Second Unit Report

Moroco Ramos, Luis Angel

July 5, 2021

Abstract

This a report of laboratories Number: 5, 6, 7 y 8 from the second unit. Here I did a analizis of algorithms like Max and Quick and we did used Dinamic Programming for solved problemas like Fibonacci and get the numer of squeares of a cartesian point. I did used g++ for compile CPP code and PYTHON3 for execute python code in Linux and Visual Studio Code how IDE.

Processor: RYZEN 3600X RAM: 8 GB OS: Linux Mint 20.04 Ulyana .

1 Introduction

I will develop labs 5, 6, 7 and 8 in order in this report will contain screenshots of the execution and tables with the results.

2 Solving the Laboratory N 6

2.1 Laboratory N 6.1 : Demonstrate the average case of the maximum algorithm

Generate 200 vectors of size 256, 512, 1024, ... 1048576. Calculate the average number of times line 4 is executed. Create a table showing

- n
- Average
- 1+ln(n)

For solve this problem we've generate 200 different vector each one size() and evaluate each one ans save its results, we execute just now.

```
std::vector<int> n = {256, 512, 1024, 2048,
                        4096, 8192, 16384, 32768,
                        65536, 131072, 262144, 524288,
                        1048576};
int n_vecs = 200;
```

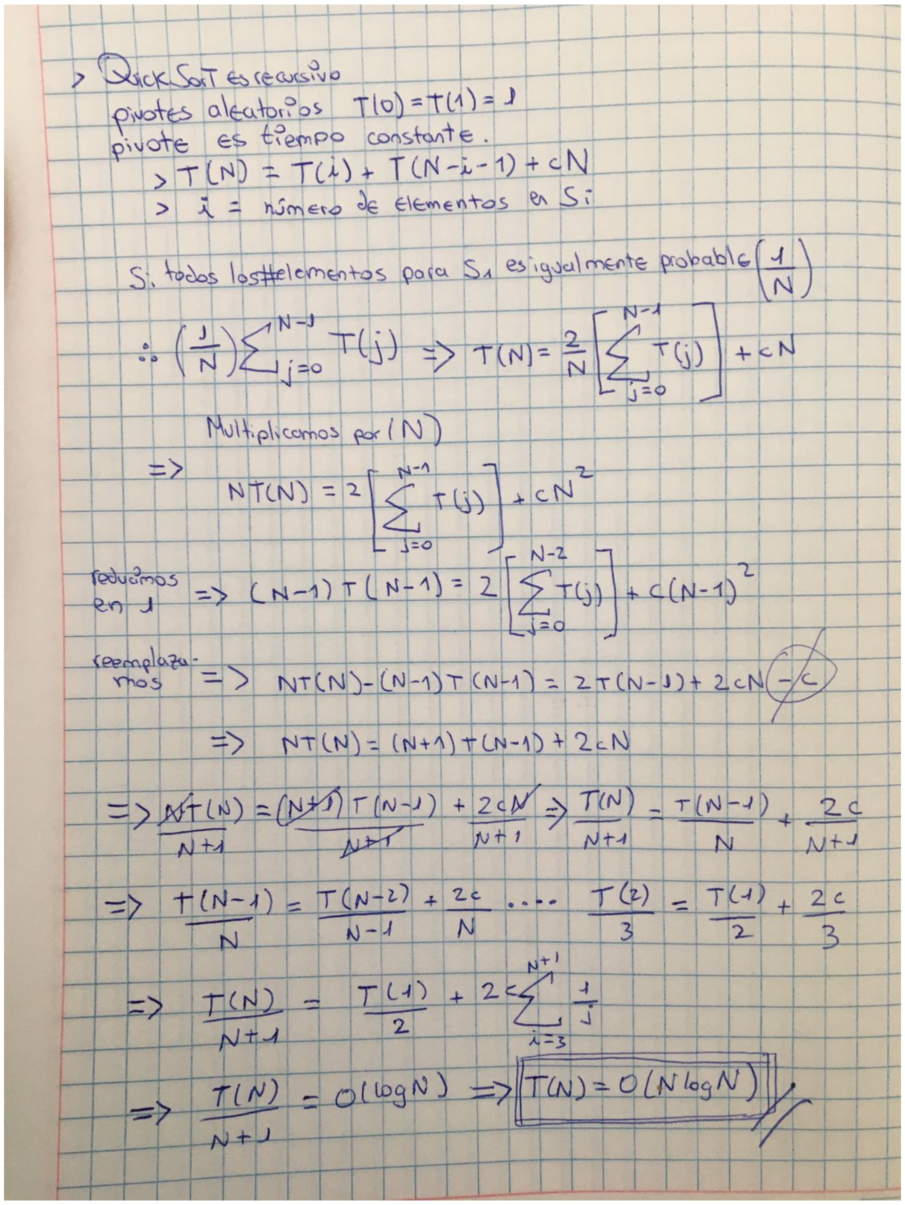
This is the code of Maximum Algorithm:

```
int get_max(std::vector<int> v)
{
    int max = v[0], cont = 0;
    for (auto i = 1; i < v.size(); ++i)
    {
        if (v[i] > max)
        {
            max = v[i];
            ++cont;
        }
    }
    return cont;
}
```

