# Best datatype structure for the input data

## I. Introduction

In order to boost the program, we want to use a datatype structure that is light and robust, in order to contains the model's inputs (USMS). What will follow in this document are the tests done in the purpose of determining the best datatype structure suited for that. The tests will concern the generation speed, the memory weight and the manipulation ease.

### I.A The Dataypes structures

Initially, they were 6 candidates :

- The List type
- The Reference Class type
- The DataFrame type
- The DataTable type
- The Tibble type
- The Matrix type

However, the Matrix type was the first one to be put away because it can only contains informations with homogenous types (e.g only numeric or character but not numeric and character together). Or, an USM is composed of :

- a name (character)
- a beginning date (numeric)
- an ending date (numeric)
- an initialisation file name (character)
- the ground name (character)
- a station file name (character)
- a first climate file name (character)
- a second climate file name (character)
- the year of culture (numeric)
- the number of plants (numeric)
- the simulation_code (numeric)
- informations about the first plant (3 informations(fplt,ftec,flai)) (character)
- informations about the second plant (same)
- the observation file name (character)

So here are examples of usm in the remaining datatypes.

#### I.A.1 List

```
example_usm_list <- usm_list("usm_1","sol_canne")
example_usm_list
```

```
## $name
## [1] "usm_1"
##
## $date_begin
## [1] 256
##
## $date_end
## [1] 384
##
## $finit
## [1] "canne_ini.xml"
##
## $ground_name
## [1] "sol_canne"
##
## $fstation
## [1] "climcanj_sta.xml"
##
## $fclim1
## [1] "climcanj.1998"
##
## $fclim2
## [1] "climcanj.1999"
##
## $culturean
## [1] 0
##
## $nb_plant
## [1] 1
##
## $simulation_code
## [1] 0
##
## $plant_dominance_1_fplt
## [1] "proto_sugarcane_plt.xml"
##
## $plant_dominance_1_ftec
## [1] "canne_tec.xml"
##
## $plant_dominance_1_flai
## [1] "null"
##
## $plant_dominance_2_fplt
## [1] "null"
##
## $plant_dominance_2_ftec
## [1] "null"
##
## $plant_dominance_2_flai
## [1] "null"
```

```
##
## $observation_file
## [1] "file.obs"
```

### I.A.2 Class

```
example_usm_class <- usm_class("usm_1","sol_canne")
example_usm_class
```

```
## Reference class object of class "usm"
## Field "name":
## [1] "usm_1"
## Field "date_begin":
## [1] 256
## Field "date_end":
## [1] 384
## Field "finit":
## [1] "canne_ini.xml"
## Field "ground_name":
## [1] "sol_canne"
## Field "fstation":
## [1] "climcanj_sta.xml"
## Field "fclim1":
## [1] "climcanj.1998"
## Field "fclim2":
## [1] "clilmcanj.1999"
## Field "culturean":
## [1] 0
## Field "nb_plant":
## [1] 0
## Field "simulation_code":
## [1] 1
## Field "plant_dominance_1_fplt":
## [1] "proto_sugarcane_plt.xml"
## Field "plant_dominance_1_ftec":
## [1] "canne_tec.xml"
## Field "plant_dominance_1_flai":
## [1] "null"
## Field "plant_dominance_2_fplt":
## [1] "null"
## Field "plant_dominance_2_ftec":
## [1] "null"
## Field "plant_dominance_2_flai":
## [1] "null"
## Field "observation_file":
## [1] "file.obs"
```

### I.A.3 DataFrame

```
example_usm_dataframe <- usm_dataframe("usm_1","sol_canne")
example_usm_dataframe
```

```
##    name date_begin date_end         finit ground_name         fstation
## 1 usm_1        286      384 canne_ini.xml   sol_canne climcanj_sta.xml
##          fclim1        fclim2 culturean nb_plant simulation_code
## 1 climcanj.1998 climcanj.1999         0        1               0
##    plant_dominance_1_fplt plant_dominance_1_ftec plant_dominance_1_flai
## 1 proto_sugarcane_plt.xml          canne_tec.xml                   null
##   plant_dominance_2_fplt plant_dominance_2_ftec plant_dominance_2_flai
## 1                   null                   null                   null
##   observation_file
## 1         file.obs
```

### I.A.4 DataTable

```
example_usm_datatable <- usm_datatable("usm_1","sol_canne")
example_usm_datatable
```

```
##     name date_begin date_end         finit ground_name         fstation
## 1: usm_1        286      384 canne_ini.xml   sol_canne climcanj_sta.xml
##           fclim1        fclim2 culturean nb_plant simulation_code
## 1: climcanj.1998 climcanj.1999         0        1               0
##     plant_dominance_1_fplt plant_dominance_1_ftec plant_dominance_1_flai
## 1: proto_sugarcane_plt.xml          canne_tec.xml                   null
##    plant_dominance_2_fplt plant_dominance_2_ftec plant_dominance_2_flai
## 1:                   null                   null                   null
##    observation_file
## 1:         file.obs
```

### I.A.5 Tibble

```
example_usm_tibble <- usm_tibble("usm_1","sol_canne")
example_usm_tibble
```

```
## # A tibble: 1 x 18
##   name  date_begin date_end finit ground_name fstation fclim1 fclim2 culturean
##   <chr>      <dbl>    <dbl> <chr> <chr>       <chr>    <chr>  <chr>      <dbl>
## 1 usm_1        286      384 cann~ sol_canne   climcan~ climc~ climc~         0
## # ... with 9 more variables: nb_plant <dbl>, simulation_code <dbl>,
## #   plant_dominance_1_fplt <chr>, plant_dominance_1_ftec <chr>,
## #   plant_dominance_1_flai <chr>, plant_dominance_2_fplt <chr>,
## #   plant_dominance_2_ftec <chr>, plant_dominance_2_flai <chr>,
## #   observation_file <chr>
```

