Moncef Karim AIT BELKACEM; travail non finalisee:

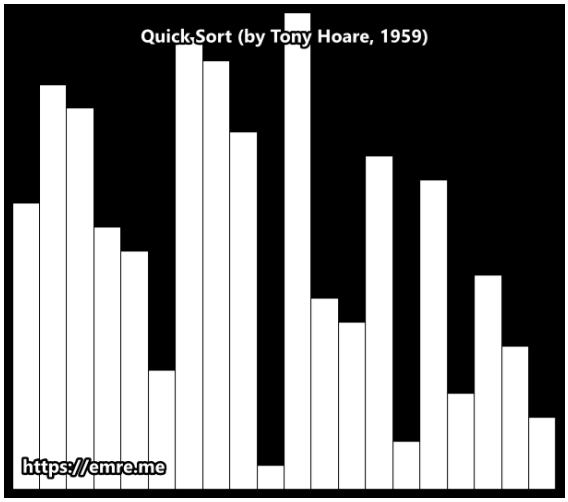
```
- refaire l'echantillonage (limiter les cas extremes en multipliant les tests)
- faire le travail sur les 2 algorithmes restants
- mettre les accents (~claviers sans accents)
- La methode mathematique est correcte (math appliquees - taux d'erreurs)
```

Grand Oral de Mathematique:

> **Comment évaluer l'efficacité d'un algorithme ?**
>
> **Introduction:** Plusieurs algorithmes nous permettent d'effectuer le meme traitement de l'information; neanmoins certains sont plus efficaces que d'autres suivant la quantite de donnees a traiter.
>
> **Etude de cas:**
>
> ```
> Nous cherchons à trier une liste de nombres par ordre croissant
> Nous considérons quatre algorithmes:
>     1. Quick Sort
>     2. Bubble Sort
>     3. Insertion Sort
>     4. Bogo Sort
> ```

Algorithme 1: Quick Sort

In [1]:
```
def quickSort(arr):
    """Sort array using Bubble Sort"""
    elements = len(arr)
    if elements < 2:
        return arr
    current_position = 0
    for i in range(1, elements):
         if arr[i] <= arr[0]:
              current_position += 1
              temp = arr[i]
              arr[i] = arr[current_position]
              arr[current_position] = temp
    temp = arr[0]
    arr[0] = arr[current_position]
    arr[current_position] = temp
    left = quickSort(arr[0:current_position])
    right = quickSort(arr[current_position+1:elements])
    arr = left + [arr[current_position]] + right
    return arr
```

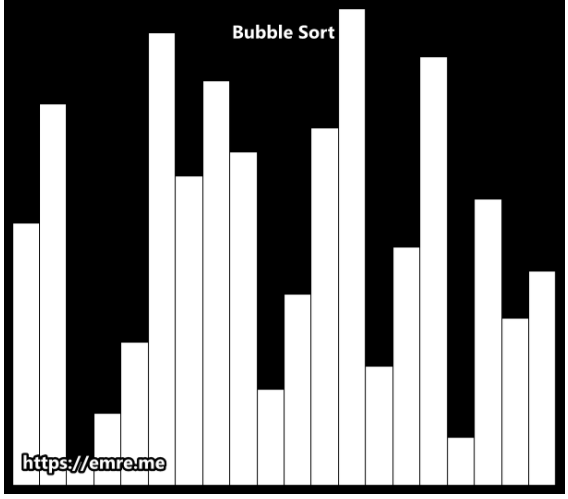


In [2]:
```
liste_de_noms = ["jamy","fred","sabine","marcel"]
quickSort(liste_de_noms)
```