The next step is generate the results

n	average	ln(n)+1
256	4.98	2.60944
512	5.64	7.23832
1024	6.365	7.93147
2048	7.23	8.62462
4096	7.655	9.04045
8192	8.17	8.61776
16384	9.09	10.7041
32768	9.685	11.0047
65536	10.115	10.6047
131072	10.97	12.2752
262144	11.94	12.701
524288	12.6	14.1698
1048576	13.095	14.4776

2.2 Find the average case in QuickSort



> QuickSort es recursivo
pivotes aleatorios $T(0) = T(1) = 1$
pivote es tiempo constante.

> $T(N) = T(i) + T(N-i-1) + cN$
> i = número de elementos en S_i

S_i todos los ~~#~~ elementos para S_1 es igualmente probable $\left(\frac{1}{N}\right)$

$$\therefore \left(\frac{1}{N}\right)\sum_{j=0}^{N-1} T(j) \Rightarrow T(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} T(j)\right] + cN$$

Multiplicamos por (N)

$$\Rightarrow NT(N) = 2\left[\sum_{j=0}^{N-1} T(j)\right] + cN^2$$

reducimos en 1
$$\Rightarrow (N-1)T(N-1) = 2\left[\sum_{j=0}^{N-2} T(j)\right] + c(N-1)^2$$

reemplazamos
$$\Rightarrow NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN \cancel{-c}$$

$$\Rightarrow NT(N) = (N+1)T(N-1) + 2cN$$

$$\Rightarrow \frac{\cancel{N}T(N)}{N+1} = \frac{\cancel{(N+1)}T(N-1)}{\cancel{N+1}} + \frac{2c\cancel{N}}{N+1} \Rightarrow \frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2c}{N+1}$$

$$\Rightarrow \frac{T(N-1)}{N} = \frac{T(N-2)}{N-1} + \frac{2c}{N} \cdots\cdots \frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}$$

$$\Rightarrow \frac{T(N)}{N+1} = \frac{T(1)}{2} + 2c\sum_{i=3}^{N+1}\frac{1}{j}$$

$$\Rightarrow \frac{T(N)}{N+1} = O(\log N) \Rightarrow \boxed{T(N) = O(N\log N)}$$