# II. Tests

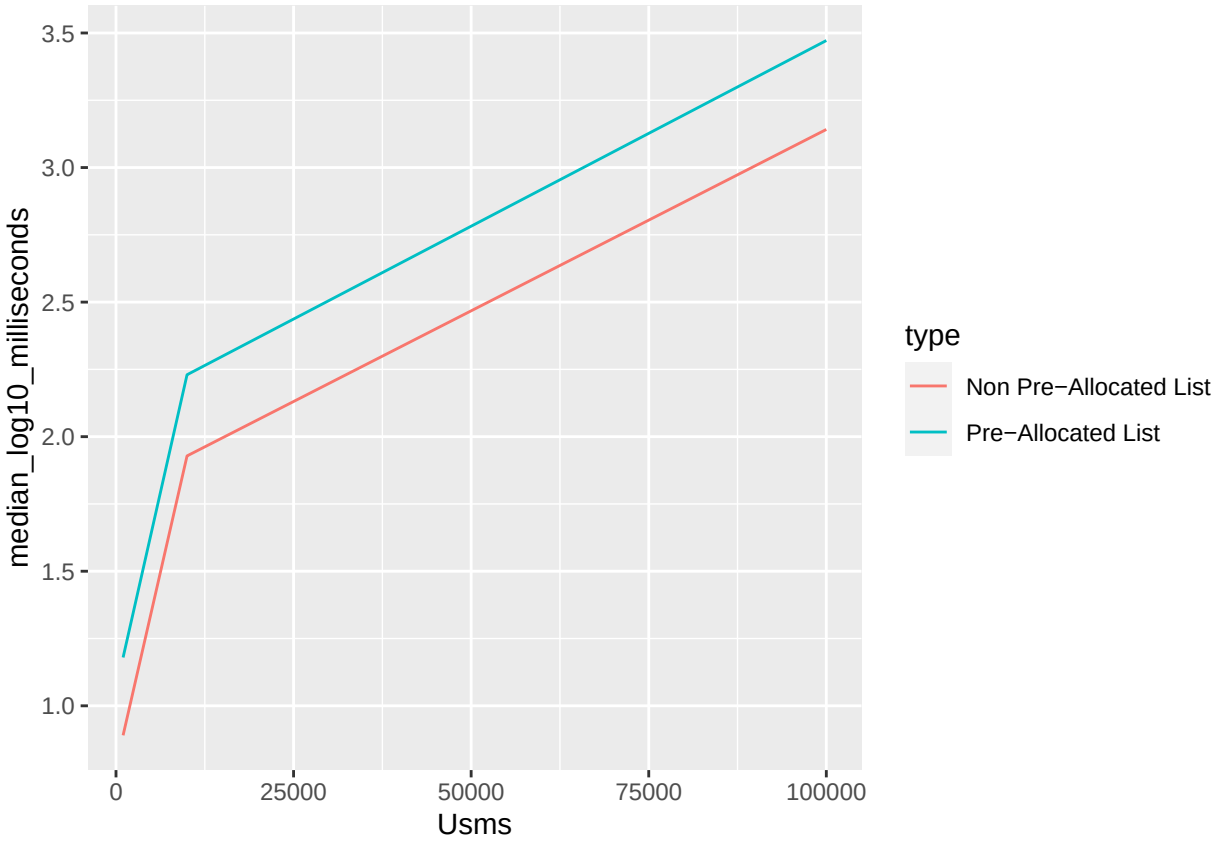
The firsts tests to be made were instanciation tests. We tested the instanciation time and weight for each datatype structure with different instanciation method and for differents sizes (10 000, 100 000, 1 000 000)

## II.A Instanciation Tests

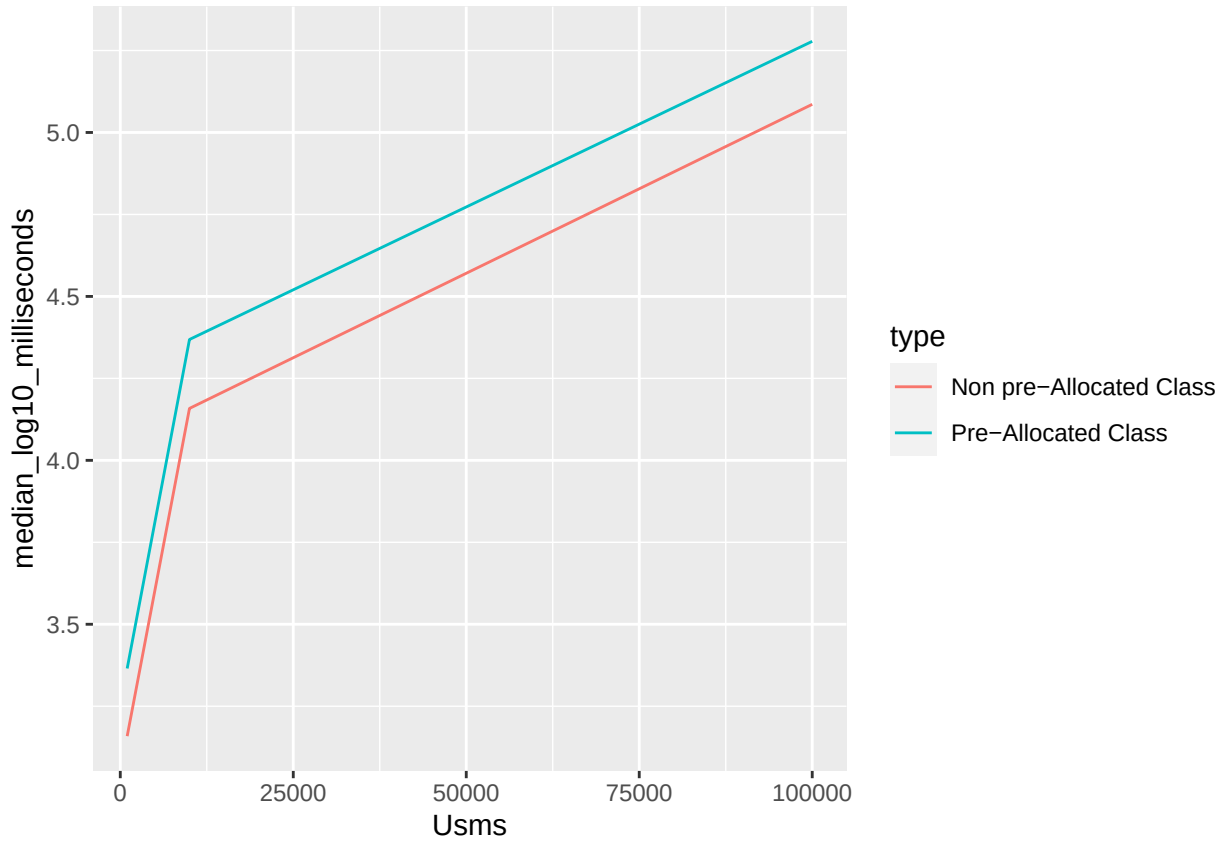
### II.A.1 List Vs Class

These tests were made for list and class types because only these two types can be instanciated with a non pre-allocated method, which is not possible for the other types.

