**A summary report (in the .doc or docx format), which should clearly demonstrate the performance and energy consumption of each version of your code (you can show it in a table). In addition, you should clearly state the details of optimizations you have made to improve the energy efficiency of your code.**

This part discusses about different optimization techniques that are applied in three versions of code to make it more efficient. One thing to mention here is that one of the most straightforward ways to make an efficient python code is to use the latest version of it. Because of some limitations of Green Code compiler, we had to use an on older version of python.

**First Version**

***Use of Proper Data Structure:***

Proper data structure has a significant impact on the runtime. Python has list, tuple and dictionary for built in data structures. Here in this code, list comprehension is used instead of regular list to create a separate integer list which contains only the integer values from the input.

Example from code:

*for i in range(1,len(int1\_list)):*

***Use of Local Variable:***

Python has a global keyword to declare global variable. But global variable takes higher time during operation than a local variable. To speed up the code , no map version of the for loop is used as local variable. In every for loop, the use of append and casting it as a function, makes the variable as local one. Python accesses local variable much more efficiently than global variable.

Example from code:

*line\_no.append(int(int1\_list[i][0]))*

*int1\_list.append(list.split())*

***Concatenate strings with join:***

Join function concatenate strings faster than operation. Because operator create a new string and then copies the old content at each step. But join() doesn’t work that way. In this code, for character encoding from the input the join method is used.

Example from code:

*joint=unicode(joint, 'utf-8')*

***No library is called with dot operation:***

When a function is called using . (dot) it first calls \_getattribute()\_ or \_getattr\_ which then use dictionary operation costing time. In this code, calling library with dot operation is avoided.

***Initializing Dictionary Element:***

Building a dictionary to access the word along with the frequency is more efficient than using other data structures. The code is structured also using set. But the total run time gets affected due to using sets and the output gets a compile time error.

Example from the code:

*for word in words:*

*# Check if the word is already in dictionary*

*if word in d:*

*# Increment count of word by 1*

*d[word] = d[word] + 1*

*else:*

*# Add the word to dictionary with count 1*

*d[word] = 1*

***Import Statement Overhead:***

Import statements can be executed anywhere. It’s useful to place them inside function to restrict their visibility and reduce initial startup time, In this code, the import statements are called inside function.

*for line in sys.stdin:*

*import re*

*no\_specials\_string = re.sub('[!#?,.:";]', '', line)*

***Mapping Function:***

Python supports a couple of looping constructs. The for statement is commonly used. The interpreter overhead of the for loop itself consumes a substantial amount of the overhead, there, map function is handy. The map function is used to split the words.

*for i in range(0,len(test\_cases)):*

*x=test\_cases[i]*

*x = ' '.join(map(str, x))*

*lines= x.strip()*

*words = x.split(" ")*

Power consumption of Version 1

Chart, pie chart

Description automatically generated

Figure 1 : Breakdown of Power of Consumption of first version code

**Second Version:**

***Sorting words in descending order:***

Sorting list of basic python objects is efficient. The sort method takes a function as an argument to change the sorting behavior. The key argument is a built in sort which is the fastest way to sort.

The difference between the first and second version of the python code is to call the sorting algorithm in a function manner. In this version, the sorting of words in descending order and return the result in done in a function. And the result is called inside the final for loop which prints the test cases.

Example from code:

*def Convert(f):*

*s= sorted(f.items(), key=lambda kv: kv[::-1], reverse=True)*

*result = []*

*for k, v in itertools.groupby(s, lambda item: item[1]):*

*result.extend(sorted(v))*

*return result*

Chart, pie chart

Description automatically generated

Figure 2: Breakdown of Power Consumption of second version code (first version is considered as base)

**Third Version:**

The third version lessens an extra for loop is deciding up to which position most frequent word should be printed for that input. Instead of using enumerate which consumes two for loops , this version of code is optimized with one for loop using append function.

Chart, table, pie chart

Description automatically generated

**Power Consumption comparison (in joules) between three versions of code:**

|  |  |  |
| --- | --- | --- |
| Version No | Program Execution | Greenup |
| Version 1 | 116 | 1 |
| Version 2 | 110 | 1.01 |
| Version 3 | 109 | 1.02 |

Among them, version 3 is the most efficient one.