BENCHMARK FOR MOBILE DEVICES

Name: Șandor Cristian-Ionuț

Group: 30431

1. **Project proposal**

The objective of this project is to implement a benchmark application on Android which evaluates the performance of a mobile device and displays its representative score after the evaluation. This application should run multiple tests for the processor and memory of the device.

1. **Plan**

|  |  |
| --- | --- |
| 28.09.2020 – 11.10.2020 | - first week of laboratory; choosing the project |
| 12.10.2020 – 25.10.2020 | - documenting regarding the objective and the concepts of the project, planning the working schedule over the weeks  - planning an overall design and thinking about how the application should work and look |
| 26.10.2020 – 08.11.2020 | - experimenting with the Android Studio IDE  - experimenting with different possible algorithms and functions  - planning the user interface |
| 09.11.2020 – 22.11.2020 | - starting the implementation according to the advices and feedback of the teacher  - working on the algorithms and functionality for the computing evaluation of the application |
| 23.11.2020 – 06.12.2020 | - implementing the benchmark for the memory performance of the device and completing the user interface according to the sketch planned before |
| 07.12.2020 – 20.12.2020 | - completing the last tasks for the project and also completing the documentation for the project  - reviewing and testing the functionality  - presenting the application |

1. **Bibliographic study**

A benchmark, in terms of computer science, represents the act of running a program, an application or other operations in order to estimate the performance of a computer device, by running a set of standard tests and trials against it. This term is usually associated with assessing performance characteristics of computer hardware, for example, the floating-point operation performance of a processor, but this technique can also be applicable to software. Benchmarks represents a method for comparing the performance of various subsystems across different chip/system architectures.[1]

Benchmarks provide a way to measure real-world performance without getting bogged down in comparing specs. There is no need in knowing how many cores a processor has or how much RAM memory in order to compare the devices You can compare them only by the score resulted after the benchmark. There are multiple ways to test a device, for example calculation tests to measure a CPU’s processor power, memory tests to measure how fast you can read and write data or for example 2D and 3D tests to measure a graphics card’s performance. [2]

Characteristics a benchmark should have:

1. Relevance: Benchmarks should measure relatively vital features.
2. Representativeness: Benchmark performance metrics should be broadly accepted by industry and academia.
3. Equity: All systems should be fairly compared.
4. Repeatability: Benchmark results can be verified.
5. Cost-effectiveness: Benchmark tests are economical.
6. Scalability: Benchmark tests should work across systems possessing a range of resources from low to high.
7. Transparency: Benchmark metrics should be easy to understand.[3]

A benchmark tests the performance of a system based on execution time, which is inversely proportional to performance.

Benchmark applications:

**Antutu Benchmark** is one of the leading benchmarking apps. Available for free on both Android and iOS, this app puts your phone’s processor and RAM through their paces. This app gives detailed information about your battery level and its internal temperature meaning that you can see at a glance how your handset is holding up.

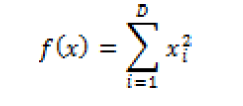
**Primate Lab’s Geekbench** test the processor speed and memory performance of any phone and take the guesswork out of benchmarking. Geekbench scores are easy to understand and the app keeps a record of your scores over time. This makes it easy to identify any degradation in your phone’s speed.

In common with Antutu, Geekbench runs tests that simulate real-world tasks. Geekbench 4 can differentiate between single and multi-core CPUs to ensure that it is measuring the full potential of your device. You can also measure your phone’s GPU with Geekbench 4’s new GPU Compute tests.

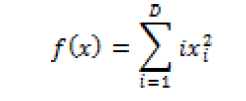
**3DMark** app benchmarks the CPU and GPU in your device and gives you a score. You can compare your score with those of other devices around the world and see whether it’s time for an upgrade. OS upgrades are more demanding and can be a major reason behind smartphones slowing down. 3DMark lets you see how your phone’s speed has been affected by each OS upgrade.[4]

I managed to find a document with functions that are commonly used in benchmarks. There is no standard list of mathematical functions since different researchers choose different set of benchmark test functions. A couple of functions found in the document are:

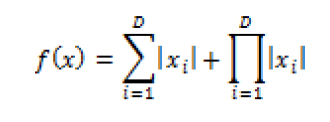
1. *Sphere*: Sphere is a unimodal and continuous function, which is considered as easy to solve. This function is evaluated using range between [*-5.12,5.12*] and its minimum solution is *0* which is located at *f(x\*)*=[*0,0,…,0*]. It is mathematically expressed as: [5]