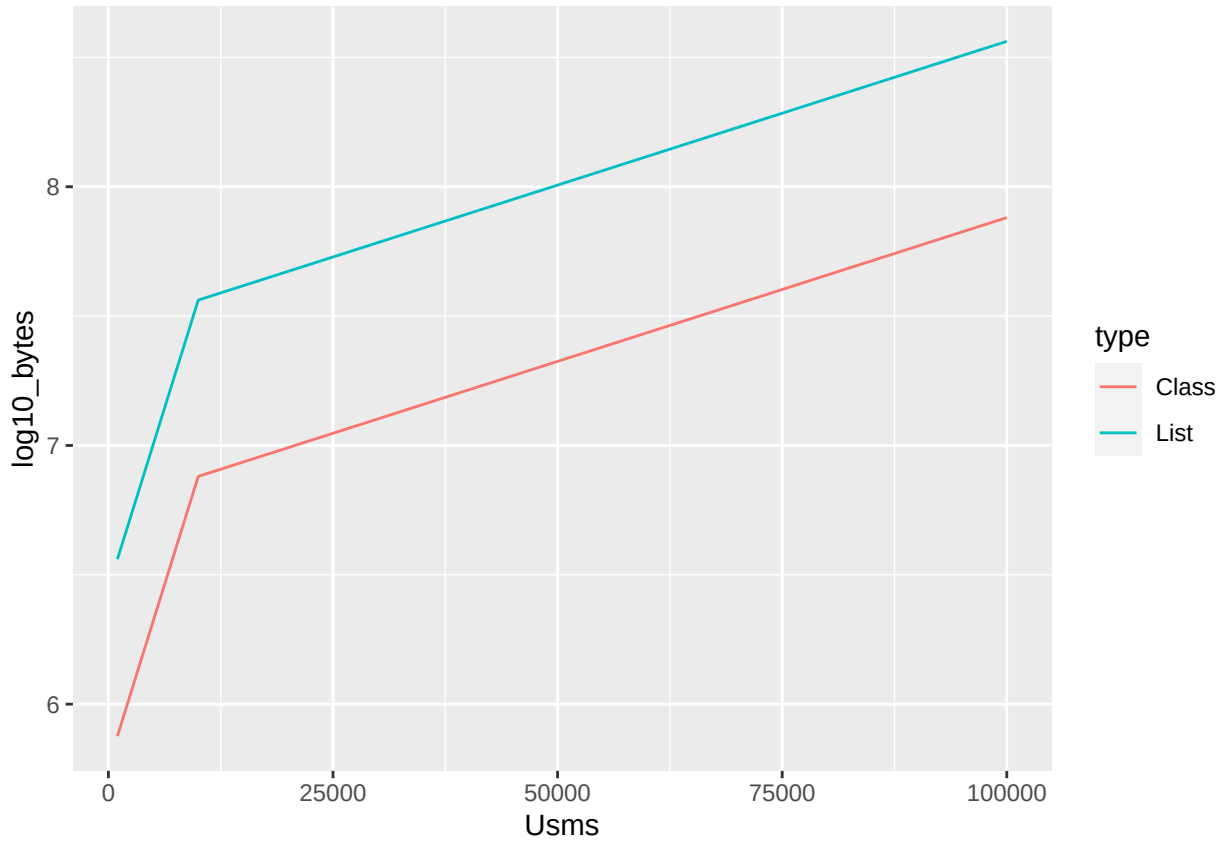
```
## List time graph for 1K Usms to 100K Usms
```



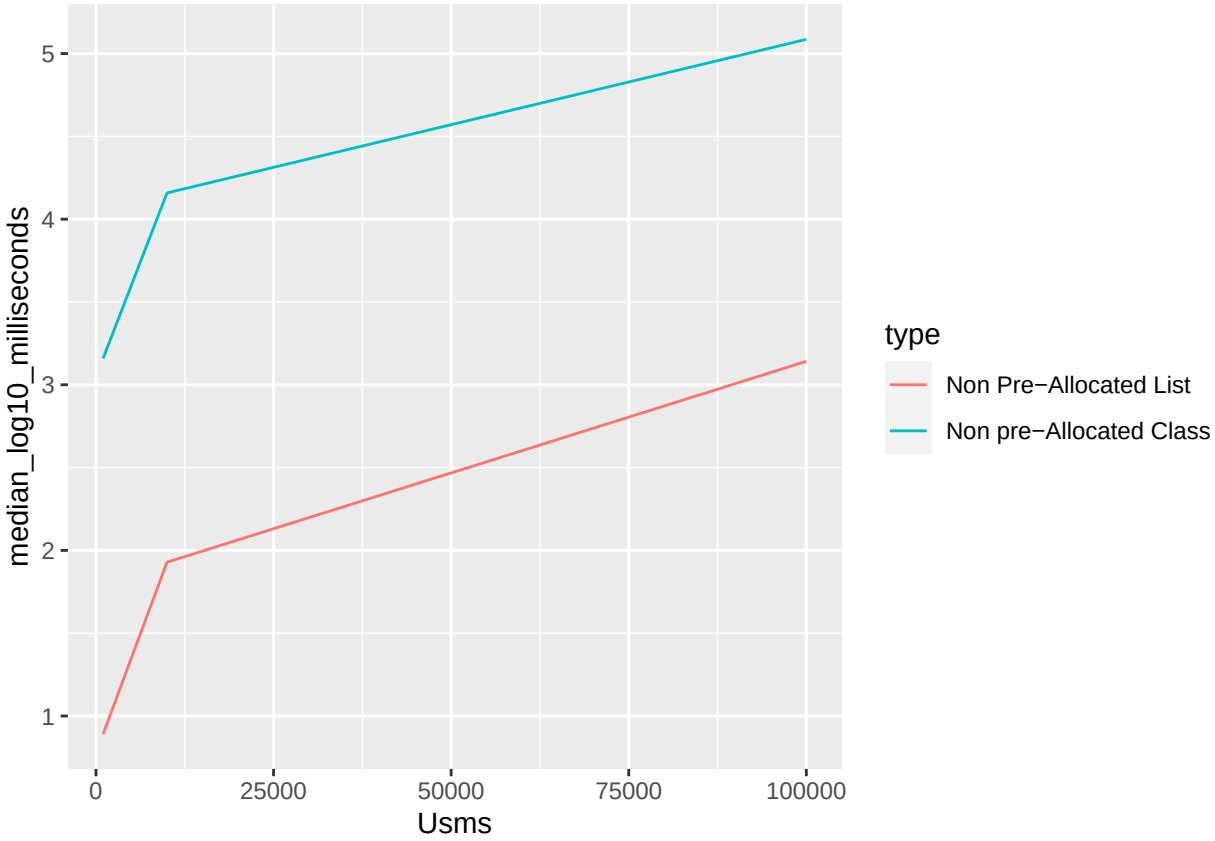

```
## Class time graph for 1K Usms to 100K Usms
```




```
## Weights graph between List and Class
```




```
## Comparative time graph for fastest instanciation method for List and Class
```