Out[2]:
```
['fred', 'jamy', 'marcel', 'sabine']
```

Algorithme 2: Bubble Sort

In [35]:
```
def bubbleSort(array):
    """Sort array using Bubble Sort"""
    arr = list(array)
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

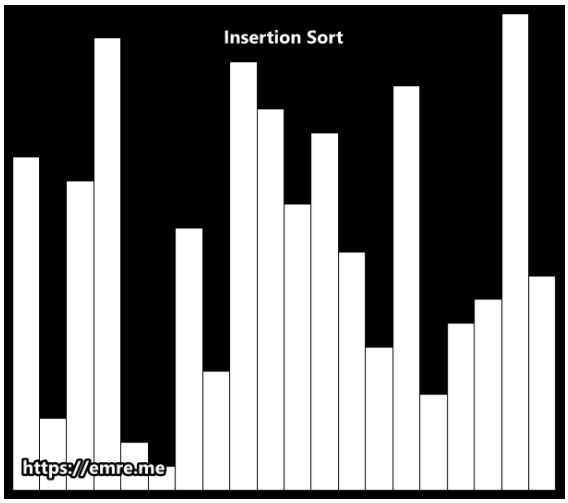


In [4]:
```
liste_de_noms = ["jamy","fred","sabine","marcel"]
bubbleSort(liste_de_noms)
```

Out[4]:
```
['fred', 'jamy', 'marcel', 'sabine']
```

Algorithme 3: Insertion Sort

In [5]:
```
def insertionSort(array):
    """Sort array using Insertion Sort"""
    arr = list(array)
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = key
    return arr
```

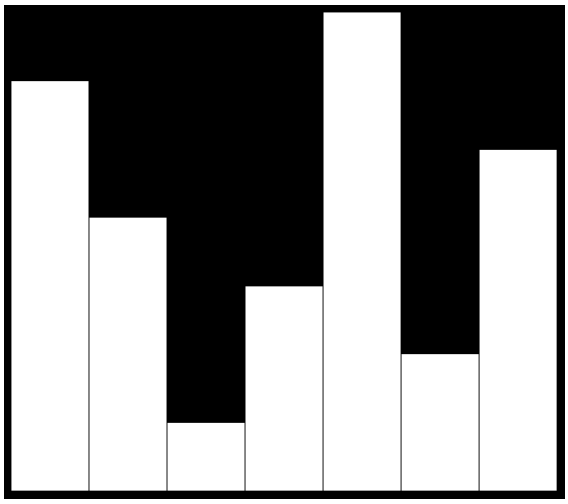


In [6]:
```
liste_de_noms = ["jamy","fred","sabine","marcel"]
insertionSort(liste_de_noms)
```

Out[6]:
```
['fred', 'jamy', 'marcel', 'sabine']
```

Algorithme 4: Bogo Sort

In [7]:
```
def bogoSort(array):
    """Sort arr using Bogo Sort"""
    arr = list(array)
    from random import randint
    def is_sorted(a):
        n = len(a)
        for i in range(0, n-1):
            if (a[i] > a[i+1] ):
                return False
        return True
    def shuffle(a):
        n = len(a)
        for i in range (0,n):
            r = randint(0,n-1)
            a[i], a[r] = a[r], a[i]
    n = len(arr)
    while (is_sorted(arr)== False):
        shuffle(arr)
    return arr
```



In [8]:
```
liste_de_noms = ["jamy","fred","sabine","marcel"]
bogoSort(liste_de_noms)
```

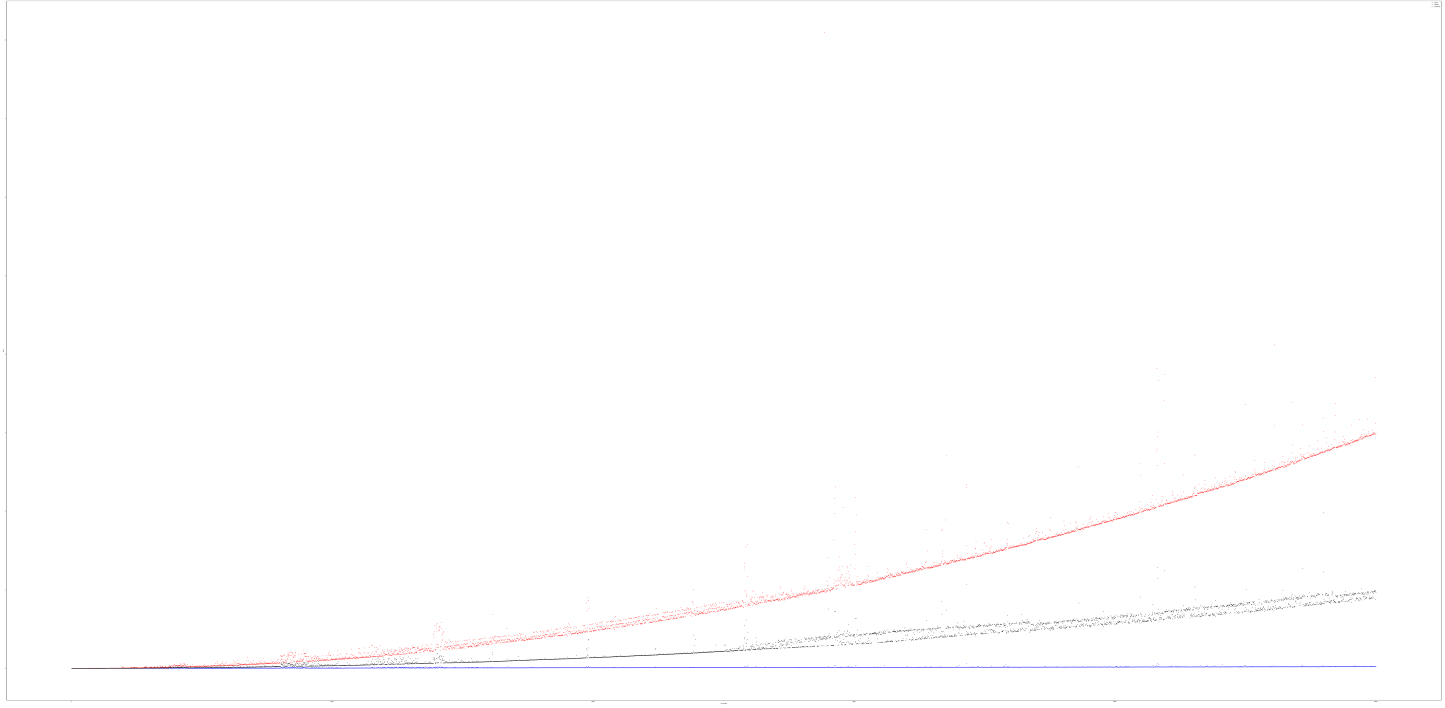
Out[8]:
```
['fred', 'jamy', 'marcel', 'sabine']
```

Graphe des temps d'execution:

Nous executons et chronometrons 3 des 4 algorithmes:

- Quick Sort
- Insertion Sort
- Bubble Sort

In [23]:
```
from numpy import float64
from time import perf_counter as now
from json import loads
from sys import setrecursionlimit
setrecursionlimit(10**6)