3 Solving the Laboratory N 7

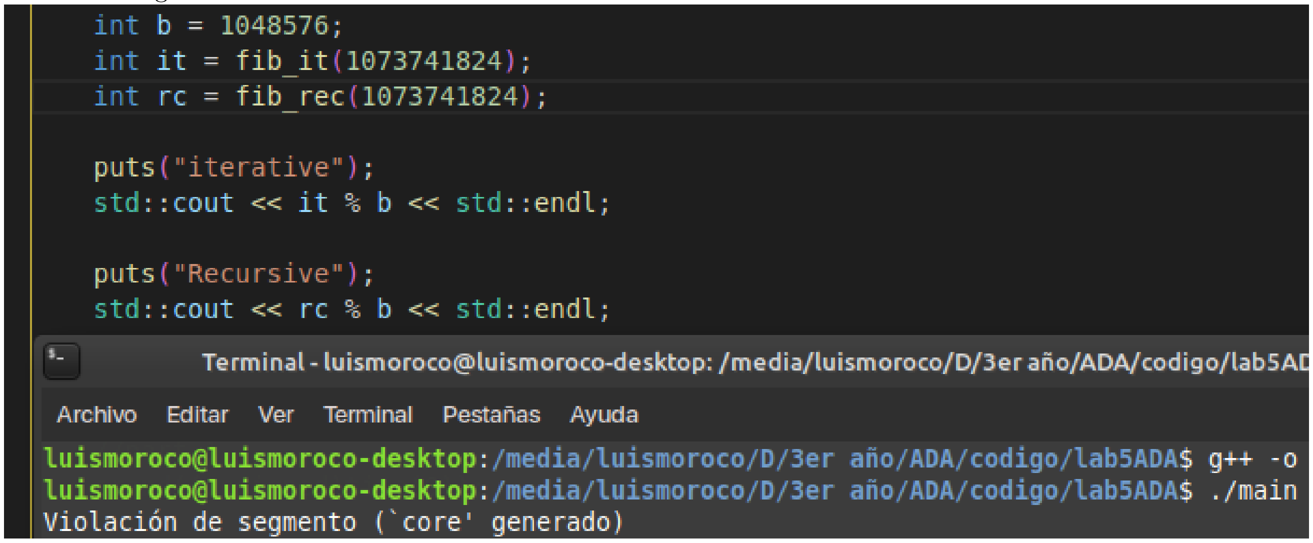
How we can calculate the Fibonacci(2**30) mod 2**20 ?

3.1 Laboratory N 7.1 : Fibonacci recursively and iteratively in C++

For these problems we take the basic Fibonacci form (recursive) and for the iterative form we use pointers to be able to change the variables and update the values iteratively.

```
//fibonacci recursivo
int fib_rec(int n)
{
    if (n <= 1)
        return n;
    return fib_rec(n - 1) + fib_rec(n - 2);
}

//fibonacci iterativo
int fib_it(int n)
{
    if (n <= 1)
        return n;

    int x1 = 0, x2 = 1, temp = 0;
    int *_x1 = &x1;
    int *_x2 = &x2;
    int *_temp = &temp;

    for (int i = 2; i <= n; ++i)
    {
        temp = x1 + x2;
        *_x1 = x2;
        *_x2 = temp;
        *_temp = 0;
    }

    return x2;
}
```

What do we get?



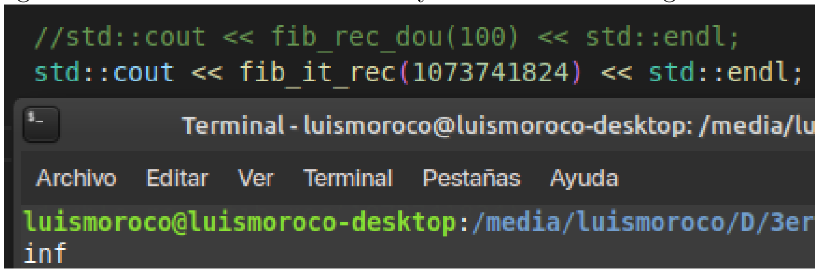
```
    int b = 1048576;
    int it = fib_it(1073741824);
    int rc = fib_rec(1073741824);

    puts("iterative");
    std::cout << it % b << std::endl;

    puts("Recursive");
    std::cout << rc % b << std::endl;
```

Terminal - luismoroco@luismoroco-desktop: /media/luismoroco/D/3er año/ADA/codigo/lab5AD
Archivo Editar Ver Terminal Pestañas Ayuda

```
luismoroco@luismoroco-desktop:/media/luismoroco/D/3er año/ADA/codigo/lab5ADA$ g++ -o
luismoroco@luismoroco-desktop:/media/luismoroco/D/3er año/ADA/codigo/lab5ADA$ ./main
Violación de segmento (`core' generado)
```

3.2 Laboratory N 7.2 : What if we deal with doubles

Using the recursive method the memory is violated and hangs



```
//std::cout << fib_rec_dou(100) << std::endl;
std::cout << fib_it_rec(1073741824) << std::endl;
```