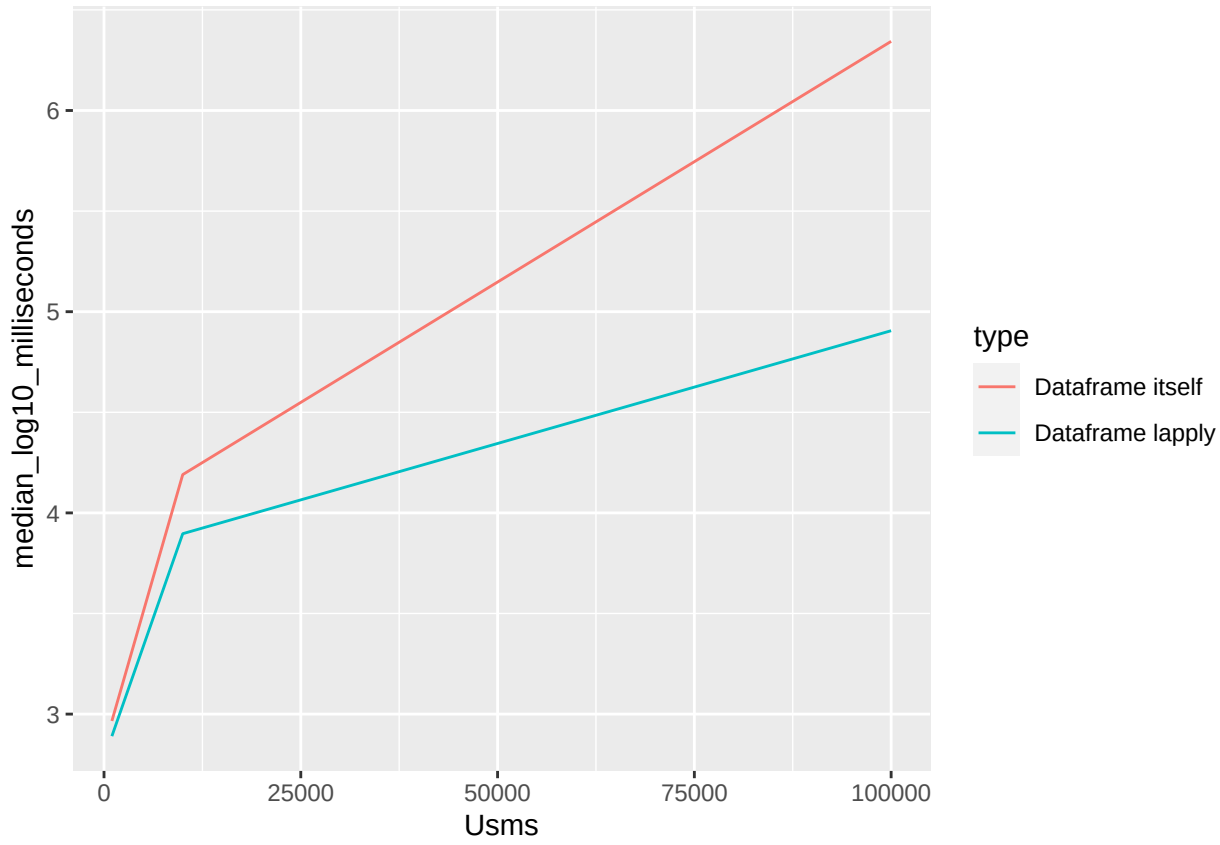



First of all, the NPA version is faster for both types. Furthermore, the Class type is lighter than the List type, but we can see that the difference remains stable through the Usms augmentation. The reason why the Class type is lighter than the List type can be explain by the fact that the Class type stores only the addresses of its variables and not the variables themselves. Finally, we can see that the List type has an instanciation time faster than the Class type.

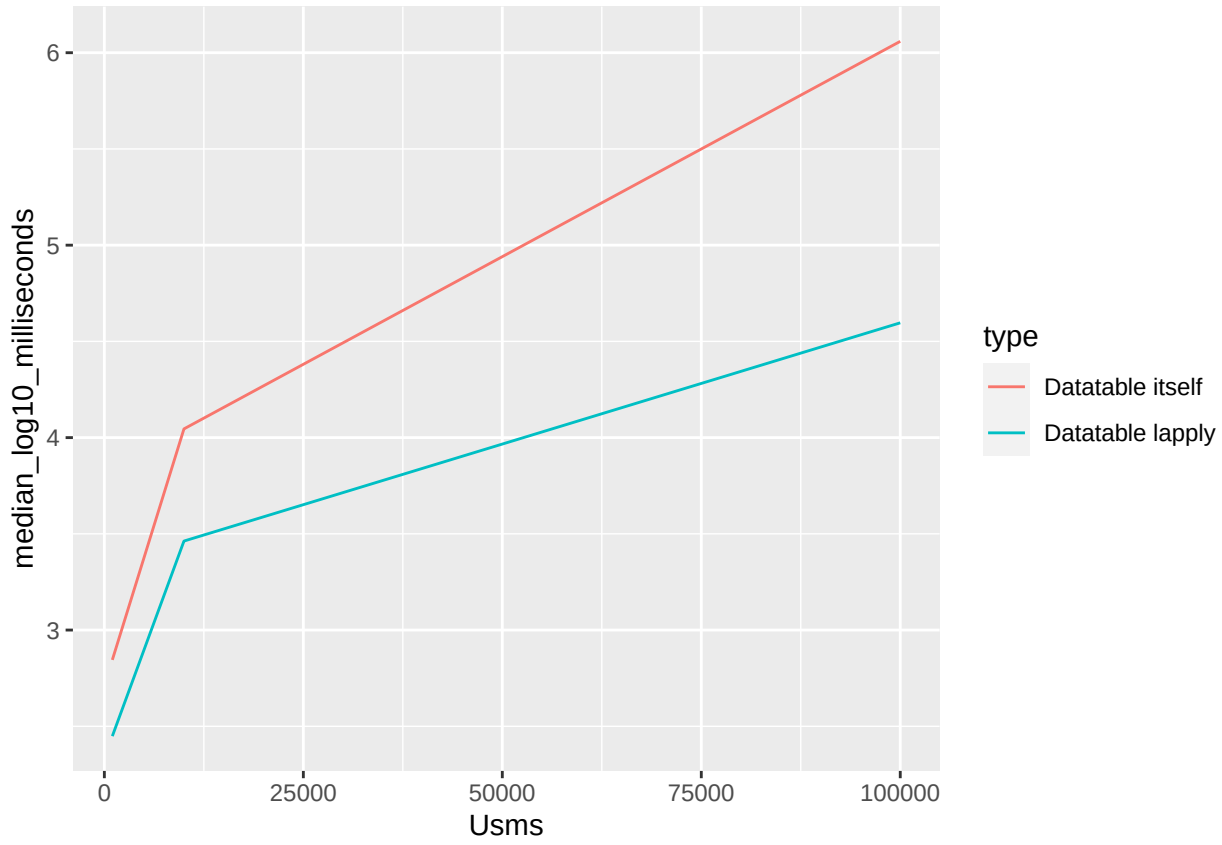
### II.A.2 DataFrame, DataTable and Tibble