with open("numbers.json","r") as d:
    data = loads(d.read())

quick_time_list = []
insertion_time_list = []
bubble_time_list = []

for i in range(2, 5002):
    subdata = data[:i]
    print(i, end=" | ")

    quick_time_start = float64(now())
    quickSort(subdata)
    quick_time_end = float64(now())
    quick_execution_time = quick_time_end - quick_time_start
    quick_time_list.append(quick_execution_time)

    insertion_time_start = float64(now())
    insertionSort(subdata)
    insertion_time_end = float64(now())
    insertion_execution_time = insertion_time_end - insertion_time_start
    insertion_time_list.append(insertion_execution_time)

    bubble_time_start = float64(now())
    bubbleSort(subdata)
    bubble_time_end = float64(now())
    bubble_execution_time = bubble_time_end - bubble_time_start
    bubble_time_list.append(bubble_execution_time)

import matplotlib.pyplot as plt
plt.scatter(range(2, 5002), quick_time_list,label="quick", marker='o',color="blue", s=10)
plt.scatter(range(2, 5002), bubble_time_list, marker='o', label="bubble", color="red", s=10)
plt.scatter(range(2, 5002), insertion_time_list, marker='o', label="insertion", color='black', s=10)
plt.xlabel("list length")
plt.ylabel("time")
plt.legend()
plt.show()
plt.rcParams['figure.figsize'] = [120, 70]
plt.show()
```



Sur la figure ci-contre:
des points d'abcisse L (longueur de la liste {2,5002}) et d'ordonnée T (temps d'execution de l'agorithme de tri).

Bubble Sort • Insertion Sort • Quick Sort •

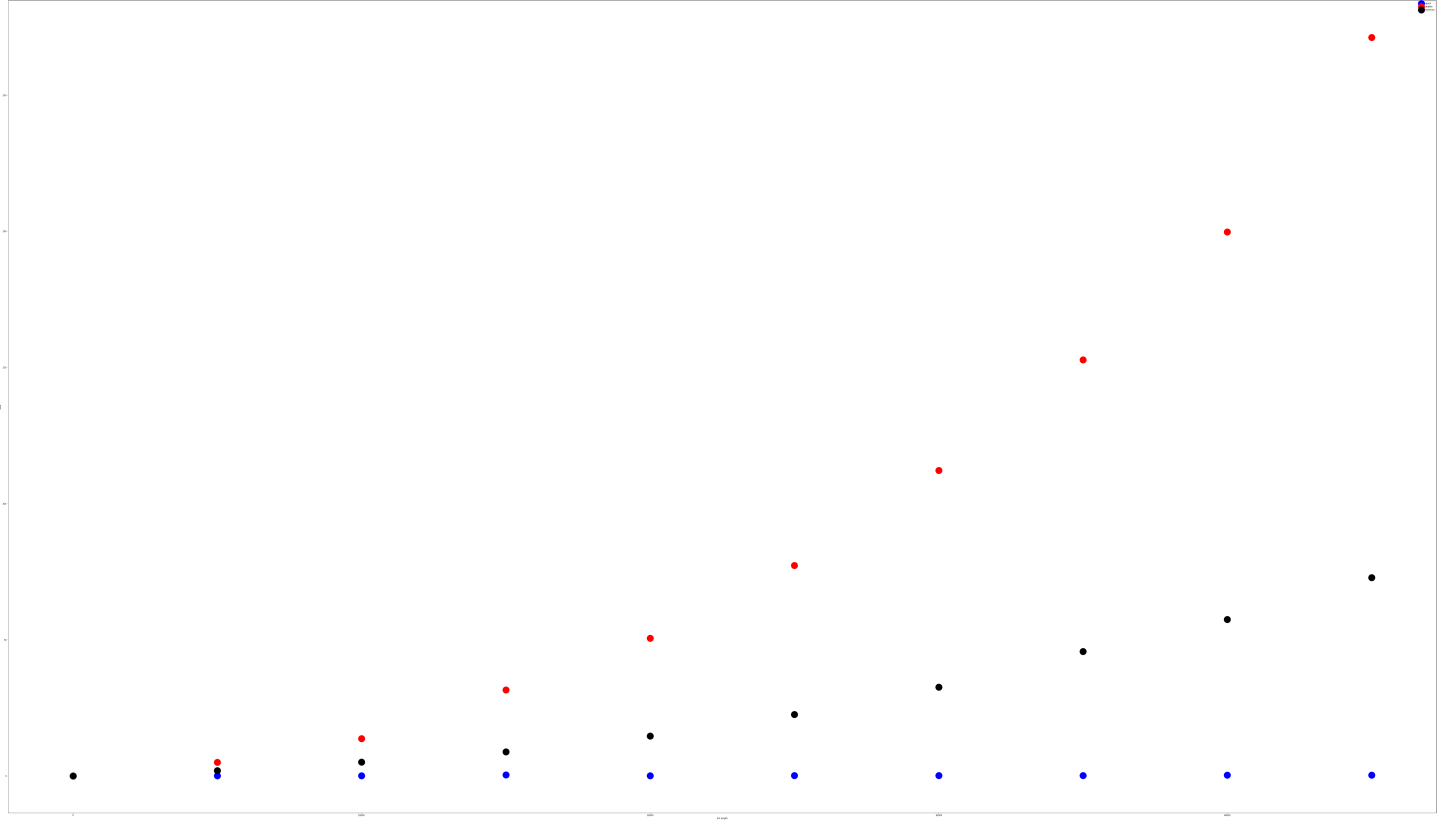
On remarque la presence de courbes principales et de points dispersés ça et là; le chronometrage est une mesure peu exacte, nous nous contenterons d'etudier les points les plus proches de la courbe.
Les valeurs importent peu, leur evolution est l'objectif de notre etude.

C'est pour cela qu nous utiliserons des abcisses plus eloignés (limiter la marge d'erreur (de 2 à 50000))

In [58]:

