linked Lists for all of our hypotheses; Adding, Inserting, Reading, Removing, and Overall. Linked Lists were not found to be effective in performing any four of the methods, but did compare well in inserting data.

In conclusion this project would recommend that a software development team that is using methods that add, remove, read or insert into an array choose ADT List Array as their data structure within the Google Android environment of a handheld device.

6.2 Future Work

Several areas for future work are suggested by this study.

1. A comparison of the same tests with a hard drive based x86 Intel desktop workstation on a full implementation of an Operating system such as Windows XP, Windows Vista, or Mac OS X. A direct comparison to contrast the differences and furthermore derive reasons if the results are significantly different.
2. A test of larger values for the array to push the boundaries and capabilities of the handheld device. Although our application encountered difficulty running larger than 7500 nodes, it is possible to modify the test, such as simply running a large array instead of multiple arrays. It is also preferable to wait some time for the hardware maker to release patches for the handheld device.
3. Perform further trend analysis with exponential equations to study the difference between data structures beyond the array sizes the study chose.
4. Perform analysis for other types of data structures.

REFERENCES

1. Android (Operating System), Online, 1 Page, Available at http://en.wikipedia.org/wiki/Android_%28mobile_device_platform%29, Accessed December 1, 2009

2. Erick Schonfeld, November 23, 2009, Apple And Android Now Make Up 75 Percent Of U.S. Smartphone Web Traffic, Online, 1 Page, Available at http://www.techcrunch.com/2009/11/23/apple-and-android-now-make-up-75-percent-of-u-s-mobile-web-traffic/, Accessed December 10, 2009

3. Android Architecture, Online, 1 Page, Available at http://developer.android.com/guide/basics/what-is-android.html, Accessed December 10, 2009

4. Brian DeLacey , November 12, 2007, Google Calling: Inside Android, the gPhone SDK, Online, 1 Page, Available at http://onlamp.com/pub/a/onlamp/2007/11/12/google-calling-inside-the-gphone-sdk.html, Accessed December 10, 2009

5. Yang Jeong Soo, February 28, 2009, Android Run-time (Dalvik VM & Core Lib.), Online, 23 Pages, Available at http://www.kandroid.org/board/kandroid_home.php, Accessed November 20, 2009

6. Michelle Megna, November 18, 2009, Droid sales surge, Online, 1 Page, Available at http://www.pdastreet.com, Accessed November 20, 2009

7. Peter N. Glaskowsky, July 30, 2008, An iPhone 3G - better late than never, CNET NEWS, Online, 1 Page, Available at news.cnet.com/8301-13512_3-10002513-23.html, Accessed December 13, 2009

8. Robert N. Mayo and Parthasarathy Ranganathan, August 12, 2003, Why Future Systems Need Requirements-Aware Energy Scale-Down, Energy Consumption in Mobile Devices, Internet Systems and Storage Laboratory, Hewlett Packard Labs, Page 5

9. Ibid.

10. Pdadb.net, April 13, 2008, Qualcomm MSM7201A RISC Chipset, Online, 1 Page, Available at http://pdadb.net/index.php?m=cpu&id=a7201a&c=qualcomm_msm7201a, Accessed November 20, 2009

11. Intel Core 2, Wikipedia.org, December 4, 2009,Online, 1 Page, Available at http://en.wikipedia.org/wiki/Intel_Core_2, Accessed December 10, 2009

12. David Renrut, Blog, June 8, 2009, HTC Dream – JTAG Tutorial to fix “bricked” devices, Available at http://r3nrut.com/?p=1, Accesed November 10, 2009

13. What is Android?, Online, 1 Page, Available at http://developer.android.com/guide/basics/what-is-android.html, Accessed November 10, 2009

14. Application Fundamentals, Online, 1 Page, Available at http://developer.android.com/guide/topics/fundamentals.html, Accessed November 10, 2009

15. Ibid.

16. Charles Manning, September 20, 2002, Linuxdevices.org, YAFFS: the NAND-specific flash file system - Introductory Article, Online, 1 Page, Available at http://www.yaffs.net/yaffs-nand-specific-