Now, let's move on to the tests involving the DataFrame,DataTable and Tibble instanciation methods. Indeed, they are severals ways to instanciate each of these types. DataFrame : By affecting list in a dataframe's row or by row binding a list of dataframe. DataTable : Doing the same as before but for a DataTable or by casting the DataFrame output into a DataTable. Tibble : Same ways as for DataTable

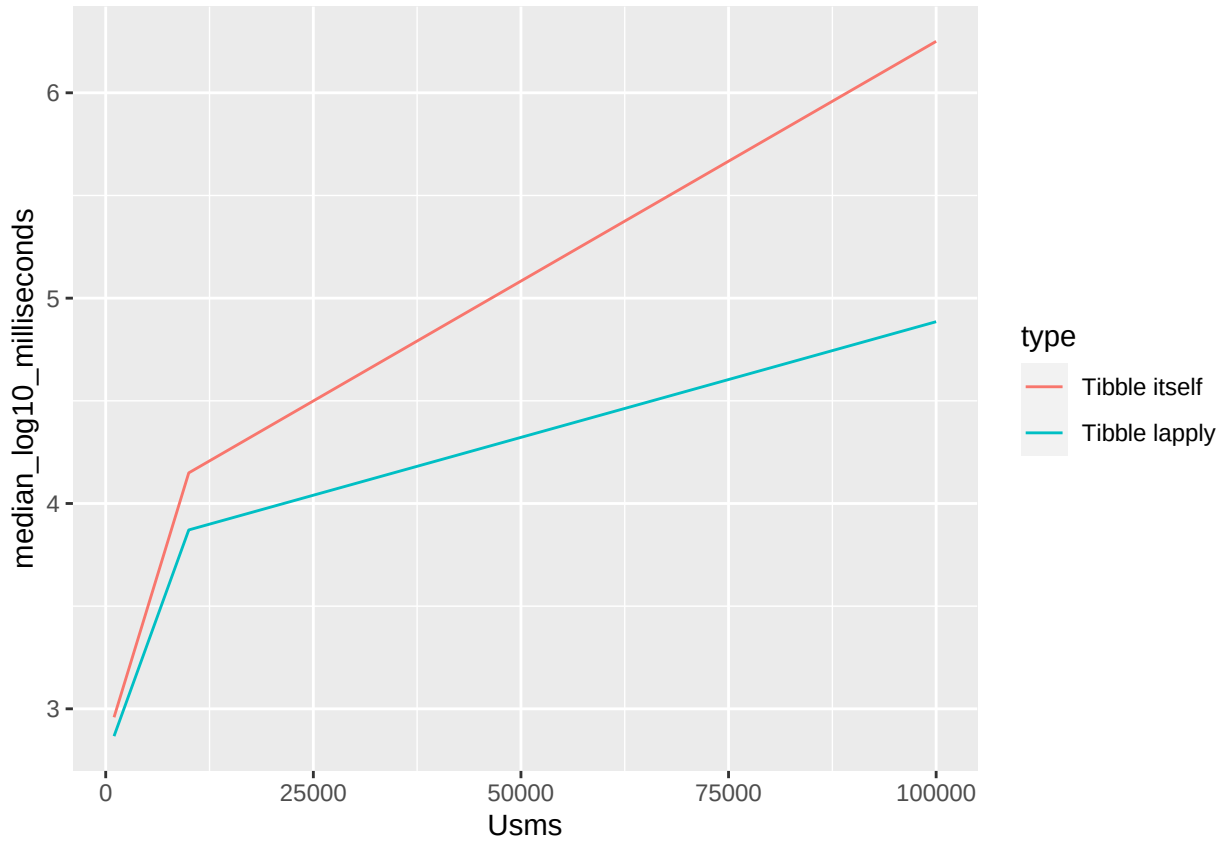
```
## Time graph for Dataframe for 1K USms to 100K USms
```



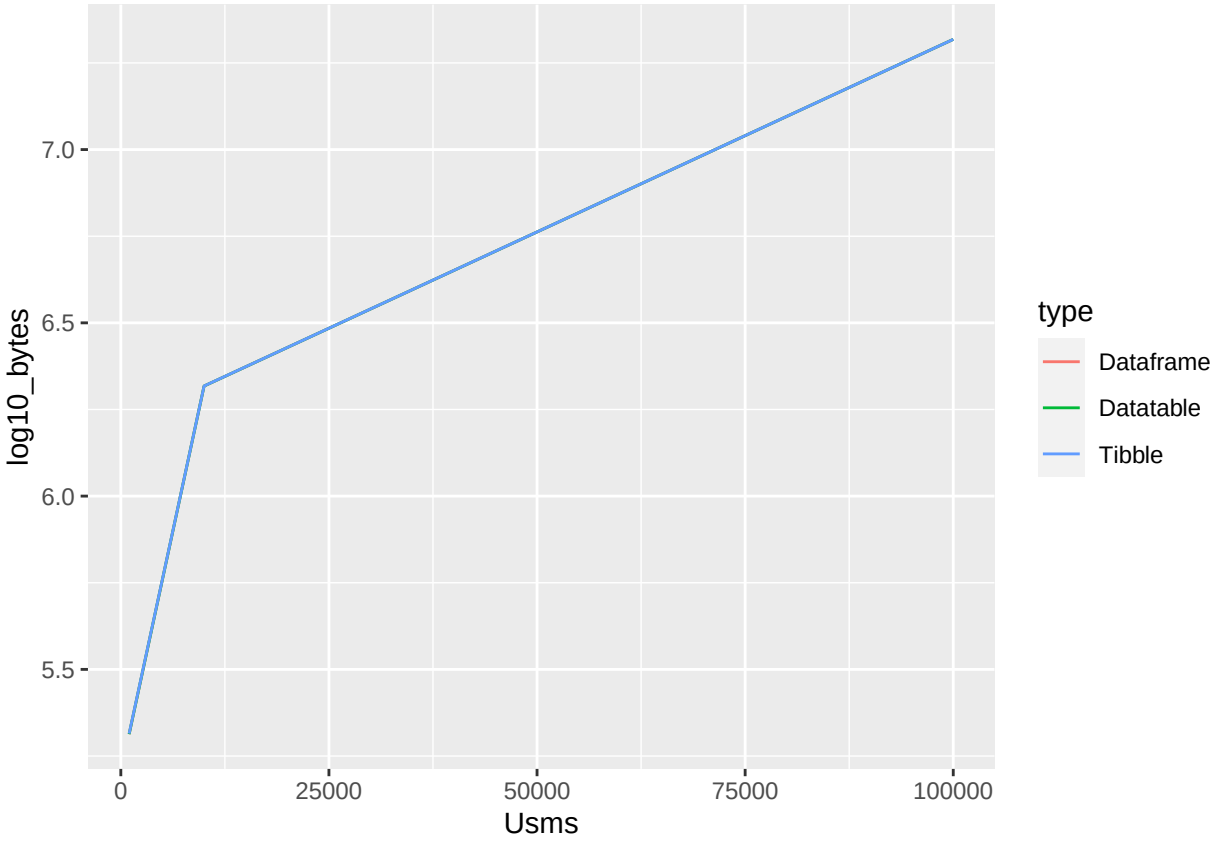

```
## Time graph for Datatable for 1K USms to 100K USms
```



