Grupo de Arquiteturas e Circuitos Integrados - GACI
Universidade Federal de Pelotas – UFPel
Pelotas – RS – Brasil

Generation and Analysis of Android Benchmarks with Different Algorithm Design Paradigms

Andrws Vieira, Cristian Bosin,

Luciano Agostini, Felipe Marques, Júlio Mattos

Outline

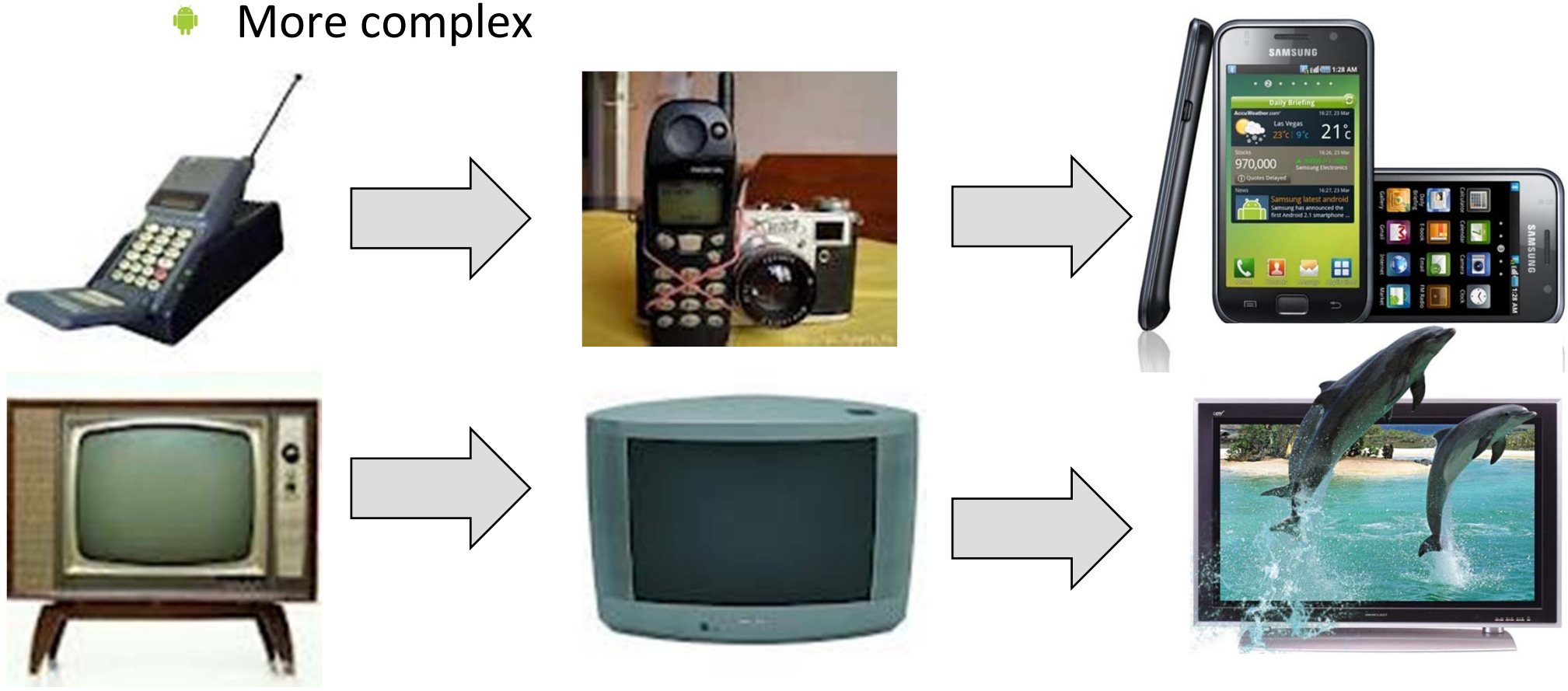
- Introduction
- Motivation
- Description of Developed Work
  - Results Achieved
- Conclusion and Future Works

Introduction

- Embedded Systems
  - An embedded system is a computational system designed for specific control functions “embedded” a larger system