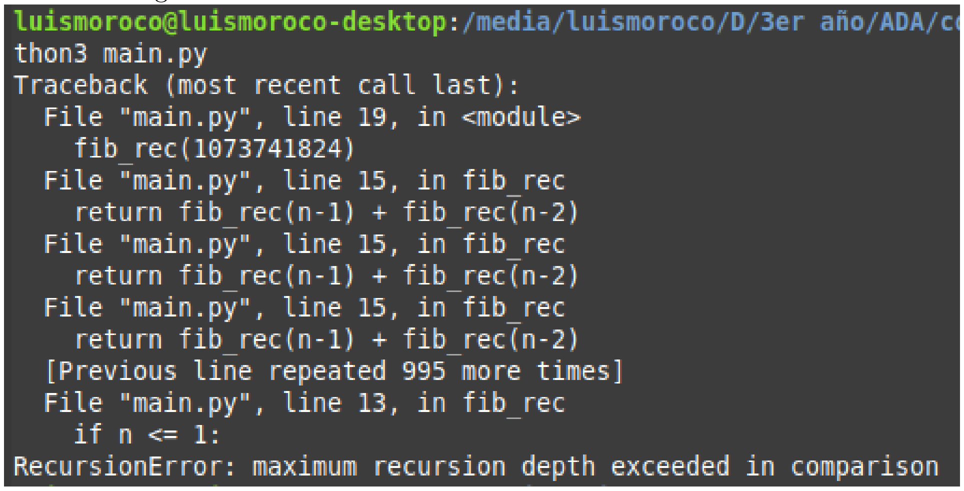
Terminal - luismoroco@luismoroco-desktop: /media/lu
Archivo Editar Ver Terminal Pestañas Ayuda

```
luismoroco@luismoroco-desktop:/media/luismoroco/D/3er
inf
```

3.3 Laboratory N 7.3 : What would happen to memory if we tried it in python?

```python
#Fibonacci iterativo
def fib_ite(n):
    a = 0
    b = 1
    for i in range(1,n):
        b += a
        a = b-a
    return n if n == 0 else b


#Fibonacci recursivo
def fib_rec(n):
    if n <= 1:
        return n
    return fib_rec(n-1) + fib_rec(n-2)
```

What do we get?



```
luismoroco@luismoroco-desktop:/media/luismoroco/D/3er año/ADA/c
thon3 main.py
Traceback (most recent call last):
  File "main.py", line 19, in <module>
    fib_rec(1073741824)
  File "main.py", line 15, in fib_rec
    return fib_rec(n-1) + fib_rec(n-2)
  File "main.py", line 15, in fib_rec
    return fib_rec(n-1) + fib_rec(n-2)
  File "main.py", line 15, in fib_rec
    return fib_rec(n-1) + fib_rec(n-2)
  [Previous line repeated 995 more times]
  File "main.py", line 13, in fib_rec
    if n <= 1:
RecursionError: maximum recursion depth exceeded in comparison
```

A stack overflow occurs

3.4 Laboratory N 7.4 : How would it be useful if we used the theory of Modular Arithmetic?

In this case in the Fibonacci series there are patterns every ”n” elements when finding its module with a number ”m”, it happens with the whole series, we will take advantage of this pattern to be able to find the module of a huge fibonacci number as 2 ** 30 without having to calculate it, because as we have seen so far the number of recurrences or iterations we could increase it using PYTHON with sys but it is still insufficient.

Here we use modular arithmetic, first we find the Pisan period -¿ "m", which is the number that tells us every time the series repeats, and we take the F (n) mod from the Pisan cardinal which would be "s" and thus we can say that F (n) mod m = F (s) mod m considerably reducing the operation and the same way find the rest, for find the answer.

```
def fib_mod(n, m):
    if n <= 1:
        return n
    a = 0
    b = 1
    n = n % get_pisano(m)
    for i in range(n-1):
        aux = 0
        aux = b
        b = (a + b) % m
        a = aux
    return b % m

#execute side
print(fib_mod(1073741824,1048576))
```