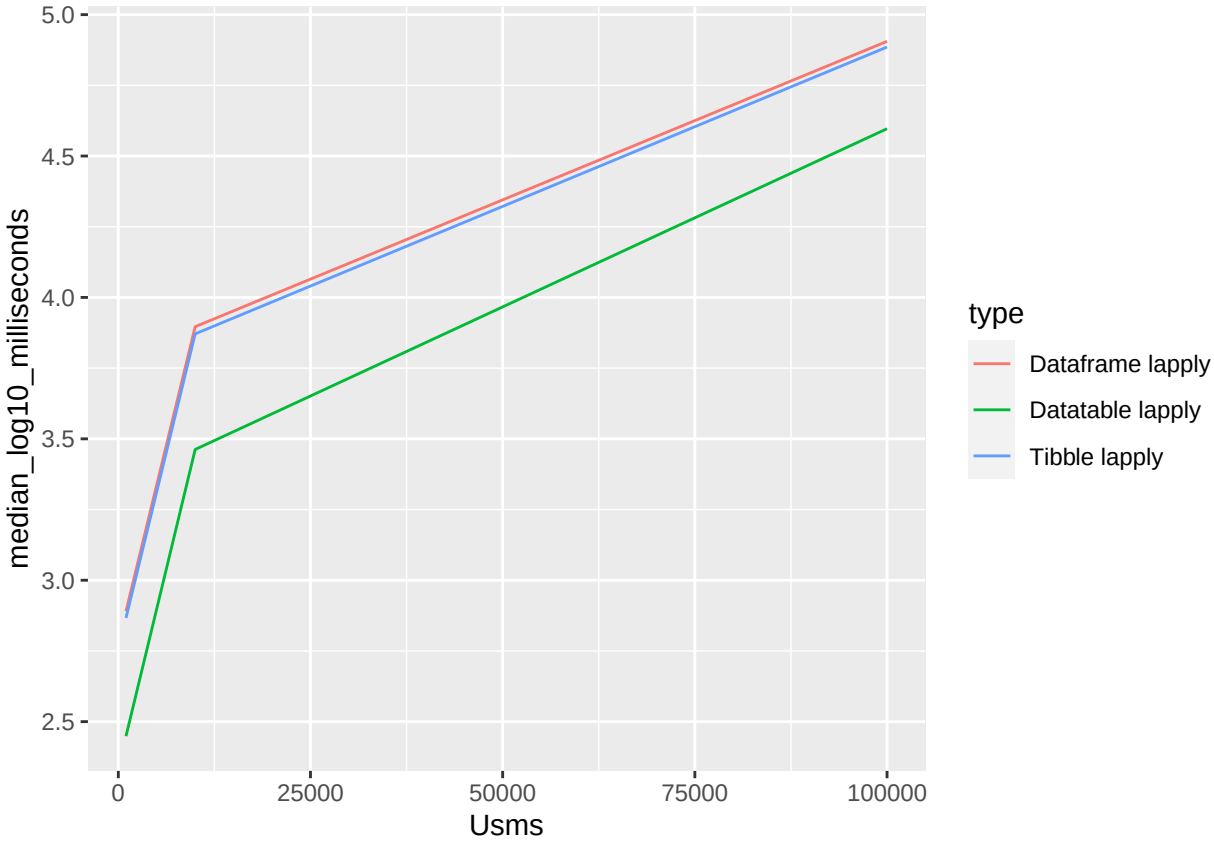

```
## Time graph for Tibble for 1K USms to 100K USms
```




```
## Weights graph betweend Dataframe, Datatable and Tibble
```




```
## Comparative time graph for the fastest instanciation method for DataFrame, DataTable and Tibble
```




Firstly, for all of them, when the number of Usms contained is inferior to 10K, the instanciation time to create a DataFrame-like structure from scratch is as equivalent as converting a List into a Dataframe-like structure. However, when we exceed 10K Usms, we can see a clear difference in instanciation time between both instanciation methods and the method using the list conversion is far faster. Secondly, we can see that all the structures have the same weights whatever the Usms number they have. Finally, the DataTable type instanciation time seems to be the best but the gap with the others types instanciation times is not so huge.