Introduction

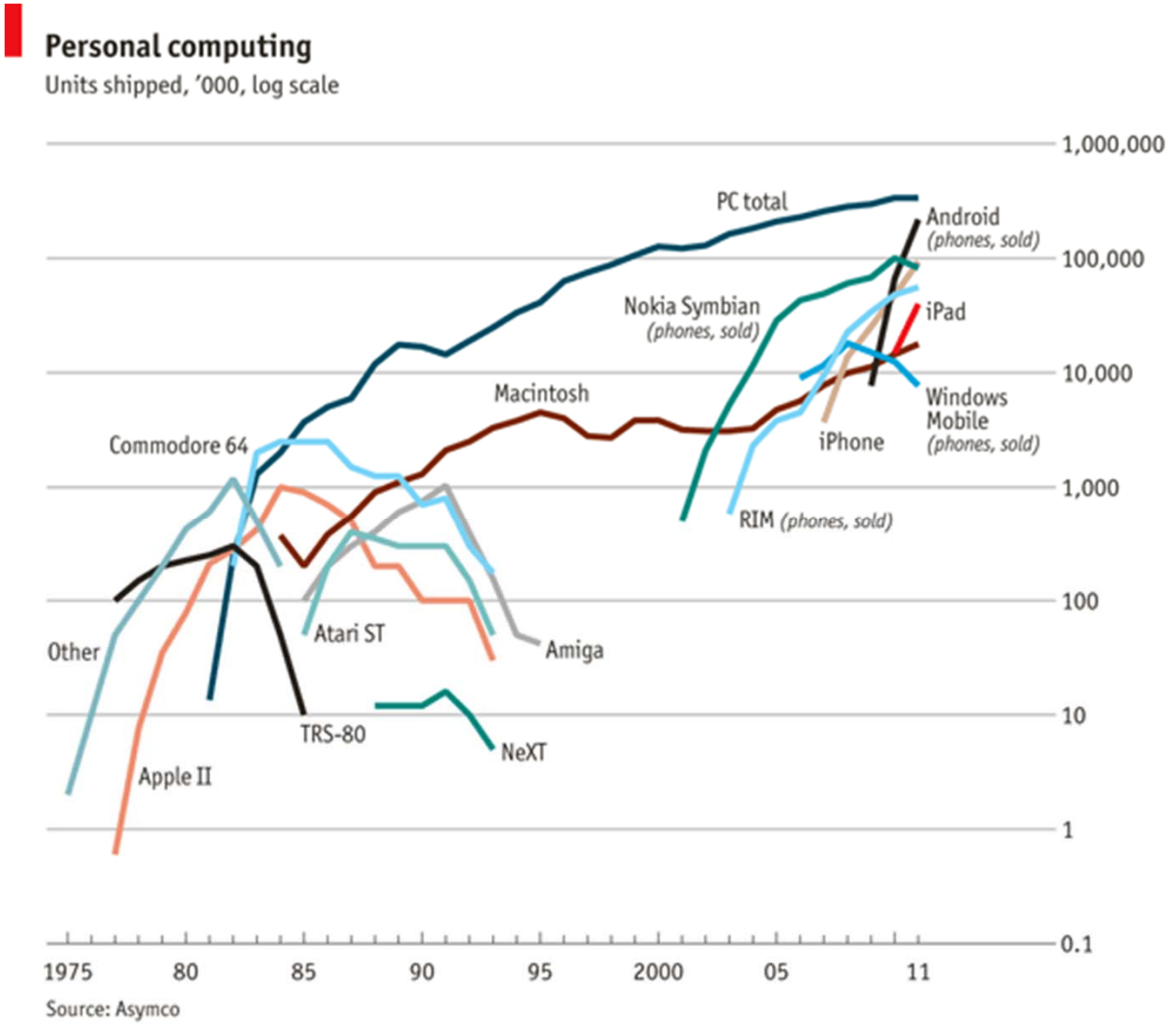
- Embedded Systems
  - More complex



Introduction

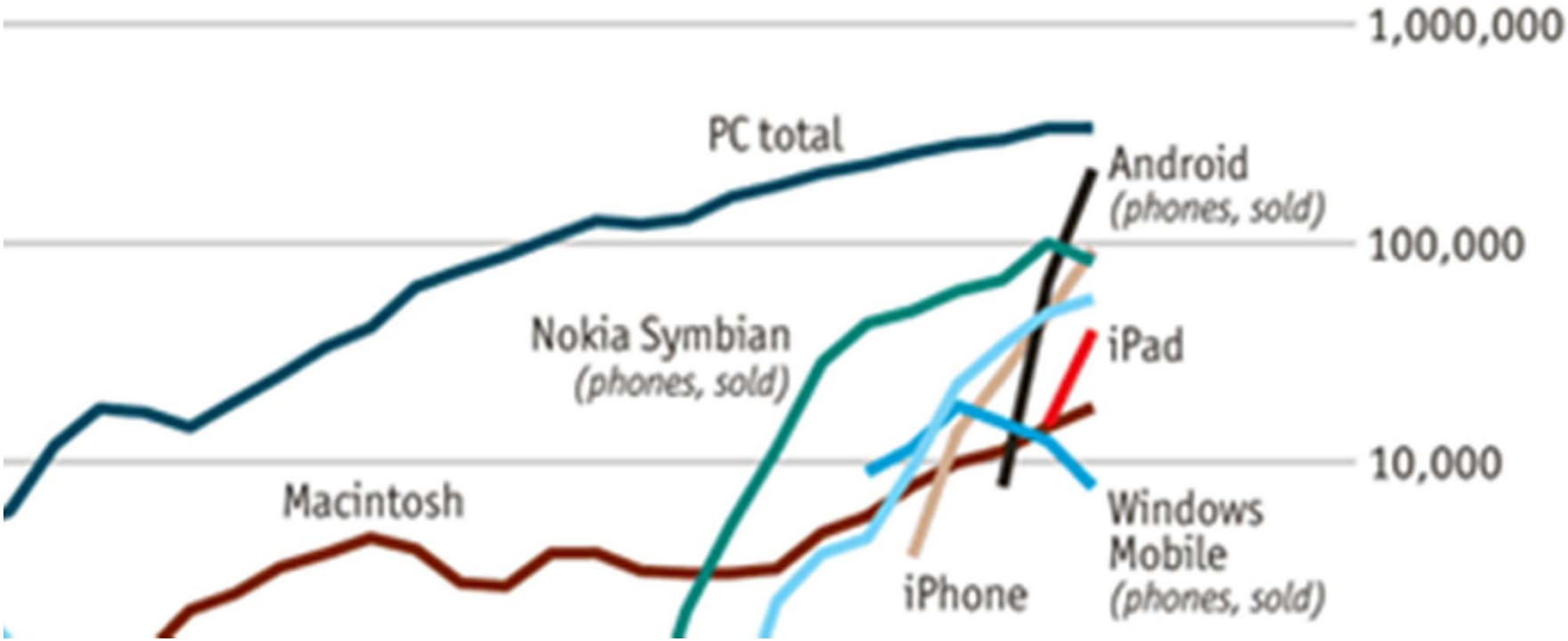
- Embedded Systems
  - Mobile systems (battery)
  - More than 6 billion consumers have a mobile phone (2012)

Motivation



Source: The Economist

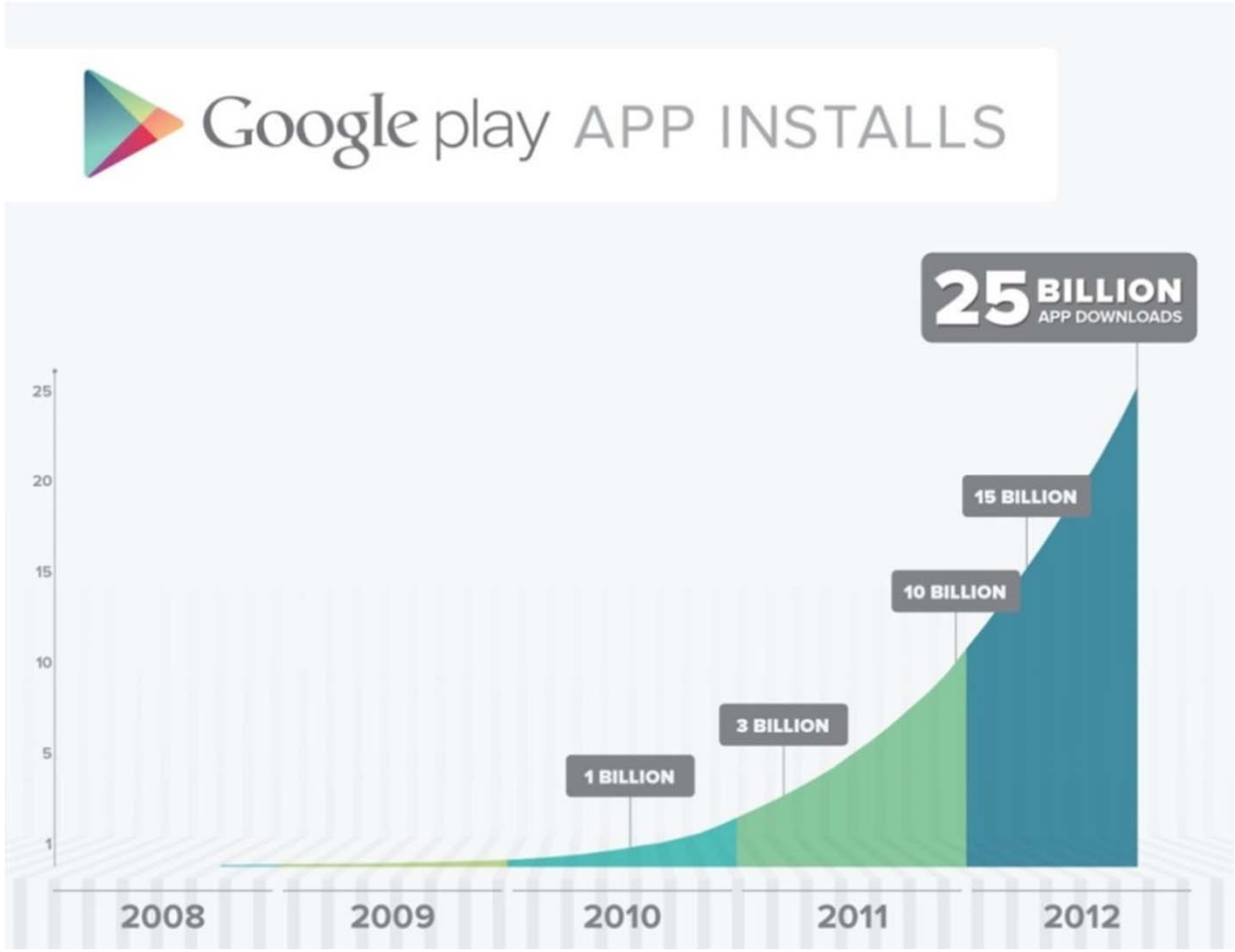
Motivation



500 million Android devices sold (September, 2012)