1. SumSquare: This function is also known as Axis Parallel Hyper-Ellipsoid function which maintains no local optima but single global optima f(x\*) = [0, 0, …, 0]. The function is normally evaluated with continuous values within the range of [-10,10]. The function expressed as: [5]



1. Schwefel 2.22: This function is a unimodal with search space usually spread over [-10,10] values. The global minimum 0 is located at f(x\*) = [0, 0, …, 0].[5]



1. **Analysis and design**

In this application I must build a suitable environment for a user of a mobile device to safely and easily evaluate the performance of their device. I have in mind to meet the general requirements and standards of a benchmark application, so I will project it according to the gathered information. The project will be developed for Android users so the programming language I am going to use will be Java. With that being said, I need a way of getting information about the execution time. This is computed by subtracting the initial time when the program started from the final time when the computation stops. I am willing to use some methods already existent in java to get these values of time such as System.currentTimeInMillis() or System.nanoTime().

The problem basically divides in multiple tasks, which I will implement one. Firstly, I am looking forward to evaluate the computing performance of the device. One way of getting the desired result is by calculating the prime numbers in a long interval, calculating the first n decimals of pi or using a mathematical function like the ones already found. I am going to spend some time on finding the perfect configuration for this section since I want it to represent the result as close to reality as possible.

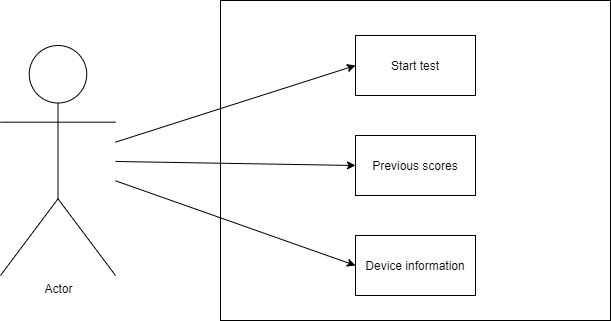
For the other type of benchmark, the component for which I will compute the performance is the memory for which I am planning to do multiple operations on memory locations, working with variables, the stack and files to see how fast the execution time will be.

All this data gathered in the previous steps will be processed and then display on the front page as a score when the entire testing is over and maybe saved somewhere to be compared with other results.

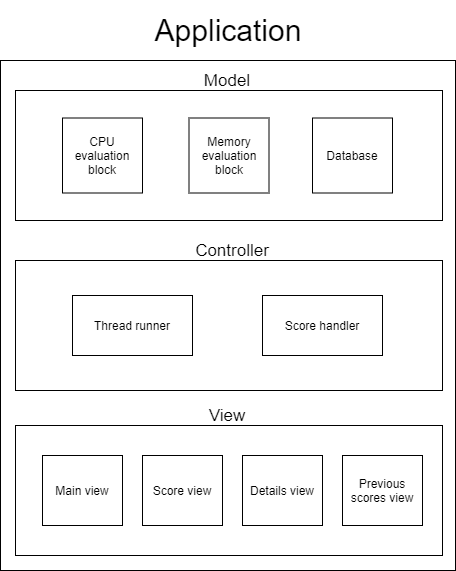
This application will also have a simple GUI in which with the press of a button the program will start. The algorithms will be started on different threads for each part individually and the scores will be gathered from every part and computed in an overall score. The score will be displayed on the screen and saved on another tab among other results which will all be saved in a database. This would be displayed on another page than the main one with the score.

Another functionality I want to implement is to give the user useful information about the device. In this tab the user is able to see the phone brand, model, serial number, display and more.

Use case:



Architecture diagram:



1. **Implementation**

As I mentioned before each section that should be tested would have an individual class. We are doing this in order to manage better the distinct threads and to gather the information separately.

In the CPU class I designed the specific code which will be used to bench the CPU component. There are two algorithms used for this testing and those are the decomposition in prime factors applied on the first 200000 natural numbers and the computation of the first 500 decimals of pi. I defined the method primeFactors(int number) which takes an integer as parameter and in a for from 2 to that number, verifies if the iterated numbers are prime factors. It does this by dividing the number as long as the remainder is 0. All the prime factors are then added to an ArrayList. The program which I am measuring is calculating the prime factors for all the numbers from 1 to 200000.