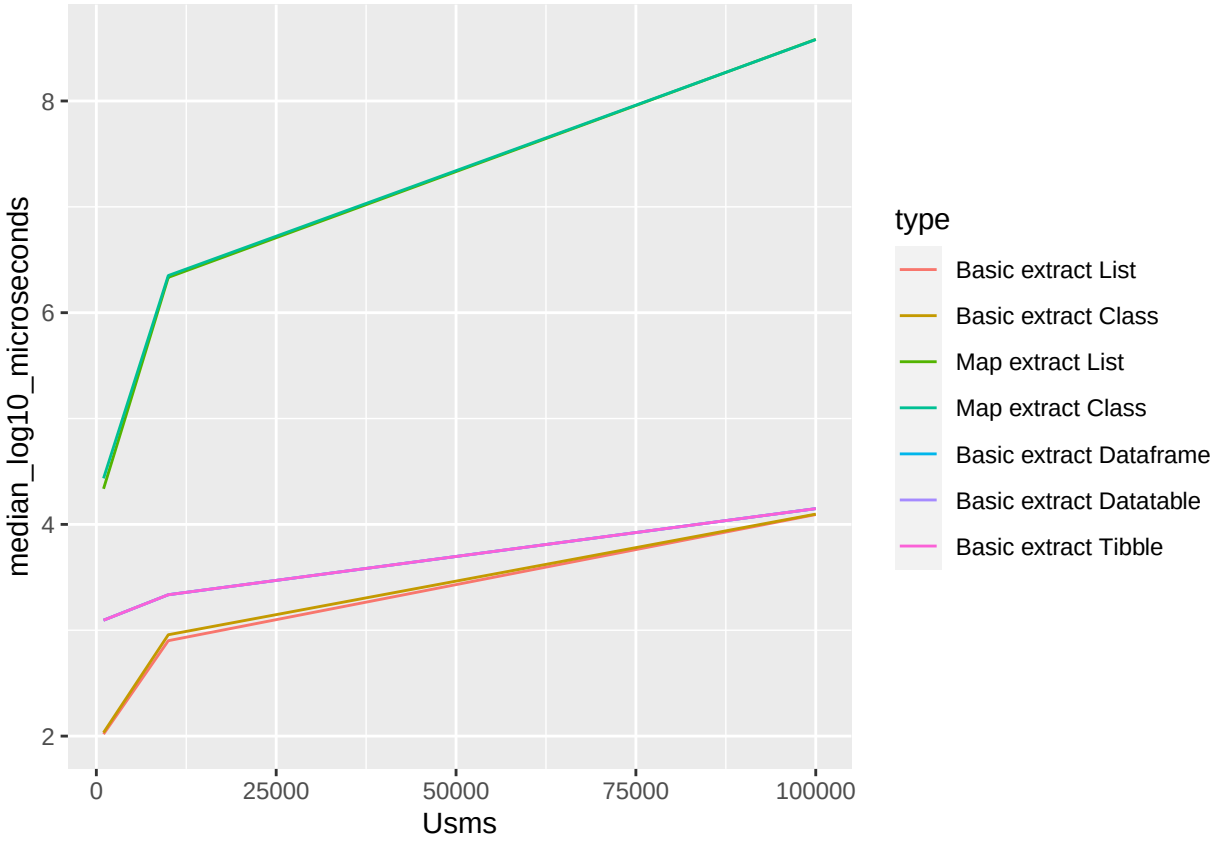
## II.B Extraction

We're now moving to Usm sublist extraction test part. For this part, we tested two methods of extracting usms sublist, by random usm name and criteria research. Each method comports differents ways to be executed depending of the type that is used, for instance, the method used to extract a subfrom a Class would not be the same than for a Dataframe-like structure. Furthermore, inside the List and Class types, two differents methods were tested, the first is a method that uses List and Class types as a dictionnary and the other method is a map method. Finally, for the Dataframe-like structure, the method tested is the dplyr::filter function.

### II.B.1 Random extraction by Usm name

In these tests we're randomly picking and extracting usm names.

```
## Graph time comparison between the differents ways to extract data and the differents structures
```

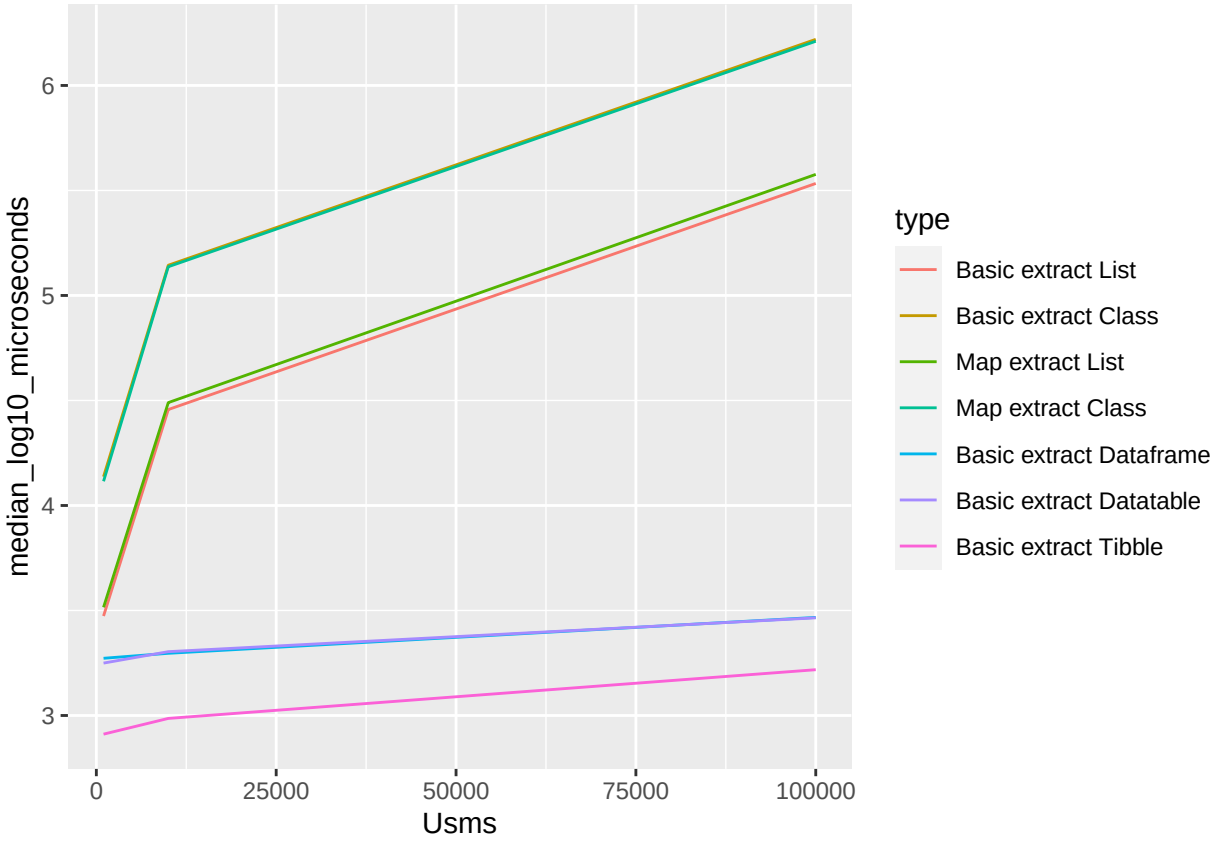



First of all, we can see the the map methods are the slowest methods. Next, we can see that the dlpyr::filter function has the same execution time whatever the Dataframe-like structure used. Then, we can see that the method to extract a sublist from a List type using it as a dictionnary is the fastest of all, slighty before the same method using the Class type. Finally, we see that there's no a huge gap between the dplyr::filter function and the method using List and Class types as dictionnary and the most important thing, the more Usms the structure have, thinner is the gap between these two methods.

### II.B.2 Extraction by criteria

In these tests we're selectionning a usm's criteria and extracting the usm in which the criteria is present.

```
## Graph time comparison between the differents ways to extract data and the differents structures
```