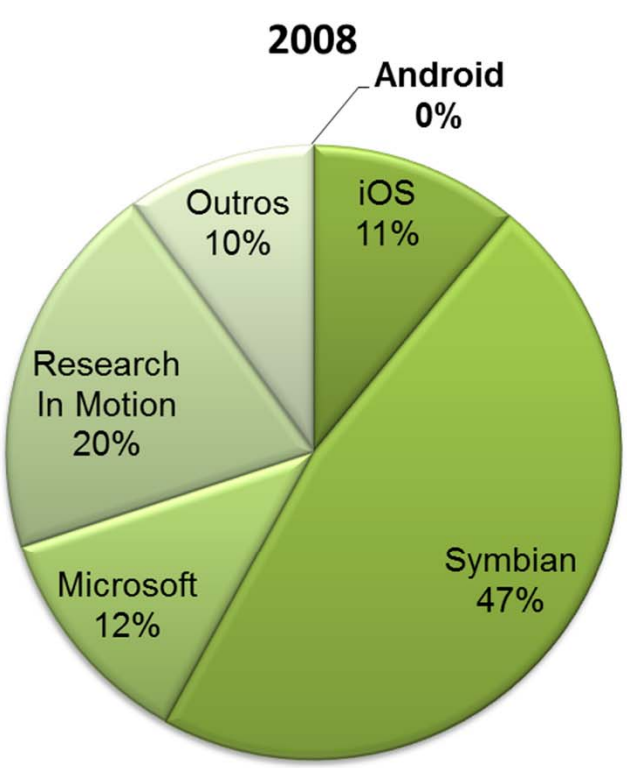
With 1.3 million devices sold daily

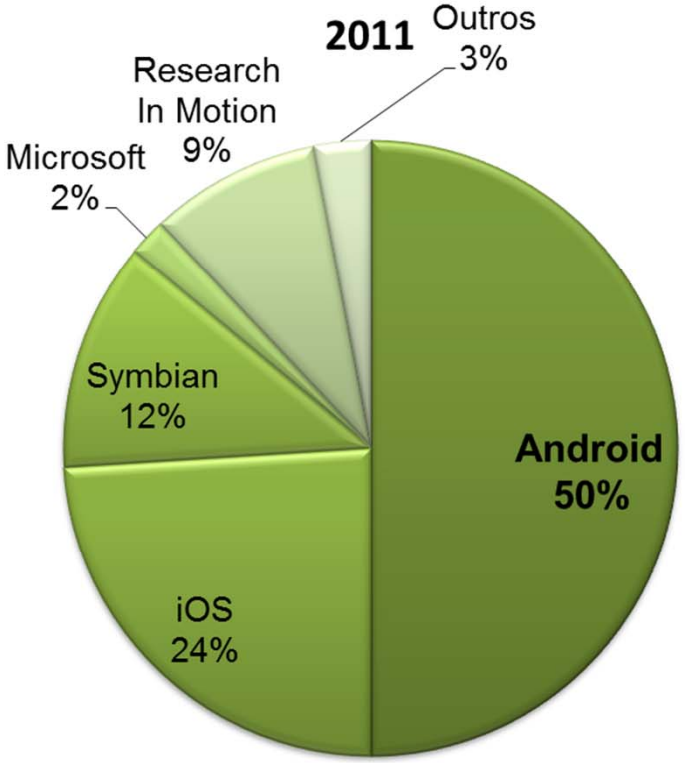
Motivation

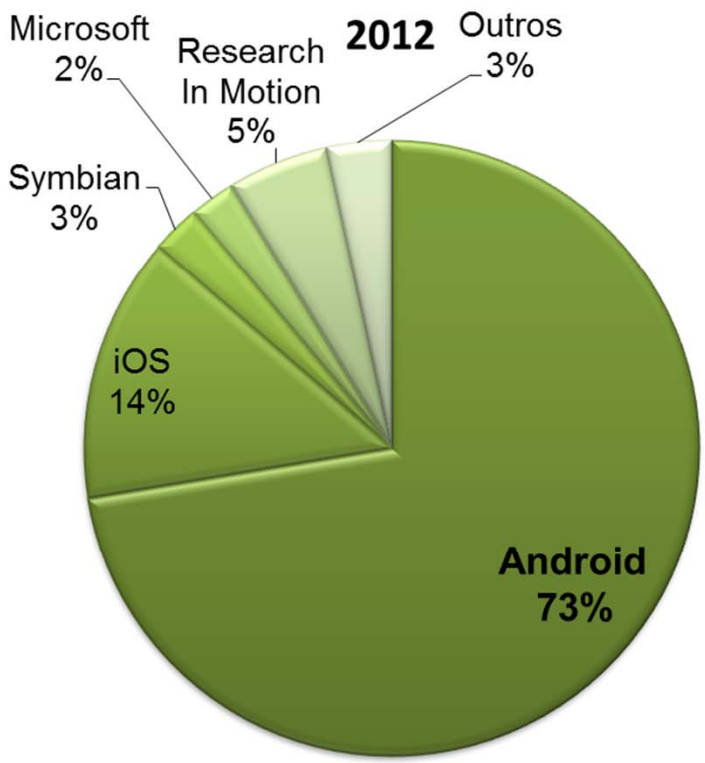


(Android, 2012).

Motivation


2008
Android
0%
iOS
11%
Outros
10%
Research
In Motion
20%
Symbian
47%
Microsoft
12%


2011
Outros
3%
Research
In Motion
9%
Microsoft
2%
Symbian
12%
Android
50%
iOS
24%


Microsoft
2%
Research
In Motion
5%
2012
Outros
3%
Symbian
3%
iOS
14%
Android
73%

Motivation

- The Android application development is simplified by its SDK
- It is necessary to analyze non-functional requirements, such as:
  - Performance;
  - Power dissipation;
  - Energy consumption.

Methodology

- Some algorithms were implemented for Android
  - For so analyze non-functional requirements
- To analyze the performance
  - Traceview Tool
- To analyze the power consumption
  - PowerTutor Tool

Description of Developed Work

- Design Paradigms Tested
  - Algorithms that uses more memory VS Algorithms that uses more CPU
  - Sorting Algorithms
  - Java Collection Framework – List Interface

Description of Developed Work

- Algorithms that uses more memory VS Algorithms that uses more CPU – Sine/Cosine
  - Trigonometric Functions – Sine/Cosine
  - Different forms calculate
    - Math Class (Java)
    - Taylor Series
    - Static tables

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

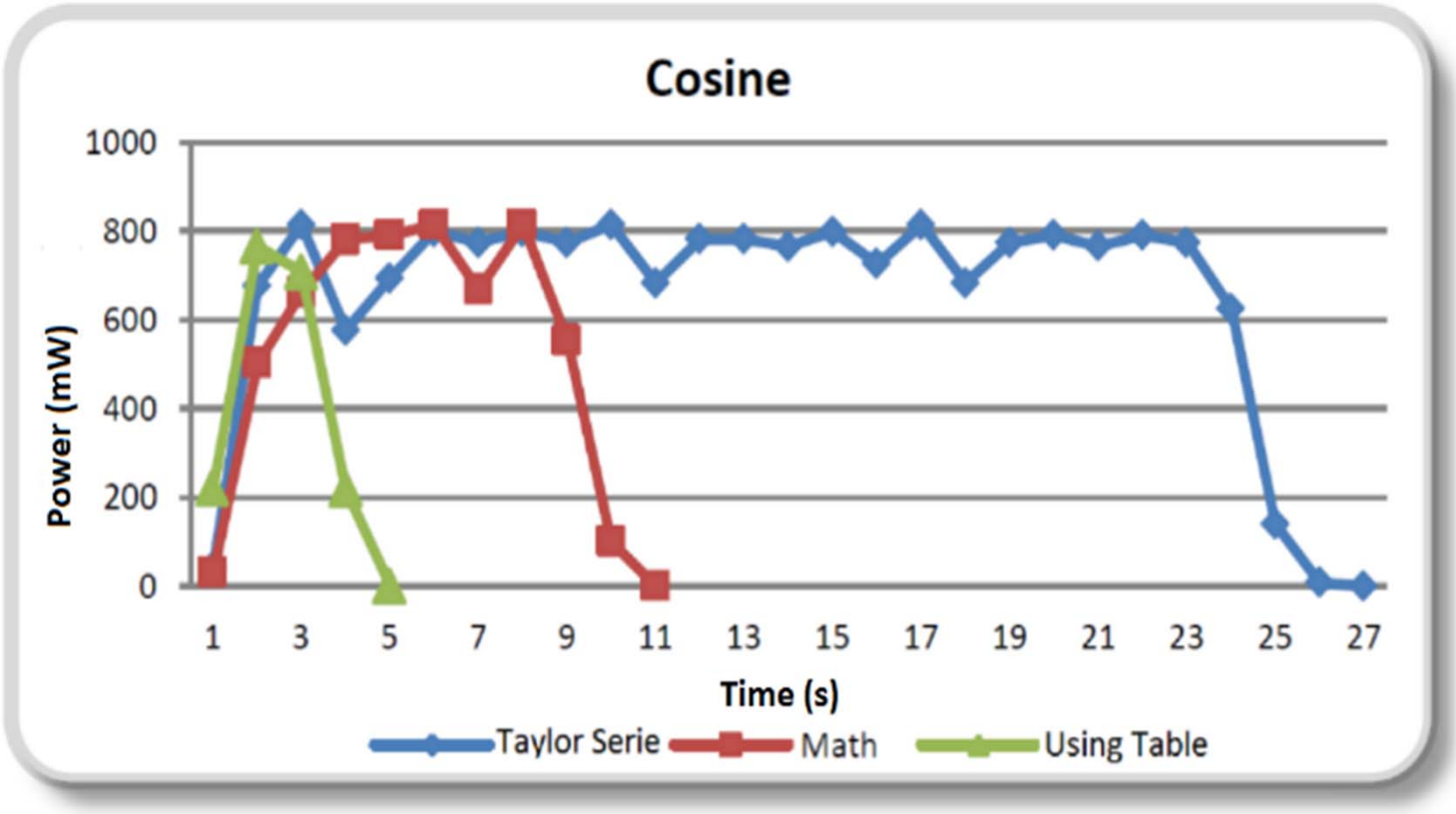
Description of Developed Work

- Algorithms that uses more memory VS Algorithms that uses more CPU – Sine/Cosine