17. Flash Memory, Wikipedia.org, Online, 1 Page, Available at http://en.wikipedia.org/wiki/Flash_memory, Accessed November 8, 2009

18. Ibid.

19. Cyferz, Nand flash structure.svg, Wikipedia.org, Online, 1 Page, Available at http://en.wikipedia.org/wiki/File:Nand_flash_structure.svg, Accessed November 10, 2009

20. YAFFS Overview,Online, 1 Page, Available at http://www.yaffs.net/yaffs-overview, Accessed Novemeber 8, 2009

21. YAFFS Spec, Online, 1 Page, Available at http://www.yaffs.net/yaffs-spec, Accessed November 8, 2009

22. Ibid.

23. Ibid.

24. Paul E. Black, December 15, 2004, Data Structure, in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology, Accessed November 10, 2009

25. Paul E. Black, December 17, 2004, Link, in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. Accessed November 10, 2009

26. Paul E. Black, November 13, 2008, Array, in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. Accessed November 10, 2009

27. David J. Eck, December 2006, Section 10.2: Lists and Sets, Introduction to Programming Using Java, Fifth Edition

28. Houman Younessi, 2009, Lecture 8, CSCI 6961 Research Methods in Computer Science, Page 8

29. Table of Critical Values. Planet Math Website. 1 Page, Chi Woo. http://planetmath.org/encyclopedia/TableOfCriticalValuesOfTDistributions.html3, Accessed April 5, 2009