Calculating the first n decimals of pi:

import java.math.\*;

static void c(int n){

BigInteger p, a = p = BigInteger.TEN.pow(10010).multiply(new BigInteger("2"));

for(int i = 1; a.compareTo(BigInteger.ZERO) > 0; p = p.add(a)){

a = a.multiply(new BigInteger(i+"")).divide(new BigInteger((2 \* i++ + 1)+""));

}

for(int i = 0; i < n; i++){

System.out.printf ((p+"").charAt(i+1) - 48));

}

}

In the cpuScore method I took the initial time with System.*currentTimeMillis*() and then I called the primeFactors and piDecimals functions. After the call of these two methods, I got the exectution time with System.currentTimeMillis() – initTime.

For the memory testing of the performance, I chose to use the quicksort algorithm. I did this because it makes modifications on an array in memory and moreover the stack will be solicitated on array of large sizes. The method createArray initializes a static array and generates 1000000 random elements, one for each position of the array. I also tested the performance of the memory by writing and reading from a text file. I generated a random string[7] of 1000000 characters which I wrote in a file with the help of FileOutputStream, and then I read the same string with the help of FileInputStream. In the memScore method I followed a similar manner with the CPU test. I took the initial time with System.*currentTimeMillis*() and then I generated the array and called the quicksort on it. After the execution of these two methods, I returned the execution time with System.currentTimeMillis() – initTime.

Sorting algorithm (used on 1000000 elements array):

private static int partition(int arr[], int low, int high)

{

int pivot = arr[high];

int i = (low-1); // index of smaller element

for (int j=low; j<high; j++)

{

if (arr[j] < pivot)

{

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i+1];

arr[i+1] = arr[high];

arr[high] = temp;

return i+1;

}

private static void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi-1);

quickSort(arr, pi+1, high);

}

}

The main activity of the application is the main page which has 3 buttons “start test”, “scores”, and “device info”.

When device info is pressed, the DeviceInfo activity will start in which are displayed some details about the used device. The information is received with the code bellow[6]:

String \_OSVERSION = System.getProperty("os.version");

String \_RELEASE = android.os.Build.VERSION.RELEASE;

String \_DEVICE = android.os.Build.DEVICE;

String \_MODEL = android.os.Build.MODEL;

String \_PRODUCT = android.os.Build.PRODUCT;

String \_BRAND = android.os.Build.BRAND;

String \_DISPLAY = android.os.Build.DISPLAY;

String \_CPU\_ABI = android.os.Build.CPU\_ABI;

String \_CPU\_ABI2 = android.os.Build.CPU\_ABI2;

String \_UNKNOWN = android.os.Build.UNKNOWN;

String \_HARDWARE = android.os.Build.HARDWARE;

String \_ID = android.os.Build.ID;

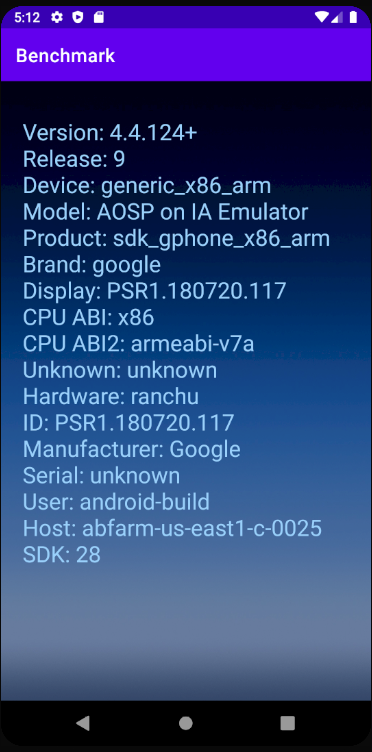
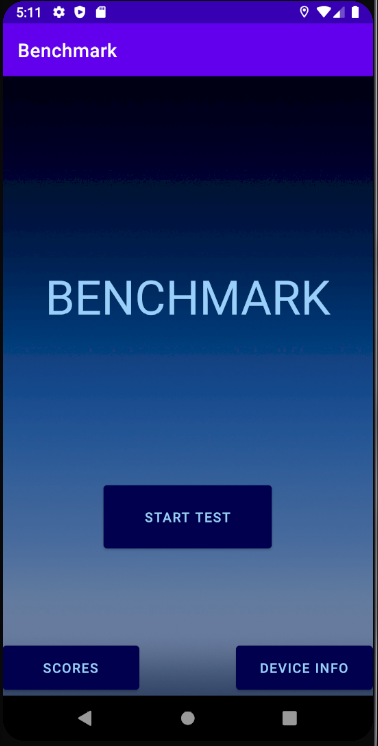
String \_MANUFACTURER = android.os.Build.MANUFACTURER;