Test 1	Taylor Series *900 values*	Math Class *900 values*	Static Table (using index) *900 values previously calculated*
Test 2	Taylor Series *3600 values*	Math Class *3600 values*	Static Table (using a hash function for index a table) *3600 values with 900 previously calculated*

Results Achieved

- Test 1 (cosine results*)



*Cosine and Sine results was the same performance

Results Achieved

- Test 1 (cosine results*)

Implementation	dex/apk size (KB)	Time (s)	Energy (mJ)
Taylor Series	433/160	26	17458
Using Table	443/167	5	1909
Math	433/160	10	5714

Static Table:

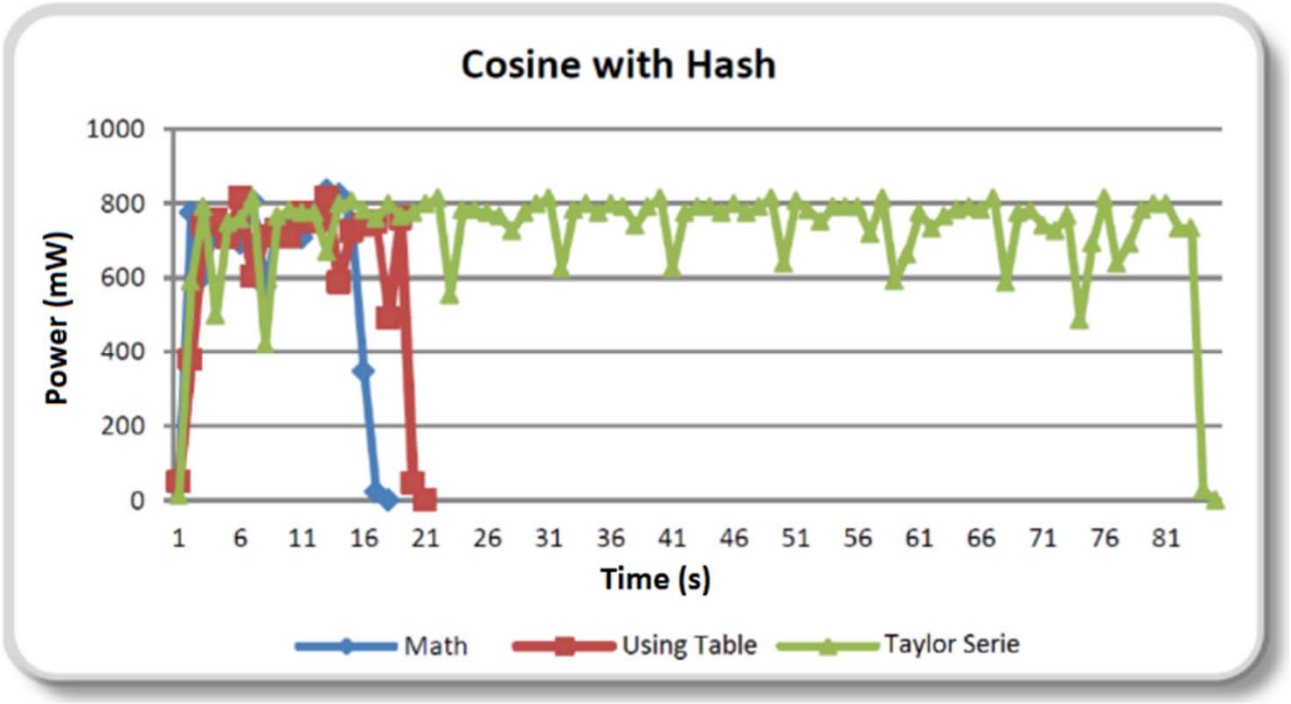
2x fast than Math

5x fast than Taylor Series

*Cosine and Sine results was the same performance

Results Achieved

- Test 2 (cosine results*)



*Cosine and Sine results was the same performance

Results Achieved

- In Test 1 → Static Table had best performance
- In Test 2 → Math Class had best performance

- Objective: usage of big tables in Android

- It's possible reduce CPU usage with a little increase in memory use.

Description of Developed Work

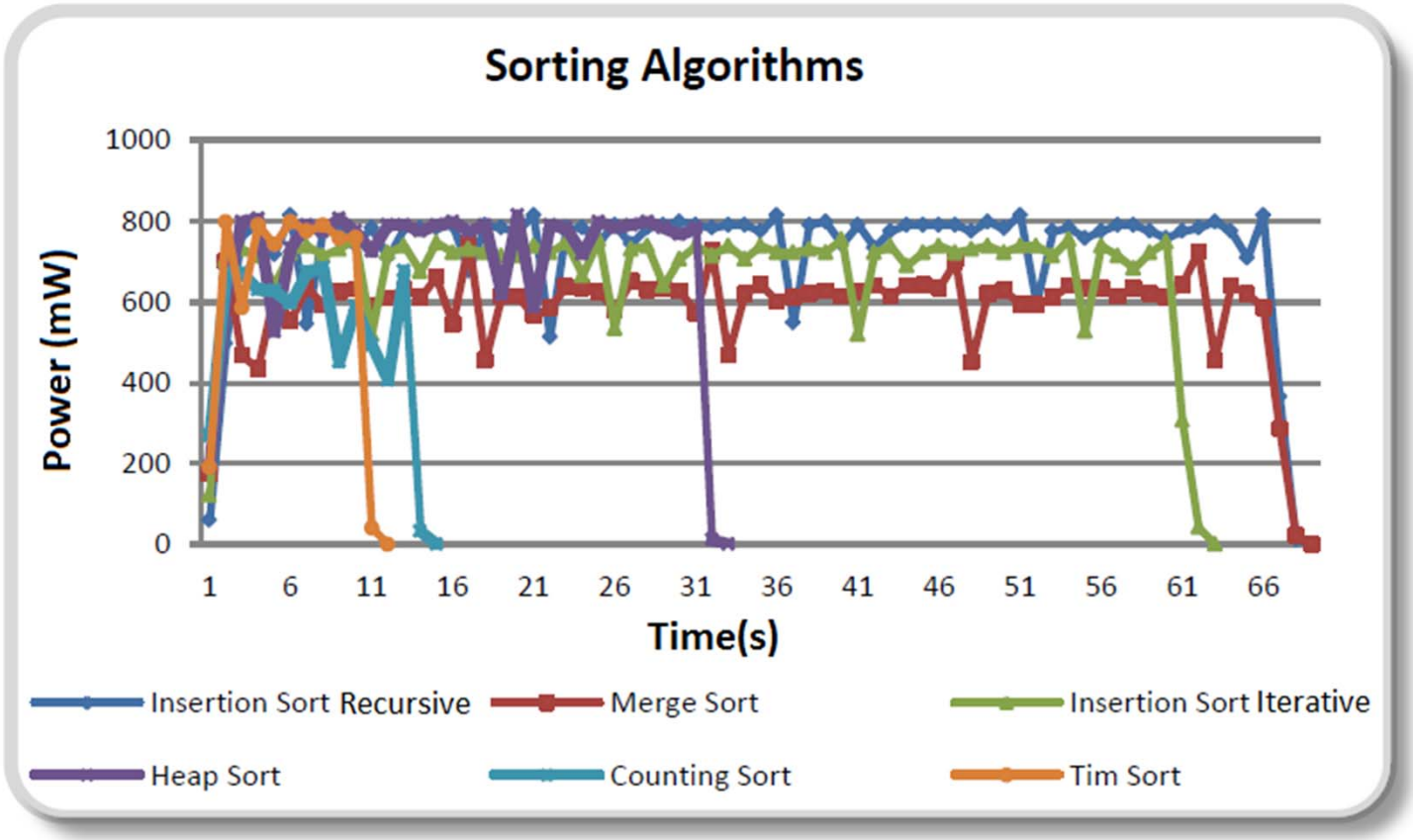
- Sorting Algorithms

Algorithms	Complexity
Insertion Sort Recursive	$O(n^2)$
Insertion Sort Iterative	$O(n^2)$
Merge Sort	$O(n \lg n)$
Heap Sort	$O(n \lg n)$
Counting Sort	$O(n + k^*)$
Tim Sort	$O(n \lg n)$