And now the answer for Fib(2**30) mod 2**20 is : 651835

3.5 Laboratory N 7.5 : Fibonacci matrix form using divide and conquer

In this section I use the numpy library for matrix multiplications, and division to conquer I didn't manage to include modular arithmetic. I did it in python

```
import numpy as np
# pow generic d_c
def power(x, y, n):
    # execute
    if n == 0:
        return 1
    if n == 1:
        return x
    if n % 2 == 0:
        tmp = power(x, y, n/2)
        return np.dot(tmp, tmp)
    else:
        tmp = power(x, y, (n - 1)/2)
        return np.dot(x, np.dot(tmp, tmp))
def main():
    # data sets
    M = [[1, 1],
         [1, 0]]
    F = [[1, 1],
         [1, 0]]
    print(power(M, F, 5))
main()
output: [[8 5], [5 3]]
```

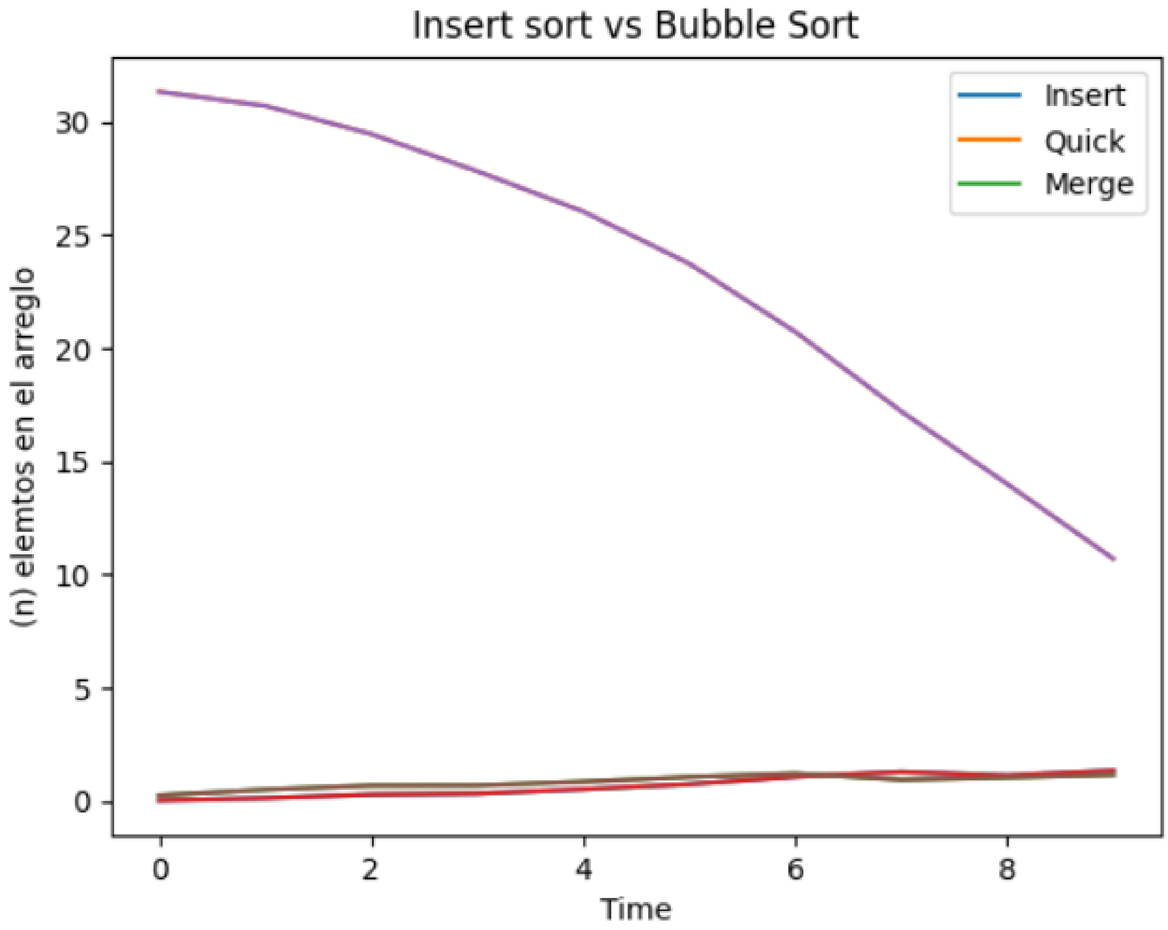
4 Solving the Laboratory N 5

- Do an analysis of the nature of the data and how it influences the results. influences the results - All that glitters is not gold: Why would you want to use one algorithm with more algorithm with greater complexity than another?