APPENDIX A

```
package list.test.com;

import android.app.Activity;      //Import tools to begin Activity thread
import android.os.Bundle;
import android.widget.TextView;   //Programmatic output to screen

import java.util.ArrayList;       //Java library for implementing ADT list arrays
import java.util.LinkedList;      //Java library for implementing linked lists
import java.util.List;            //Java library for implementing an array
import java.util.Random;          //Java library for implementing random numbers

public final class ListTestAndroid extends Activity {

         private static final int TESTS = 5;    //# of tests to be performed this run
         private static final int[] LIST_SIZES = new int[] {500, 1000, 2000, 4000,
7500}; //Size of arrays
         private static final Random RANDOM = new Random(); //Initialize random number
generator



private static long performAdds(final List<Integer> list, final int adds) {


    long startAdds = System.currentTimeMillis();//Initialize timer in milliseconds
                  for (int i = 0; i < adds; i++) {
                    list.add(RANDOM.nextInt(Integer.MAX_VALUE));    //Add random #s
                  }
                  long endAdds = System.currentTimeMillis();        //Initialize end
                  long addTime = endAdds - startAdds;               //Perform math
                  return addTime;                                   //Returns value
                }



private static long performRandomGets(final List<Integer> list) {
       long startGets = System.currentTimeMillis();   //Initialize timer
       int size = list.size();                  //Initialize size to size of array
              for (int i = 0; i < size; i++) {
                    int x = list.get(RANDOM.nextInt(size));//Get random nodes
                  }
                  long endGets = System.currentTimeMillis();        //Initialize
                  long time = endGets - startGets;                  //Perform math
                  return time;                  //Returns value of time operation
                }



private static long performRemove(final List<Integer> list, final int rems) {
    long startAdds = System.currentTimeMillis();//Initialize timer in milliseconds
       for (int i = 0; i < rems; i++) {
              list.remove(RANDOM.nextInt(rems));//Remove random nodes from whole array
                         }
       long endAdds = System.currentTimeMillis();     //Initialize end of timer
       long addTime = endAdds - startAdds;            //Perform math
       return addTime;                                //Returns value of time operation
                       }
```

```
private static long timeList(final List<Integer> list) {
      long startInsert = System.currentTimeMillis();  //Initialize timer
      int size = list.size();       //Initialize size to size of array
      for (int i = 0; i < size; i++) {
            list.add(0,(Integer.MAX_VALUE));//Insert value to populate array
                  }
      long endInsert = System.currentTimeMillis();    //Initialize
      long insertTime = endInsert - startInsert;      //Perform math
      return insertTime;      //Returns value of time operation
              }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ScrollView sv = new ScrollView(this);
        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);
        sv.addView(ll);

        long[] arrayListAddTimes = new long[TESTS];
        long[] linkedListAddTimes = new long[TESTS];

        long[] arrayListRandomGetTimes = new long[TESTS];
        long[] linkedListRandomGetTimes = new long[TESTS];


        long[] arrayListRemove = new long[TESTS];
        long[] linkedListRemove = new long[TESTS];

        long[] arrayListInsert = new long[TESTS];
        long[] linkedListInsert = new long[TESTS];


        for (int listSize : LIST_SIZES) {
          for (int i = 0; i < TESTS; i++) {
              List<Integer> arrayList = new ArrayList<Integer>();
      //Instantiate ADT List Array
              List<Integer> linkedList = new LinkedList<Integer>();
      //Instantiate linked List
              arrayListAddTimes[i] = performAdds(arrayList, listSize);
              linkedListAddTimes[i] = performAdds(linkedList, listSize);

              arrayListRandomGetTimes[i] = performRandomGets(arrayList);
              linkedListRandomGetTimes[i] = performRandomGets(linkedList);

              arrayListRemove[i] = performRemove(arrayList, listSize);
              linkedListRemove[i] = performRemove(linkedList, listSize);

              arrayListInsert[i] = timeList(arrayList);
              linkedListInsert[i] = timeList(linkedList);
          }
        }
    }
```

```
TextView tv01 = new TextView(this);
tv01.setText(String.format("\n+ Size of list : %s+\n", listSize));
          ll.addView(tv01);

TextView tv02 = new TextView(this);
tv02.setText(String.format("Array list add: %s ms",
mean(arrayListAddTimes)));
          ll.addView(tv02);

TextView tv03 = new TextView(this);
tv03.setText(String.format("Linked list add: %s ms\n",
mean(linkedListAddTimes)));
          ll.addView(tv03);

TextView tv04 = new TextView(this);
tv04.setText(String.format("Array list random read: %s
ms",mean(arrayListRandomGetTimes)));
          ll.addView(tv06);

TextView tv05 = new TextView(this);
tv05.setText(String.format("Linked list random gets mean time: %s
ms\n",mean(linkedListRandomGetTimes)));
          ll.addView(tv07);

TextView tv06 = new TextView(this);
tv06.setText(String.format("Array list remove: %s
ms",mean(arrayListRemove)));
          ll.addView(tv08);

TextView tv07 = new TextView(this);
tv07.setText(String.format("Linked list remove: %s
ms\n",mean(linkedListRemove)));
          ll.addView(tv09);

TextView tv08 = new TextView(this);
Tv08.setText(String.format("time to insert for ArrayList: %s ms",
mean(arrayListInsert))); // do timing for LinkedList
          ll.addView(tv10);

TextView tv09 = new TextView(this);
Tv09.setText(String.format("time to insert for LinkedList: %s ms\n",
mean(linkedListInsert)));
          ll.addView(tv11);


this.setContentView(sv);

        }
    }


    private static long mean(final long[] times) {

      long sum = 0;
      for (long x : times) {
```

```
    sum = sum + x;
  }
  int n = times.length;
  return sum / n;
}
```

APPENDIX B

Test Results

		Array Size					
		500	1000	2500	4000	7500	Test #
Array List	Add List	5	8	31	35	78	1
	Random Read	1	3	8	12	27	
	Random Remove	7	18	95	222	767	
	Insert List	3	16	26	48	74	
Linked List	Add List	8	26	39	77	127	
	Random Read	9	30	172	469	1795	
	Random Remove	16	51	303	866	3756	
	Insert List	5	9	23	38	99	
Array List	Add List	5	9	71	56	77	2
	Random Read	1	3	8	13	25	
	Random Remove	6	17	92	223	758	
	Insert List	7	19	30	26	89	
Linked List	Add List	7	11	51	45	109	
	Random Read	7	26	172	418	2033	
	Random Remove	28	45	292	883	3847	
	Insert List	6	9	34	65	96	
Array List	Add List	4	8	31	77	90	3
	Random Read	2	3	7	13	26	
	Random Remove	11	17	90	231	757	
	Insert List	4	6	27	26	60	
Linked List	Add List	29	21	39	46	136	
	Random Read	10	27	166	448	1853	
	Random Remove	20	49	298	827	3877	
	Insert List	7	9	23	48	95	
Array List	Add List	5	22	42	46	65	4
	Random Read	1	3	7	15	25	
	Random Remove	8	18	95	224	757	
	Insert List	3	6	103	37	88	
Lin	Add List	12	11	39	46	111	