*k = max element

Results Achieved

- Sorting Algorithms

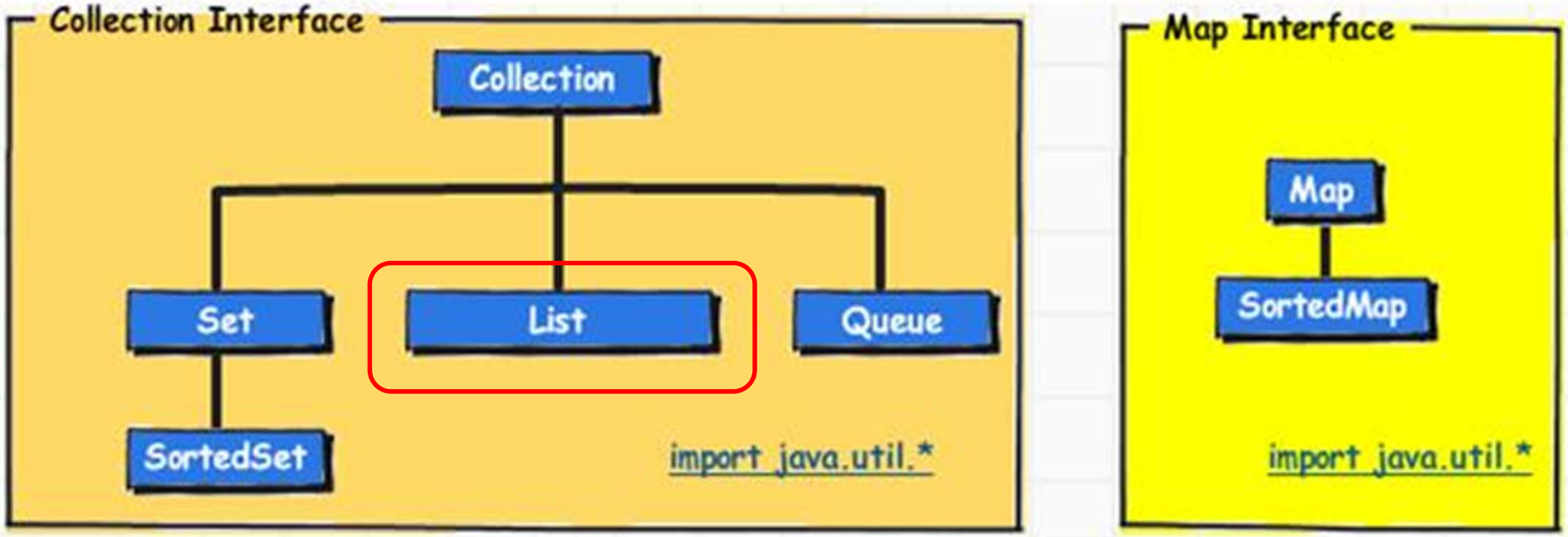


Results Achieved

- Sorting Algorithms
  - Insertion Sort Iterative – $O(n^2)$ – were more fast than Merge Sort – $O(n \lg n)$.
    - This happens because Merge Sort is a recursive algorithm
  - Tim Sort had best performance, running in 10 s.
    - Standard Java Sorting Algorithm.