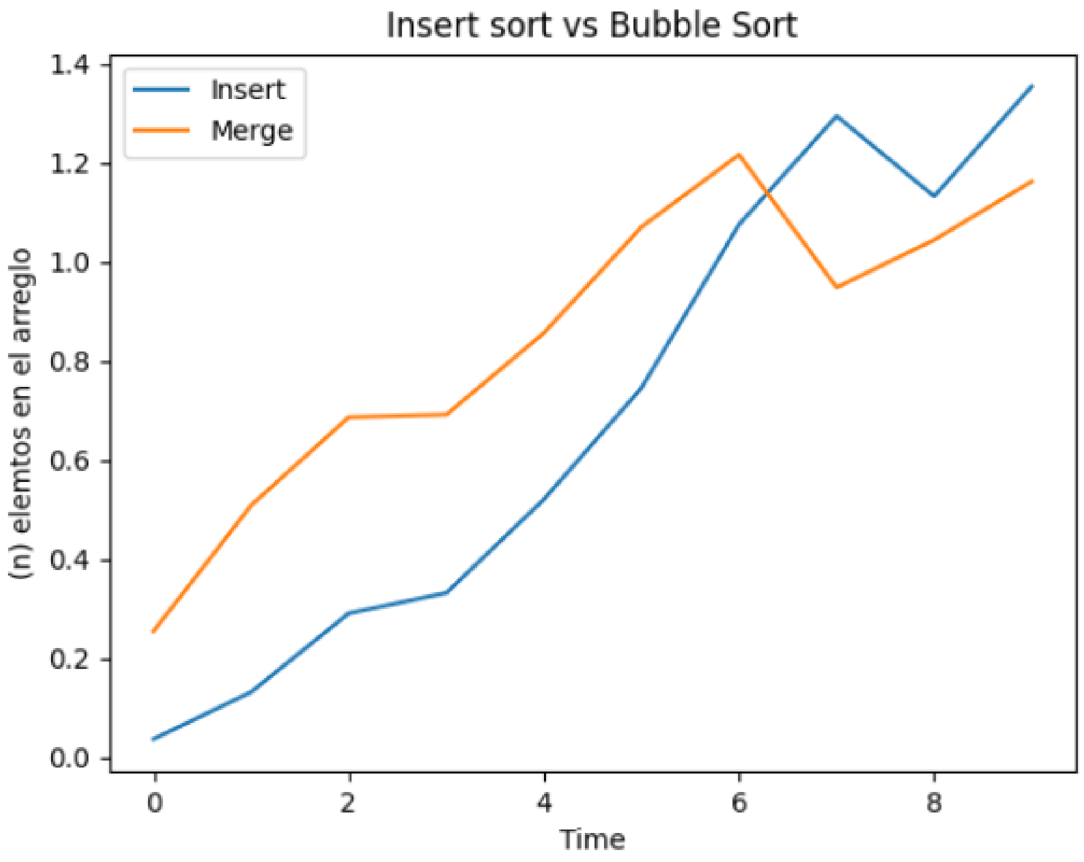
4.1 Laboratory N 5.1 :Insert vs Quick vs Merge in messy arrangements

In ADA we were able to see that, for example, that from problems like insertion sort or hybrid search algorithms the data and above all the number and pre quantity and pre-order have too much influence. The algorithms, however abrupt they may be when they receive a small amount of data, are very data, are very fast, but as the sample grows, they become slower they become slower and we have to think of more intelligent solutions, even if the solutions, even if the construction time is much longer. much longer.