	Random Read	12	28	171	420	1982	
	Random Remove	18	44	288	879	3773	
	Insert List	14	9	28	72	100	
Array List	Add List	5	9	32	13	79	5
	Random Read	1	3	7	13	25	
	Random Remove	9	17	90	220	767	
	Insert List	31	16	39	26	89	
Linked List	Add List	12	11	28	66	110	
	Random Read	8	27	162	437	1839	
	Random Remove	19	45	285	841	3782	
	Insert List	5	9	23	49	96	
Array List	Add List	6	8	32	98	82	6
	Random Read	1	2	7	13	28	
	Random Remove	9	17	97	223	786	
	Insert List	27	17	38	26	90	
Linked List	Add List	5	11	30	66	109	
	Random Read	9	27	163	471	1905	
	Random Remove	19	45	288	757	3984	
	Insert List	7	9	23	53	97	
Array List	Add List	4	8	32	66	80	7
	Random Read	1	2	8	13	26	
	Random Remove	7	17	99	223	752	
	Insert List	26	16	37	26	87	
Linked List	Add List	13	11	28	58	111	
	Random Read	13	27	214	471	2117	
	Random Remove	20	45	286	833	3762	
	Insert List	6	9	24	48	96	
Array List	Add List	7	9	32	68	78	8
	Random Read	1	2	7	13	25	
	Random Remove	9	17	90	333	765	
	Insert List	30	16	37	27	88	
Linked	Add List	6	11	29	57	121	
	Random Read	8	27	163	456	1887	

	Random Remove	15	49	280	845	3778	
	Insert List	6	9	24	52	95	
Array List	Add List	6	8	32	66	76	9
	Random Read	3	3	8	13	26	
	Random Remove	10	17	91	225	807	
	Insert List	27	16	37	26	88	
Linked List	Add List	7	11	28	57	117	
	Random Read	9	27	165	439	1953	
	Random Remove	18	49	288	843	3957	
	Insert List	5	9	24	49	87	
Array List	Add List	6	8	32	66	78	10
	Random Read	1	3	7	13	25	
	Random Remove	7	17	91	222	762	
	Insert List	26	17	37	27	89	
Linked List	Add List	7	13	28	58	109	
	Random Read	12	28	164	451	1890	
	Random Remove	18	45	291	800	3696	
	Insert List	6	12	24	49	96	
Array List	Add List	5	8	34	67	76	11
	Random Read	1	3	7	13	25	
	Random Remove	8	17	91	222	756	
	Insert List	28	16	38	26	91	
Linked List	Add List	7	11	28	57	109	
	Random Read	12	28	167	438	1839	
	Random Remove	18	46	283	819	3696	
	Insert List	5	9	23	49	95	
Array List	Add List	6	9	32	67	79	12
	Random Read	1	3	29	13	26	
	Random Remove	11	17	90	224	826	
	Insert List	26	16	40	26	90	
Linked List	Add List	8	11	28	61	109	
	Random Read	12	27	204	428	1812	
	Random Remove	14	44	282	810	3959	