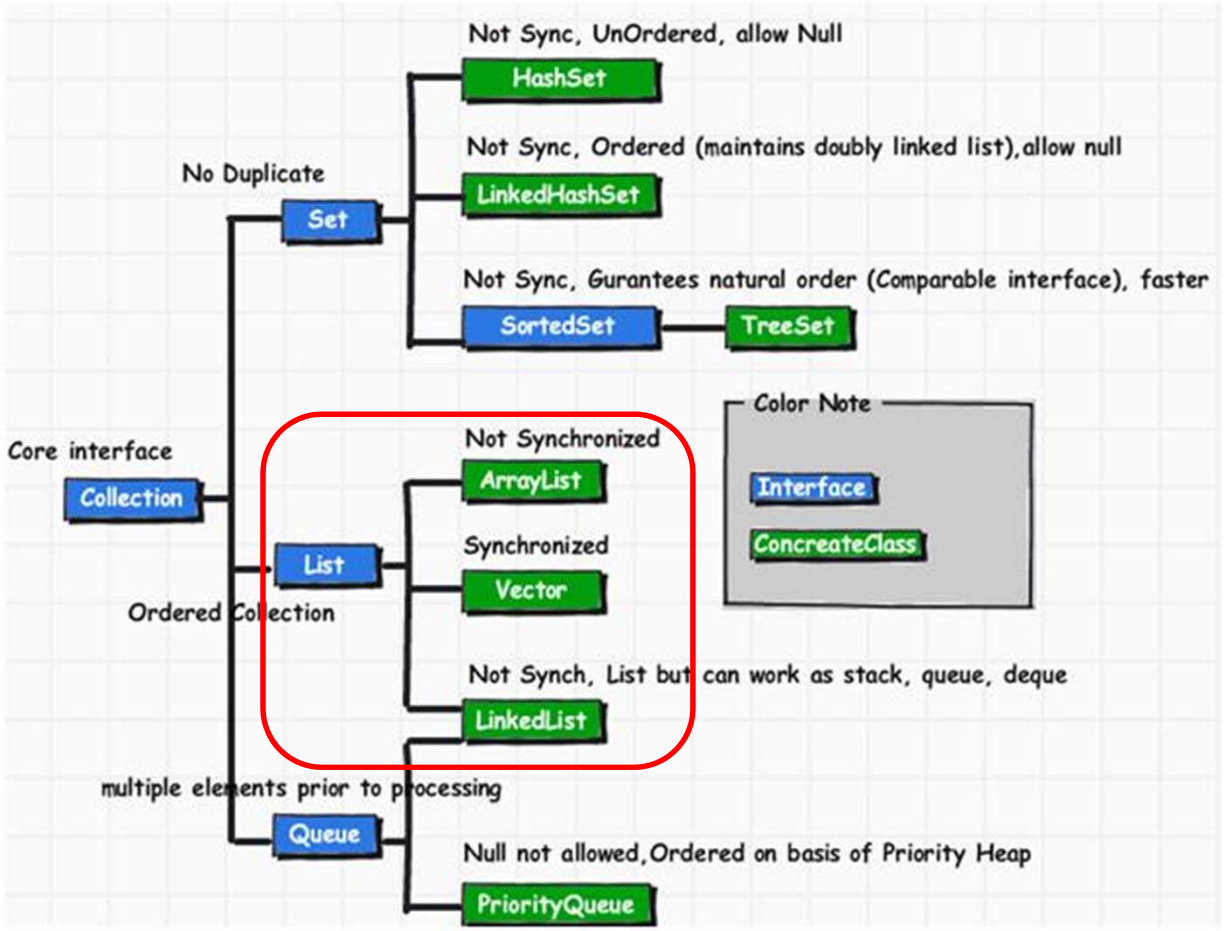
Description of Developed Work

- Java Collections Framework – List Interface



Description of Developed Work

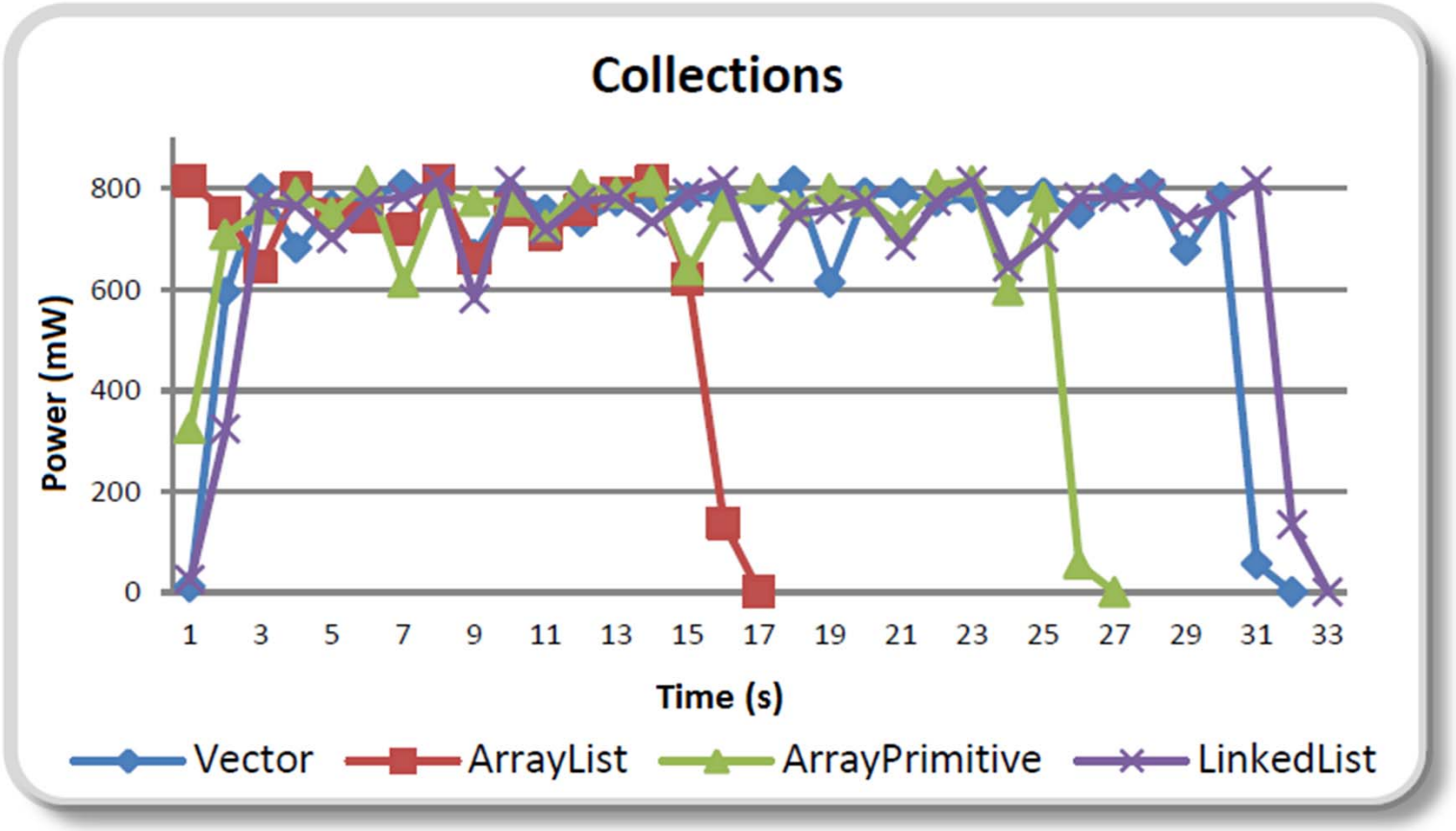
- Java Collections Framework – List Interface



And the java primitive array

Results Achieved

- Java Collections Framework – List Interface



Results Achieved

- Java Collections Framework – List Interface
  - ArrayList shown the best performance, running in 17s
  - LinkedList and Vector shown the worse one. This happens because LinkedList has linear search, and all Vector methods has synchronization, causing unnecessary overhead.

Conclusion

- Recursive algorithms show worse performance and more energy consumption
- A little memory increase causes a bigger CPU economy (lower energy consumption)
- Tim Sort were the best sorting algorithm
- In Java Collections Framework – List Interface, ArrayList shows better efficiency for developed benchmark.

Future Works

- Develop more benchmarks to evaluate
- Develop an automated tool to optimize the Android source code

Thank you

Questions?

Introduction

- Android is recently on the spotlight for mobile devices developers
  - software development kit (SDK), which enables a modern and agile development of applications
  - user interfaces that are visually elegant and easy to navigate.

What is Android?

- Android is a software stack for mobile devices that includes an operating system, middleware and key applications.

What is Android?

- The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.
- Android relies on Linux Kernel version 2.6 for core system service (the most recents use Linux Kernel version 3.0.31).

Methodology

- Traceview is a graphical viewer for execution logs that you creates by using the Debug class to log tracing information in your code
- It can help the debug an application and profile its performance
- It is possible to find bottlenecks in an application.

TRABALHOS RELACIONADOS

Ferramentas de Profiling/Tracing

Trabalhos relacionados

- **(PAUL e KUNDU, 2010) →** Comparação de desempenho entre Java Android para Dalvik versus o Java tradicional para JVM;
- **(KUNDU e PAUL, 2011) →** Apresenta um *framework* Java DSP permitindo que aplicativos *Android* executem paralelamente ;
- **(LIN, LIN, *et al.*, 2011) →** Apresenta uma análise entre JDK e NDK;
- **(BECKMANN, 2012) → A**presenta uma análise das boas práticas de codificação disponibilizadas pela Google para *Android*.

Ferramentas de Profiling/Tracing

- Ferramentas de *profiling*

  geralmente usadas em um programa e/ou código fonte específico para ver qual parte está usando mais recursos, gerando um perfil de execução.

- Ferramentas de tracing

  forma automatizada de observar o comportamento de um programa pela saída de informações sobre o estado atual, em intervalos regulares.

Ferramentas de Profiling/Tracing

- **PowerRunner** → software de gerenciamento automatizado de energia que prevê e otimiza o consumo de energia;
- **PowerBooter** → um sistema automatizado para medição de energia;
- **ANEPROF** → ferramenta de profiling que permite que seja possível obter um perfil de energia;
- **Sesame** → utiliza um paradigma de auto modelagem que um sistema móvel gera automaticamente o seu modelo de energia;
- **PowerTutor** → Detalhada a seguir ...
- **TraceView** → Detalhada a seguir ...

Traceview

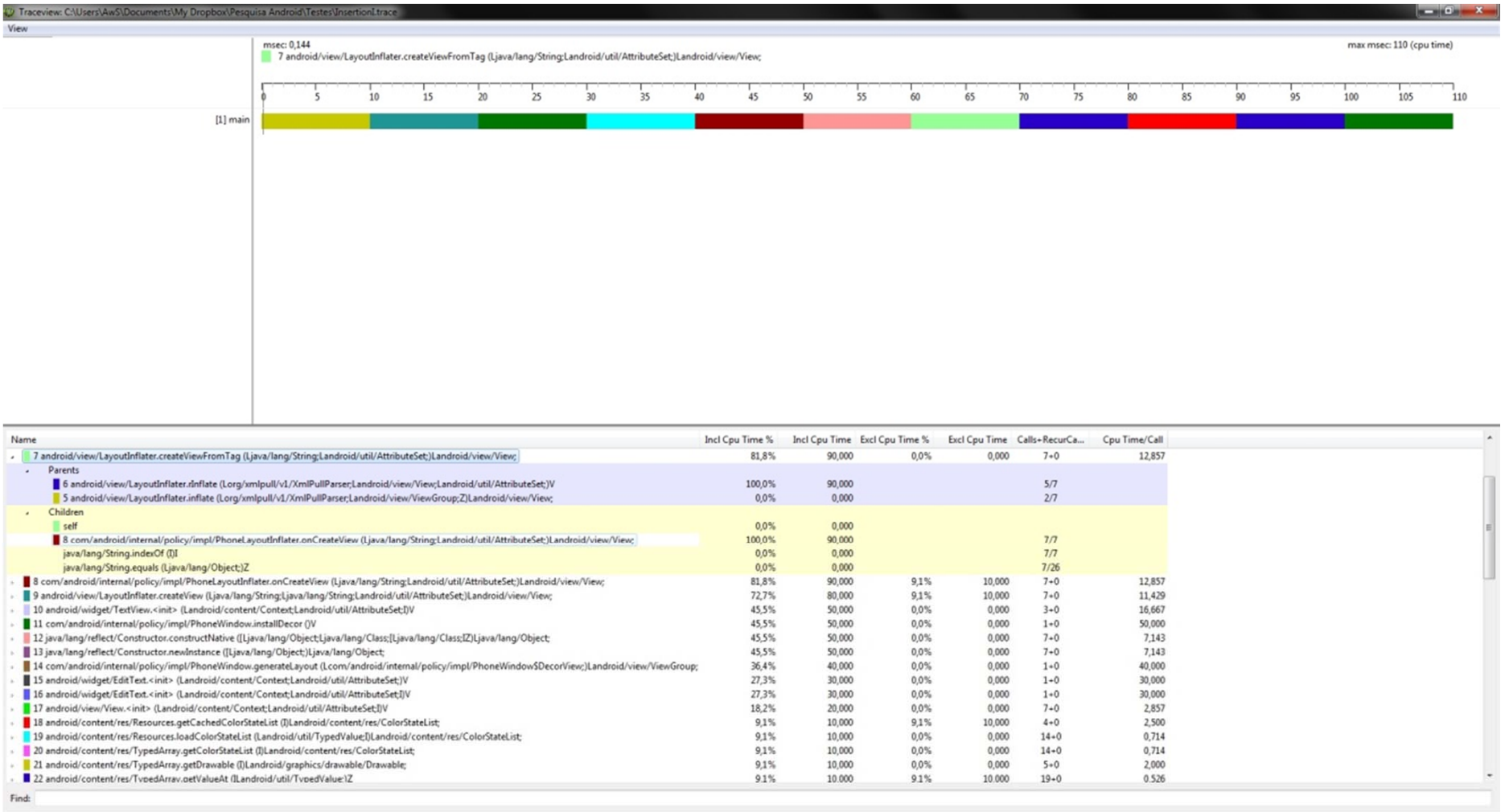
- *Traceview* é um visualizador gráfico para logs de execução;
- Registrar informações de profiling no código, ajudando a depurar a aplicação;
- Sendo possível localizar possíveis gargalos de desempenho na aplicação.