String \_SERIAL = android.os.Build.SERIAL;

String \_USER = android.os.Build.USER;

String \_HOST = android.os.Build.HOST;

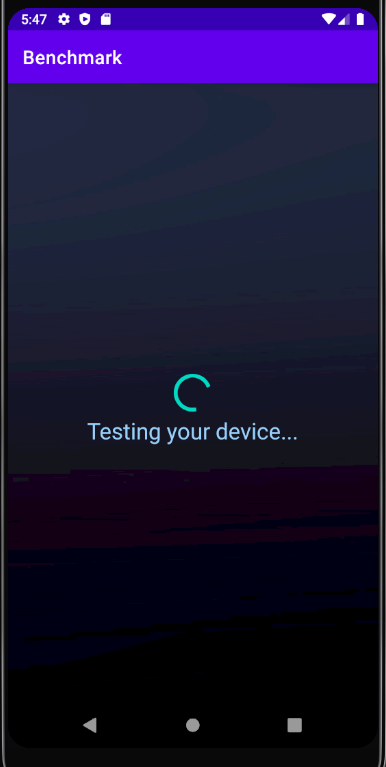
String \_SDK = android.os.Build.VERSION.SDK\_INT;



The button “SCORES” will start a new activity in which will be displayed all the past tests made before. These results are displayed in a ListView populated from a database. I used the sqlite package to create a database in which to store the execution times and the scores of each run. In order to do this I created a model for my items called Test and a class DatabaseRepository which extends SQLiteHelper and helps me access the database. The class Test has four private fields: date, cpuScore, memScore and totalScore together with getters and setters for each one of them.

In the DatabaseRepository class I made the connection to the database in the constructor, specifying the context and the name of the database “Benchmark.db”. With the method onCreate I created the table containing the needed fields for the project which are the same as in the Test model. The method insertTest takes a Test as a parameter and fills the fields of the database with the values contained in the object. The method getAllTests uses a query to take all the elements from the table and form a list of Test objects in order to pass it easily to our app and work with the data in a simple manner.

The most important part of the application is the working behind “START TEST” since it does the main functionality of the project. Once pressed, it starts the LoadingScreen activity which opens a splash screen and runs in background the program.



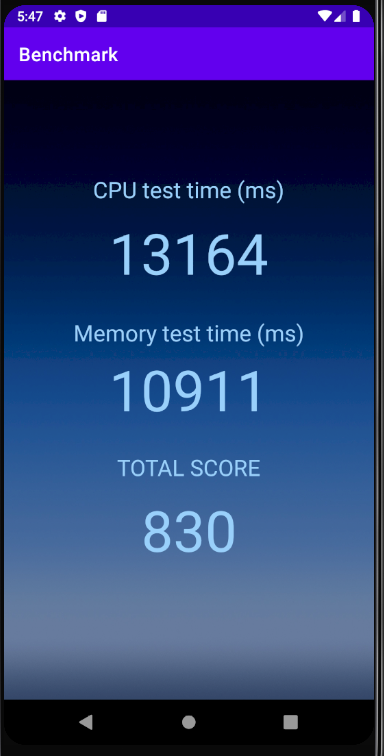
Each type of testing is done on a separate thread, thus verifying the device’s capability of multi-threading. For this I created a Collector class which has the fields “finish”, “scoreCpu” and “scoreMem” all of them being AtomicIntegers. The finish field will help us stop the loading screen one all the threads are done with the computation, and the other ones help us store the results. We have two classes: CPUThread and MemThread both implementing the Runnable interface and having a Collector object as argument for the constructor. The run() method from CPUThread calls the cpuScore() method and gets the execution time. After this, the result is added to the correspondent field in the collector and the finish field is incremented with 1. In MemThread the run() method calls the memScore() method and gets the execution time. In the same manner as the CPUThread, it stores the result and increments the finish field with 1.