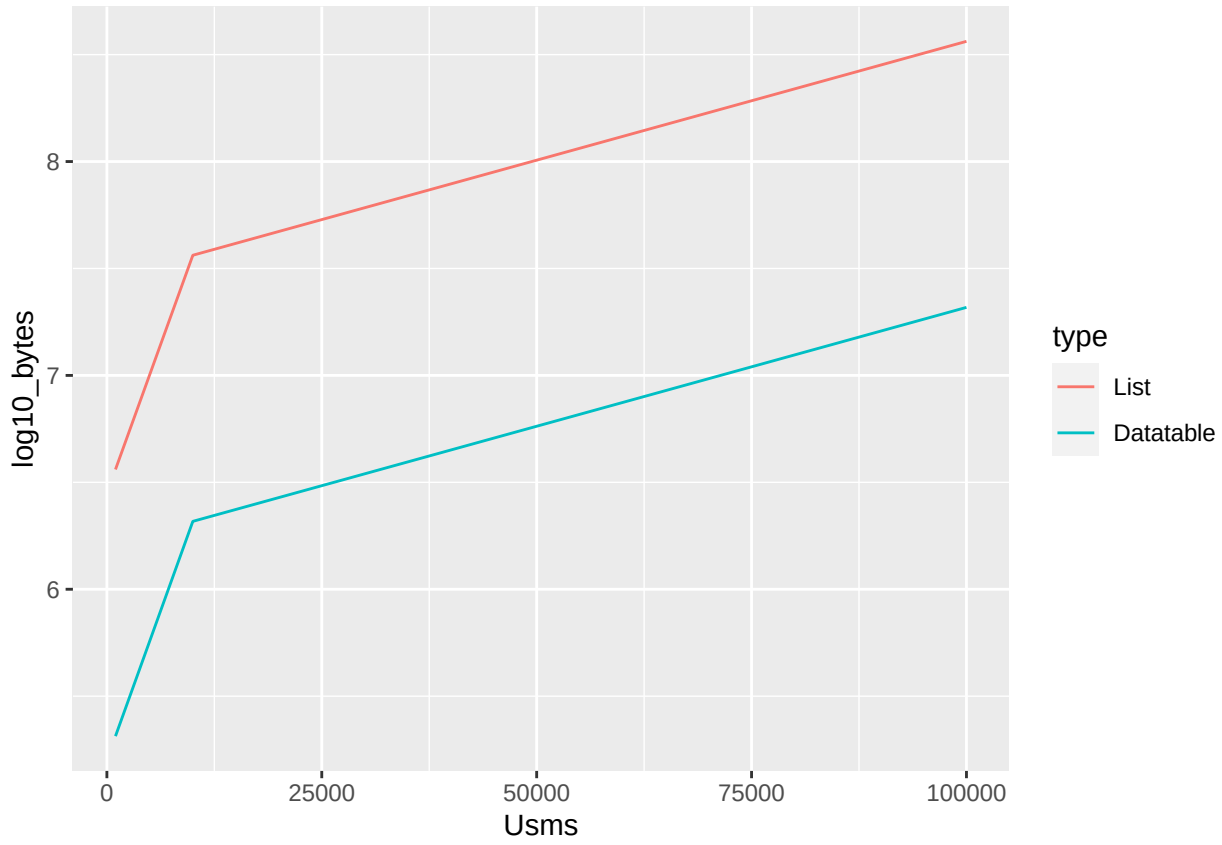



Now we can see some diffrences. First, the methods using the Class type (map method and dictionnary method) are the slowest. Then it is the same methods but using the List type. These are the firsts two diffrences (1 : not grouped by methods but by types, 2: method using List and Class types as dictionnary is not the fastest anymore). Other difference, the dplyr::function has not the same execution time depending the type used, the faster is the Tibble type then the DataFrame and DataTable types.

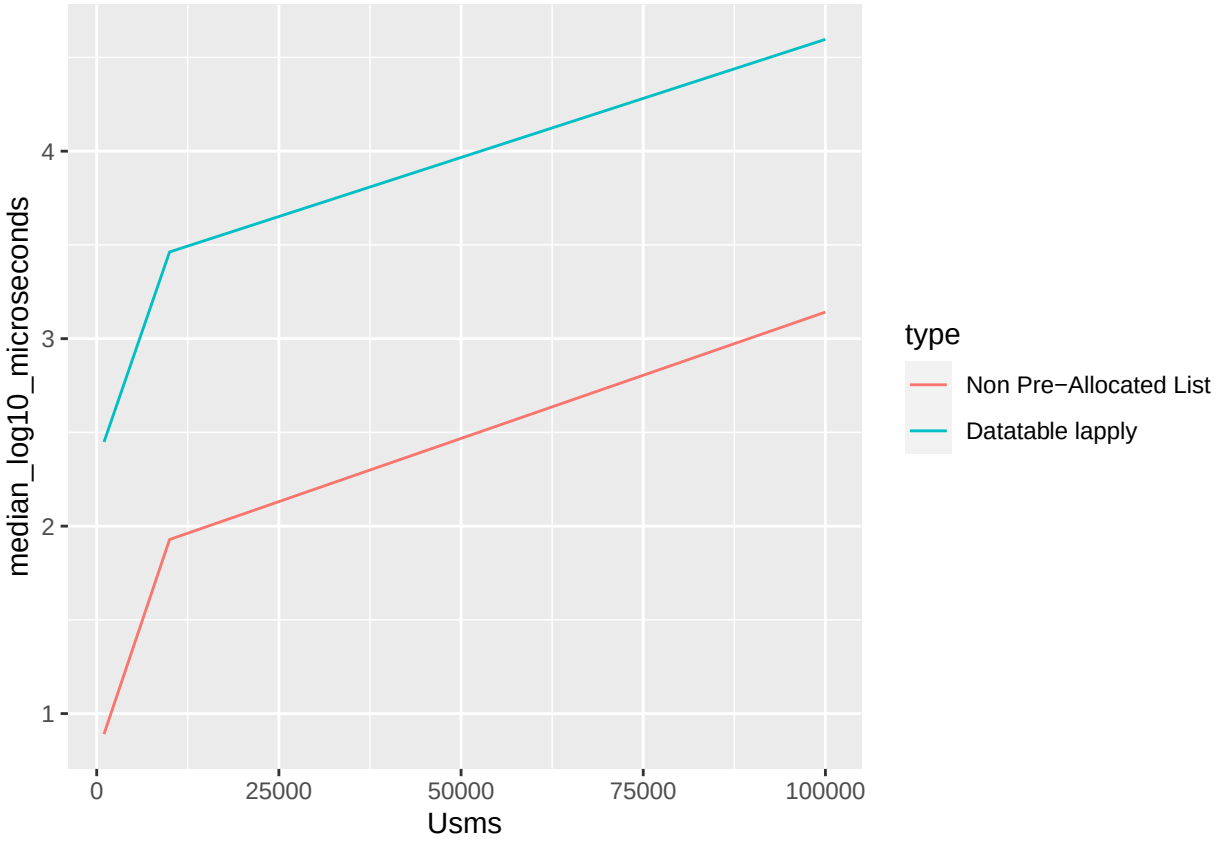
## II.C Best datatype structure

With the previous results, we now try to find the datatype structure in order to store the input data.

```
## Weights graph between List and DataTable
```




## Comparative time graph for the fastest instanciation method for List and DataTable




To begin with, the DataTable type uses less memory than the List type. Then, the List type is faster but we can see that the gap between the two types is not huge.

# III. Conclusion

According to all the analysis made, we can say that the best datatype structure to store the input data is a Dataframe-like structure because it allies a fast instanciation time, a fast execution time for sublists extraction and it is light in memory consumption.