Traceview

- Para criar um arquivo de trace é muito simples:

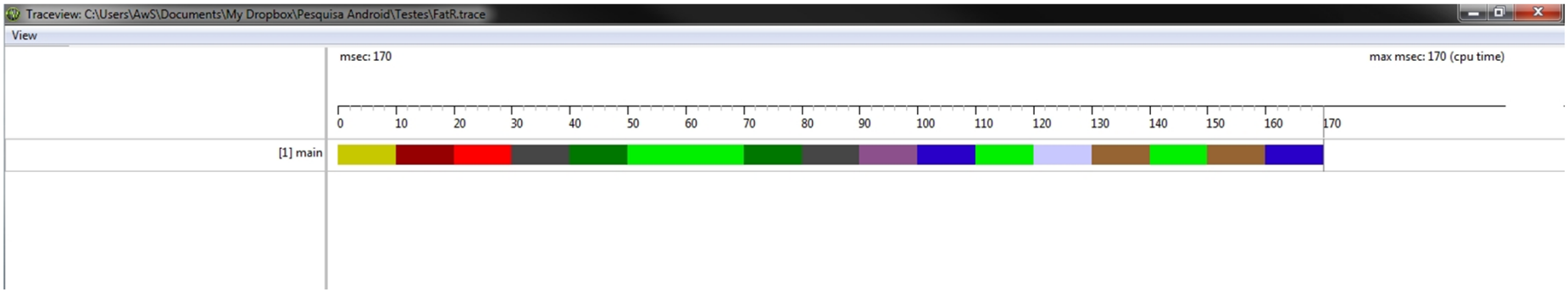
```
// start tracing to "/sdcard/TCC2013.trace"
Debug.startMethodTracing(“TCC2013");
// …
// stop tracing
Debug.stopMethodTracing();
```

Traceview


Traceview: C:\Users\AwS\Documents\My Dropbox\Pesquisa Android\Testes\InsertionI.trace
View
msec: 0,144
7 android/view/LayoutInflater.createViewFromTag (Ljava/lang/String;Landroid/util/AttributeSet;)Landroid/view/View;
max msec: 110 (cpu time)
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110
[1] main
Name
Incl Cpu Time %
Incl Cpu Time
Excl Cpu Time %
Excl Cpu Time
Calls+RecurCa...
Cpu Time/Call
7 android/view/LayoutInflater.createViewFromTag (Ljava/lang/String;Landroid/util/AttributeSet;)Landroid/view/View; 81,8% 90,000 0,0% 0,000 7+0 12,857
Parents
6 android/view/LayoutInflater.rInflate (Lorg/xmlpull/v1/XmlPullParser;Landroid/view/View;Landroid/util/AttributeSet;)V 100,0% 90,000 5/7
5 android/view/LayoutInflater.inflate (Lorg/xmlpull/v1/XmlPullParser;Landroid/view/ViewGroup;Z)Landroid/view/View; 0,0% 0,000 2/7
Children
self 0,0% 0,000
8 com/android/internal/policy/impl/PhoneLayoutInflater.onCreateView (Ljava/lang/String;Landroid/util/AttributeSet;)Landroid/view/View; 100,0% 90,000 7/7
java/lang/String.indexOf (I)I 0,0% 0,000 7/7
java/lang/String.equals (Ljava/lang/Object;)Z 0,0% 0,000 7/26
8 com/android/internal/policy/impl/PhoneLayoutInflater.onCreateView (Ljava/lang/String;Landroid/util/AttributeSet;)Landroid/view/View; 81,8% 90,000 9,1% 10,000 7+0 12,857
9 android/view/LayoutInflater.createView (Ljava/lang/String;Ljava/lang/String;Landroid/util/AttributeSet;)Landroid/view/View; 72,7% 80,000 9,1% 10,000 7+0 11,429
10 android/widget/TextView.<init> (Landroid/content/Context;Landroid/util/AttributeSet;I)V 45,5% 50,000 0,0% 0,000 3+0 16,667
11 com/android/internal/policy/impl/PhoneWindow.installDecor ()V 45,5% 50,000 0,0% 0,000 1+0 50,000
12 java/lang/reflect/Constructor.constructNative ([Ljava/lang/Object;Ljava/lang/Class;[Ljava/lang/Class;IZ)Ljava/lang/Object; 45,5% 50,000 0,0% 0,000 7+0 7,143
13 java/lang/reflect/Constructor.newInstance ([Ljava/lang/Object;)Ljava/lang/Object; 45,5% 50,000 0,0% 0,000 7+0 7,143
14 com/android/internal/policy/impl/PhoneWindow.generateLayout (Lcom/android/internal/policy/impl/PhoneWindow$DecorView;)Landroid/view/ViewGroup; 36,4% 40,000 0,0% 0,000 1+0 40,000
15 android/widget/EditText.<init> (Landroid/content/Context;Landroid/util/AttributeSet;)V 27,3% 30,000 0,0% 0,000 1+0 30,000
16 android/widget/EditText.<init> (Landroid/content/Context;Landroid/util/AttributeSet;I)V 27,3% 30,000 0,0% 0,000 1+0 30,000
17 android/view/View.<init> (Landroid/content/Context;Landroid/util/AttributeSet;I)V 18,2% 20,000 0,0% 0,000 7+0 2,857
18 android/content/res/Resources.getCachedColorStateList (I)Landroid/content/res/ColorStateList; 9,1% 10,000 9,1% 10,000 4+0 2,500
19 android/content/res/Resources.loadColorStateList (Landroid/util/TypedValue;I)Landroid/content/res/ColorStateList; 9,1% 10,000 0,0% 0,000 14+0 0,714
20 android/content/res/TypedArray.getColorStateList (I)Landroid/content/res/ColorStateList; 9,1% 10,000 0,0% 0,000 14+0 0,714
21 android/content/res/TypedArray.getDrawable (I)Landroid/graphics/drawable/Drawable; 9,1% 10,000 0,0% 0,000 5+0 2,000
22 android/content/res/TypedArray.getValueAt (ILandroid/util/TypedValue;)Z 9,1% 10,000 9,1% 10,000 19+0 0,526
Find:

Traceview

- Timeline Panel



- Para cada aplicação, todas as chamadas de métodos são apresentadas (com diferentes cores)