These threads are started in the doInBackground method from Calculate and the parent thread needs to wait until the finish field from the collector equals the number of threads. Then all the data from the collector is stored in resultArray and passed to onPostExecute method, a method which tells the program what to do after the hard computation finishes. It creates a new activity ScoreScreen and also sends the results calculated before.

In the ScoreScreen activity, all the application does is to take the sent results with the help of a bundle and setting the labels with the respective values. Also, it creates a Test object with these values and inserts the element in the database table.

The total score is computed in the following way:

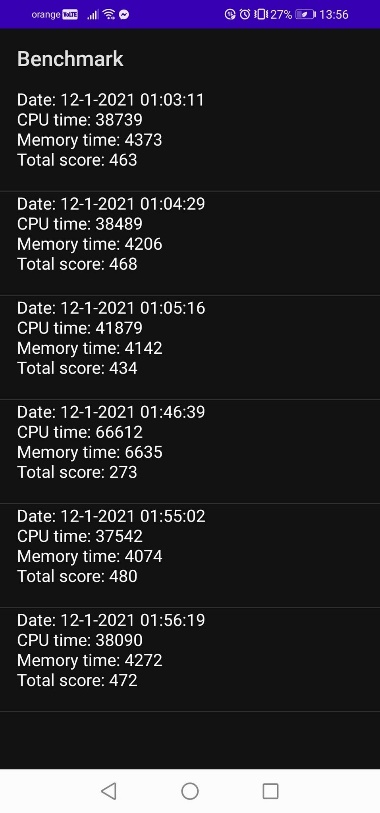
total score = 10000000 / ((CPU time + Memory time)/2)



1. **Testing**

In order to observe the results of the application, its working and the interaction with other devices I managed to test the application on several different mobile phones.