Here i did generated arrays since 100 - 1000



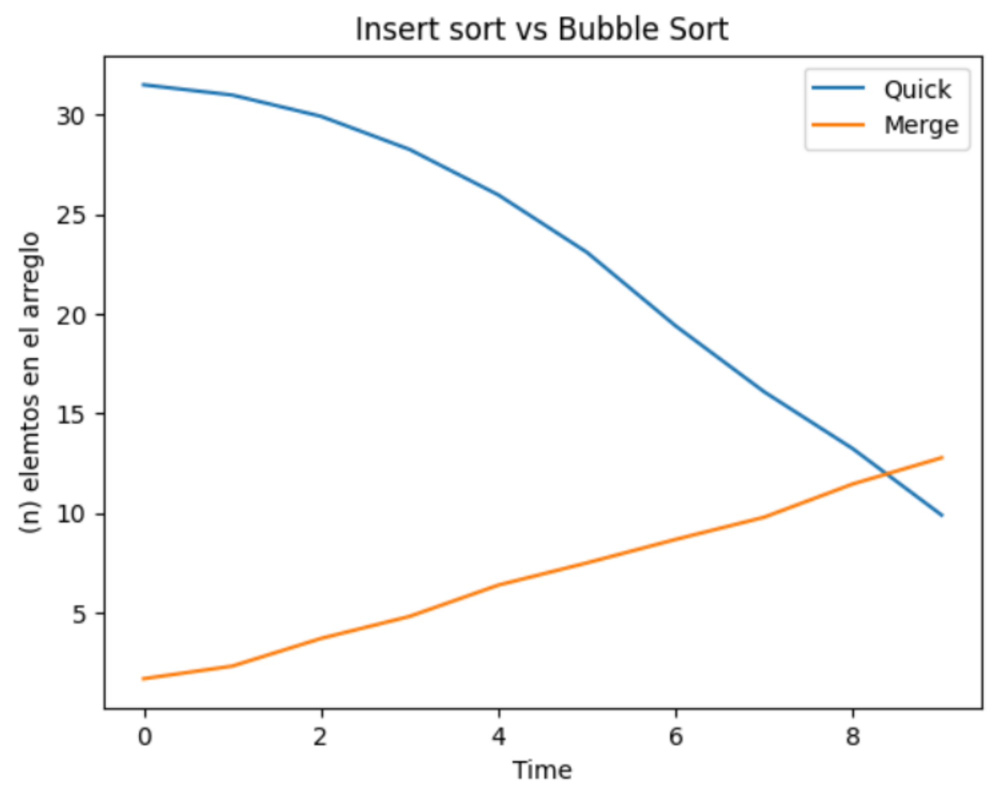
The samples are still very small, but we see a clear decline, of QUICK with a very high start, we can say. decline, from QUICK with a very high start, we can say that QUICK ends up winning for very that QUICK ends up winning for very large number arrangements, but if we are talking about arrangements below a thousand we would have to look a little bit closer.



Here we see that MERGE consumes much less time than everyone else on arrays smaller than 1000.

4.2 Laboratory N 5.2 :Why does one always win in a certain type of test than the other?

We generate numbers from 100 to 10000 and we take their times. As we said before, the 'simple' algorithms simple' algorithms are only useful for small samples and in this case and in this case when they exceed 1000 elements, it is better to use use other algorithms, even if they are more complex, like the QUICK



We can see that MERGE will grow as the number of data number of data does, so MERGE and INSERT work very well on small work very well on small arrays and QUICK for very large very large arrays.

4.3 Laboratory N 5.3 : Conclusion

We can see that for large data you would use QUICK and MERGE and for small lists and arrays INSERT, because of the same nature of data and its size, although in some cases INSERT wins for example for small data sets, where Quick or merge for its complexity of construction and problems when selecting the pivot (QUICK) for tiny arrays and MERGE at the time of splitting would not be suitable. It depends on how much data we want to fix.

5 Solving the Laboratory N 8

How many squares can be formed from a grid? Use dynamic programming

5.1 Laboratory N 8.1 : Using dynamic programming for solve the square problem

Here we use dynamic programming to store how many frames it has in each position and thus not evaluate ”n” times in a loop the same matrix.

```
# Counting squares
def countSquareMatrices(a, N, M, x, y):
    count = 0
    ans = []
    for i in range(1, N):
        for j in range(1, M):
            if (a[i][j] == 0):
                continue
            a[i][j] = min([a[i - 1][j],
                        a[i][j - 1], a[i - 1][j - 1]])+1
            # saving
            count += a[i][j]
    return count
    //output : 13,13 cord 4 x 4 = 30
```

END