Traceview

- Profile Panel

Name	Incl Cpu Time %	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Calls+RecurCa...	Cpu Time/Call
10 java/lang/reflect/Constructor.constructNative ([Ljava/lang/Object;Ljava/lang/Class;[Ljava/l	70,6%	120,000	0,0%	0,000	11+0	10,909
Parents						
11 java/lang/reflect/Constructor.newInstance ([Ljava/lang/Object;)Ljava/lang/Object;	100,0%	120,000			11/11	
Children						
self	0,0%	0,000				
14 android/widget/EditText.<init> (Landroid/content/Context;Landroid/util/AttributeS	41,7%	50,000			2/2	
41 android/widget/TextView.<init> (Landroid/content/Context;Landroid/util/Attribute	16,7%	20,000			3/3	
40 android/widget/LinearLayout.<init> (Landroid/content/Context;Landroid/util/Attrit	16,7%	20,000			3/3	
38 android/widget/Button.<init> (Landroid/content/Context;Landroid/util/AttributeSe	16,7%	20,000			1/1	
70 android/widget/FrameLayout.<init> (Landroid/content/Context;Landroid/util/Attrit	8,3%	10,000			2/2	
11 java/lang/reflect/Constructor.newInstance ([Ljava/lang/Object;)Ljava/lang/Object;	70,6%	120,000	0,0%	0,000	11+0	10,909
12 android/widget/TextView.<init> (Landroid/content/Context;Landroid/util/AttributeSet;I)V	52,9%	90,000	0,0%	0,000	6+0	15,000
13 android/view/View.<init> (Landroid/content/Context;Landroid/util/AttributeSet;I)V	29,4%	50,000	0,0%	0,000	11+0	4,545
14 android/widget/EditText.<init> (Landroid/content/Context;Landroid/util/AttributeSet;)V	29,4%	50,000	0,0%	0,000	2+0	25,000
15 android/widget/EditText.<init> (Landroid/content/Context;Landroid/util/AttributeSet;I)V	29,4%	50,000	0,0%	0,000	2+0	25,000
16 com/android/internal/policy/impl/PhoneWindow.installDecor ()V	29,4%	50,000	0,0%	0,000	1+0	50,000
17 android/content/res/AssetManager.applyStyle (IIII[I[I[I[I)Z	23,5%	40,000	23,5%	40,000	60+0	0,667
18 android/content/res/Resources$Theme.obtainStyledAttributes (Landroid/util/AttributeSet;[	23,5%	40,000	0,0%	0,000	53+0	0,755

- Tempo de Execução, Chamada de Métodos, …

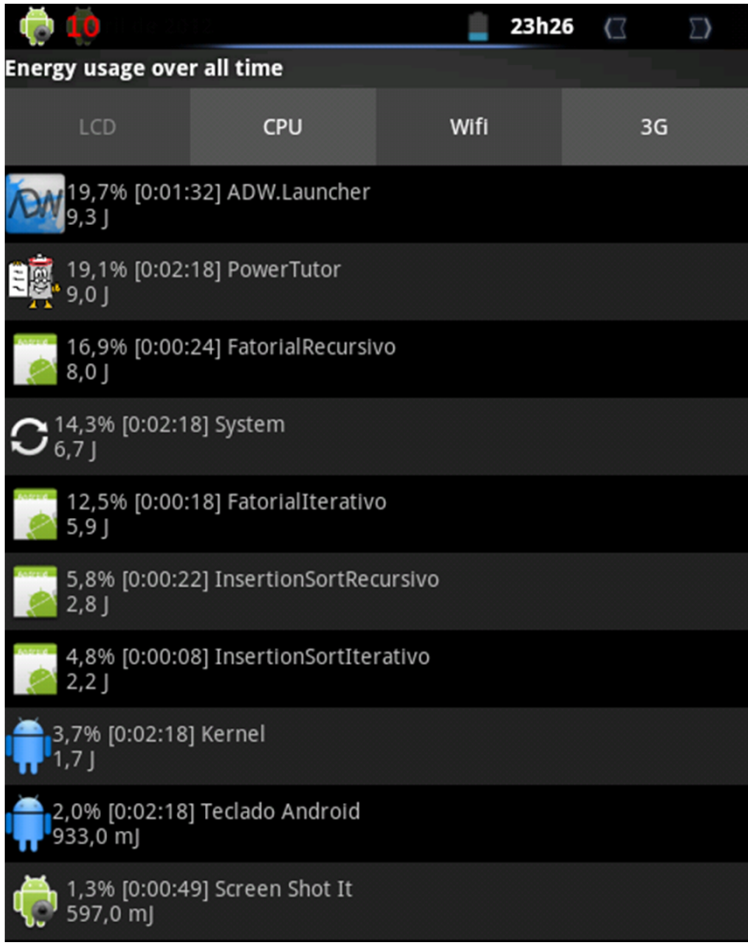
PowerTutor 

- PowerTutor é uma aplicação para dispositivos Android que fornece dados de potência dissipada pelos principais componentes do sistema, tais como:
  - CPU,
  - Interface de Rede,
  - Display,
  - GPS,
  - …

PowerTutor 

- PowerTutor provê uma configurável história de consumo de energia
- Fornece Log em formato de texto que contém os resultados detalhados.

PowerTutor


10
23h26
Energy usage over all time
LCD
CPU
Wifi
3G
19,7% [0:01:32] ADW.Launcher
9,3 J
19,1% [0:02:18] PowerTutor
9,0 J
16,9% [0:00:24] FatorialRecursivo
8,0 J
14,3% [0:02:18] System
6,7 J
12,5% [0:00:18] FatorialIterativo
5,9 J
5,8% [0:00:22] InsertionSortRecursivo
2,8 J
4,8% [0:00:08] InsertionSortIterativo
2,2 J
3,7% [0:02:18] Kernel
1,7 J
2,0% [0:02:18] Teclado Android
933,0 mJ
1,3% [0:00:49] Screen Shot It
597,0 mJ


10
20h12
Chart View
Pie View
Stat View
Total (mW)
2250.0
1500.0
750.0
0.0
LCD (mW)
1050.0
700.0
350.0
0.0
CPU (mW)
600.0
400.0
200.0
0.0
3G (mW)
600.0
400.0
200.0
1
13
25
37
49
61
73
85
97
109
120

PowerTutor

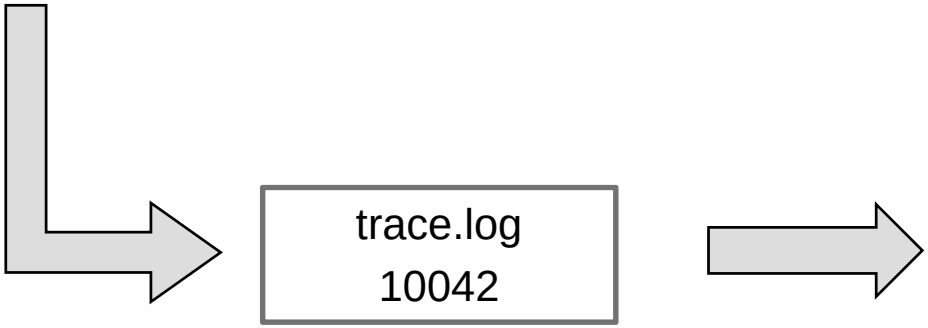
- Log

```
Audio-on false
associate 10042 com.example.linkedlisttest@1
begin 50
total-power 1073
meminfo 179624 10696 536 49888
LCD 900
LCD-brightness 255
LCD-screen-on true
LCD-10042 900
CPU 173
CPU-sys 40
CPU-usr 0
CPU-freq 720.0
CPU-0 40
CPU-1000 200
CPU-1002 0
```

PowerTutor

```
fileName = io.read()
id = "CPU-" .. io.read()

for line in io.lines(fileName) do
    word, word2 = line:match"^(%S+) (.*)$"
    if word == id then print (word2) end
end
```




```
0
814
750
643
798
748
741
718
814
660
757
708
757
790
814
619
136
0
```

Routine – JCF Benchmark

This set of operations was defined in six steps and each step run the following routine:

(a) Values are inserted into a list;

(b) The elements are removed one by one, in increasing order;

(c) The same vector is reinserted;

(d) The elements are removed one by one, in decreasing order;

(e) The same vector is reinserted;

(f) The elements are removed at random order (predetermined);

Each step runs this way:

(1) Run the routine with an increasing sorted vector;

(2) Same as with step 1, but with a decreasing sorted vector;

(3) Same as with step 1, but with a random vector;

(4) Same as with step 3, but after insertions, reverse the list;

(5) Same as with step 4, but sort the list;

(6) Same as with step 4, but reverse sort the list.