1. Huawei P30 Lite



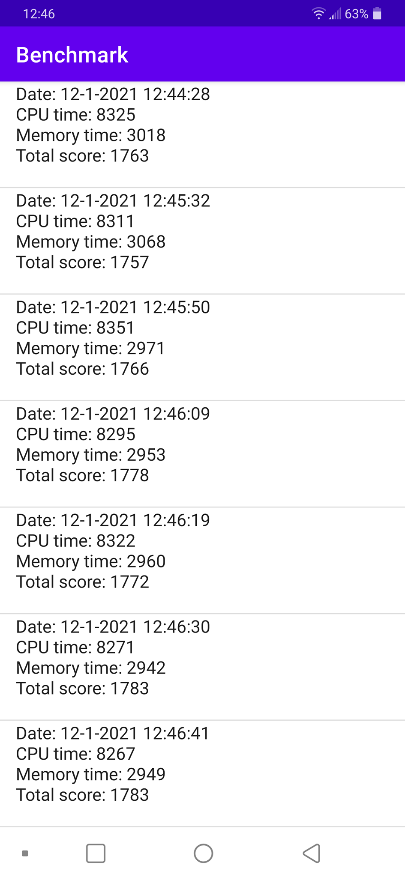


1. Samsung Galaxy S8



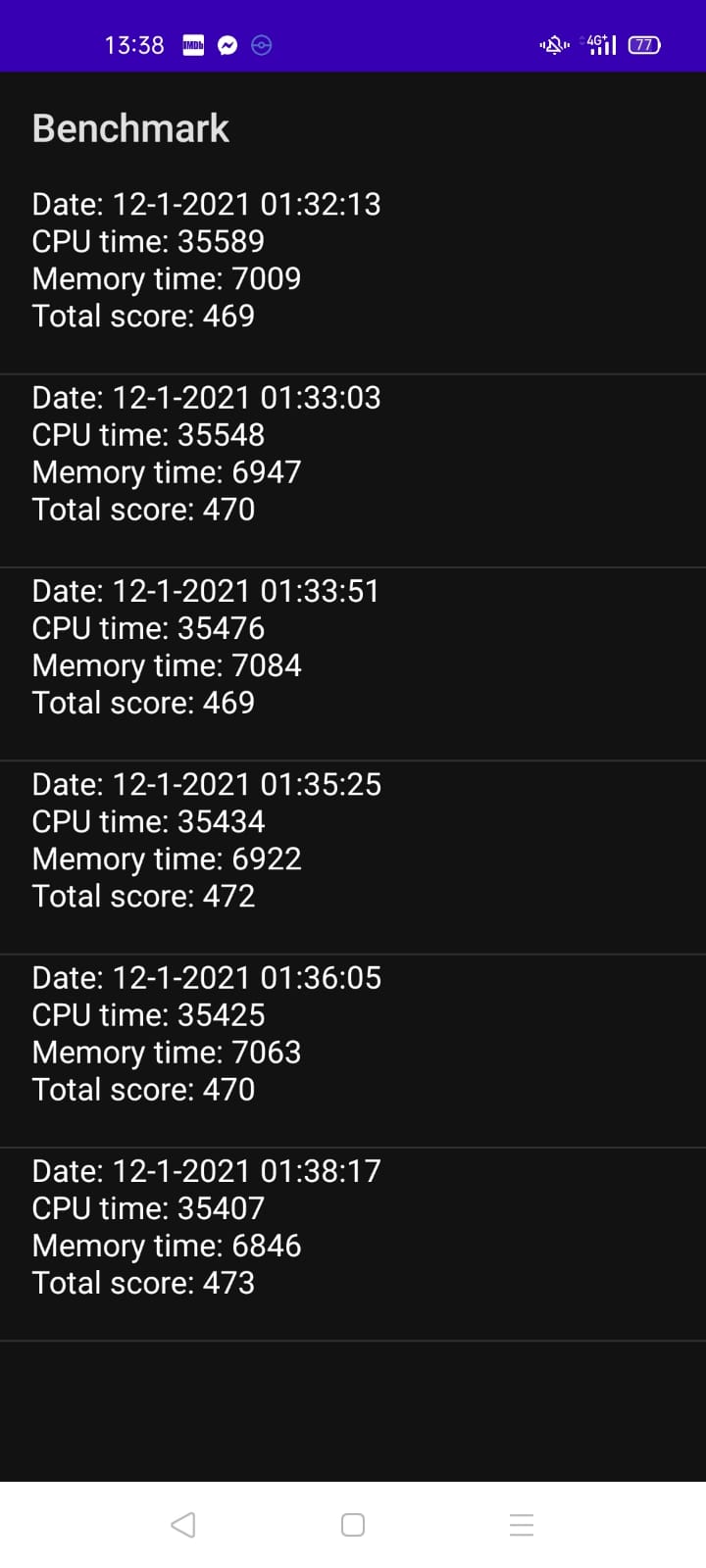


1. LG G7 ThinQ





1. Oppo A72





1. Samsung A50





For each device I ran the app multiple times and analyzed the behavior. By checking the score view, we can observe that the values for one device are almost the same thus being consistent in its calculations.

Moreover, I wanted to verify if the score reflects indeed the true capability of the devices compared to one another so there is a table of the data acquired about the mobile phones.



By looking at the table with average total scores for each device we can see that the numbers are in the same order as the scores found on Geekbench[8], so it reflects the true performance of the devices (order wise).

1. **Conclusions**

This project represented an interesting and challenging idea of implementation, by the end of which I obtained a functional and fairly accurate benchmark application for mobile devices which could be used on Android OS. The project helped me better understand the principles behind a benchmark and its functionality and also, I improved my knowledge regarding hardware components by studying the benchmark and software knowledge since I learned to better use Android studio.

For further development, the application could also have a branch for checking the GPU performance and some detailed information about the made measurements. Furthermore, the “previous scores” page could also have a “clear” functionality or maybe comparing the personal results with the ones from an online database which contains the scores of other devices.

[1] - Fleming, Philip J.; Wallace, John J. (1986-03-01). "How not to lie with statistics: the correct way to summarize benchmark results". Communications of the ACM.

[2] - https://lifehacker.com/how-benchmarks-work-and-when-you-should-pay-attention-t-1792579167

[3] - Dai, Wei; Berleant, Daniel (December 12–14, 2019). "Benchmarking Contemporary Deep Learning Hardware and Frameworks: a Survey of Qualitative Metrics" (PDF). 2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI). Los Angeles, CA, USA: IEEE. pp. 148–15

[4] - https://www.brainbridge.be/news/6-free-apps-to-test-your-smartphones-speed

[5] - Kashif Hussain#, Mohd Najib Mohd Salleh#, Shi Cheng\*, Rashid Naseem\*\*, Common Benchmark Functions for Metaheuristic Evaluation: A Review, INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

[6] - https://developer.android.com/reference/android/os/Build.html

[7] - https://www.baeldung.com/java-random-string

[8] - https://browser.geekbench.com/mobile-benchmarks