	Insert List	6	9	26	50	98	
Array List	Add List	6	8	32	65	83	13
	Random Read	5	3	8	13	25	
	Random Remove	6	17	91	224	883	
	Insert List	28	16	37	28	89	
Linked List	Add List	15	11	28	57	118	
	Random Read	9	27	166	433	1821	
	Random Remove	21	47	286	808	3884	
	Insert List	5	9	24	49	96	
Array List	Add List	7	10	34	38	80	14
	Random Read	1	3	8	13	29	
	Random Remove	7	23	93	223	779	
	Insert List	4	16	26	48	75	
Linked List	Add List	8	25	39	77	123	
	Random Read	9	26	163	445	1806	
	Random Remove	13	50	285	836	3787	
	Insert List	7	9	24	38	99	
Array List	Add List	16	8	21	35	90	15
	Random Read	1	3	8	13	26	
	Random Remove	6	17	98	396	766	
	Insert List	3	6	28	50	49	
Linked List	Add List	9	14	39	68	37	
	Random Read	7	27	161	442	1804	
	Random Remove	11	49	302	842	3828	
	Insert List	4	21	60	67	104	
Array List	Add List	4	8	22	77	76	16
	Random Read	1	3	7	15	26	
	Random Remove	6	18	92	224	758	
	Insert List	7	6	27	26	87	
Linked List	Add List	26	11	41	46	115	
	Random Read	12	27	169	445	1830	
	Random Remove	22	44	290	811	3728	
	Insert List	8	19	34	51	96	
Arr	Add List	7	8	32	78	102	17

	Random Read	2	3	8	13	26	
	Random Remove	11	17	90	227	767	
	Insert List	26	16	37	26	63	
Linked List	Add List	7	11	29	45	122	
	Random Read	11	28	162	443	1773	
	Random Remove	13	47	278	808	3685	
	Insert List	8	9	23	51	84	
Array List	Add List	4	8	22	46	77	18
	Random Read	2	3	8	14	26	
	Random Remove	10	17	91	221	731	
	Insert List	8	6	27	26	88	
Linked List	Add List	7	22	38	45	109	
	Random Read	14	29	165	435	1898	
	Random Remove	21	45	280	806	3773	
	Insert List	9	20	34	73	101	
Array List	Add List	6	8	32	105	75	19
	Random Read	1	2	8	15	25	
	Random Remove	8	17	94	221	779	
	Insert List	31	18	45	26	88	
Linked List	Add List	10	11	28	53	111	
	Random Read	8	27	172	448	1754	
	Random Remove	19	45	286	933	3681	
	Insert List	6	9	24	49	95	
Array List	Add List	6	8	32	77	103	20
	Random Read	1	2	12	13	27	
	Random Remove	11	17	91	222	769	
	Insert List	27	16	39	27	52	
Linked List	Add List	7	11	138	47	136	
	Random Read	11	27	163	436	1780	
	Random Remove	16	45	285	820	3998	
	Insert List	7	9	27	49	94	
Array	Add List	6	18	32	47	67	21
	Random Read	1	3	8	13	25	

	Random Remove	5	17	91	229	753	
	Insert List	4	6	18	37	88	
Linked List	Add List	5	11	49	45	110	
	Random Read	10	27	164	413	1773	
	Random Remove	20	44	286	840	3675	
	Insert List	10	19	23	72	96	
Array List	Add List	5	8	32	77	76	22
	Random Read	3	3	7	13	26	
	Random Remove	8	17	90	221	761	
	Insert List	22	20	37	26	87	
Linked List	Add List	6	11	28	46	117	
	Random Read	8	27	175	438	1767	
	Random Remove	20	45	288	796	3688	
	Insert List	12	9	23	49	96	
Array List	Add List	5	8	32	78	76	23
	Random Read	1	3	8	13	26	
	Random Remove	11	17	92	224	760	
	Insert List	29	16	37	26	89	
Linked List	Add List	7	11	28	47	109	
	Random Read	8	28	161	427	1750	
	Random Remove	15	45	280	807	3684	
	Insert List	7	9	24	49	100	
Array List	Add List	4	8	32	80	102	24
	Random Read	3	3	8	13	26	
	Random Remove	11	17	91	225	769	
	Insert List	19	6	26	26	51	
Linked List	Add List	12	21	39	46	123	
	Random Read	9	28	167	436	1814	
	Random Remove	17	49	298	858	3820	
	Insert List	8	9	24	49	94	
Array List	Add List	4	18	31	45	65	25
	Random Read	2	2	7	13	26	
	Random Remove	6	17	91	223	887	

	Insert List	3	6	16	37	92	
Linked List	Add List	7	12	49	47	115	
	Random Read	16	27	169	436	1794	
	Random Remove	13	47	289	823	3706	
	Insert List	5	19	23	72	95	