```
from numpy import float64
from time import perf_counter as now
from json import loads
from sys import setrecursionlimit
setrecursionlimit(10**6)

with open("numbers.json","r") as d:
    data = loads(d.read())

quick_time_list = []
insertion_time_list = []
bubble_time_list = []

for i in range(2, 50000,5000):
    subdata = data[:i]
    print(i, end=" | ")

    quick_time_start = float64(now())
    quickSort(subdata)
    quick_time_end = float64(now())
    quick_execution_time = quick_time_end - quick_time_start
    quick_time_list.append(quick_execution_time)

    insertion_time_start = float64(now())
    insertionSort(subdata)
    insertion_time_end = float64(now())
    insertion_execution_time = insertion_time_end - insertion_time_start
    insertion_time_list.append(insertion_execution_time)

    bubble_time_start = float64(now())
    bubbleSort(subdata)
    bubble_time_end = float64(now())
    bubble_execution_time = bubble_time_end - bubble_time_start
    bubble_time_list.append(bubble_execution_time)

import matplotlib.pyplot as plt
plt.scatter(range(2, 50000,5000), quick_time_list,label="quick", marker='o',color="blue", s=1000)
plt.scatter(range(2, 50000, 5000), bubble_time_list, marker='o', label="bubble", color="red", s=1000)
plt.scatter(range(2, 50000, 5000), insertion_time_list, marker='o', label="insertion", color='black', s=1000)
plt.xlabel("list length")
plt.ylabel("time")
plt.legend()
plt.rcParams['figure.figsize'] = [120, 70]
plt.show()
```

```
2 | 5002 | 10002 | 15002 | 20002 | 25002 | 30002 | 35002 | 40002 | 45002 |
```



Bubble Sort • Insertion Sort • Quick Sort •

Le tableau ci-dessous regroupe ces valeurs.

In [59]:
```
from pandas import Series, DataFrame, concat
q = DataFrame(quick_time_list, index=range(2, 50000, 5000))
b = DataFrame(bubble_time_list, index=range(2, 50000, 5000))
I = DataFrame(insertion_time_list, index=range(2, 50000, 5000))
a = concat([q,b,I], axis=1)
a.columns=["Quick Sort","Bubble Sort","Insertion Sort"]
a
```

Out[59]:

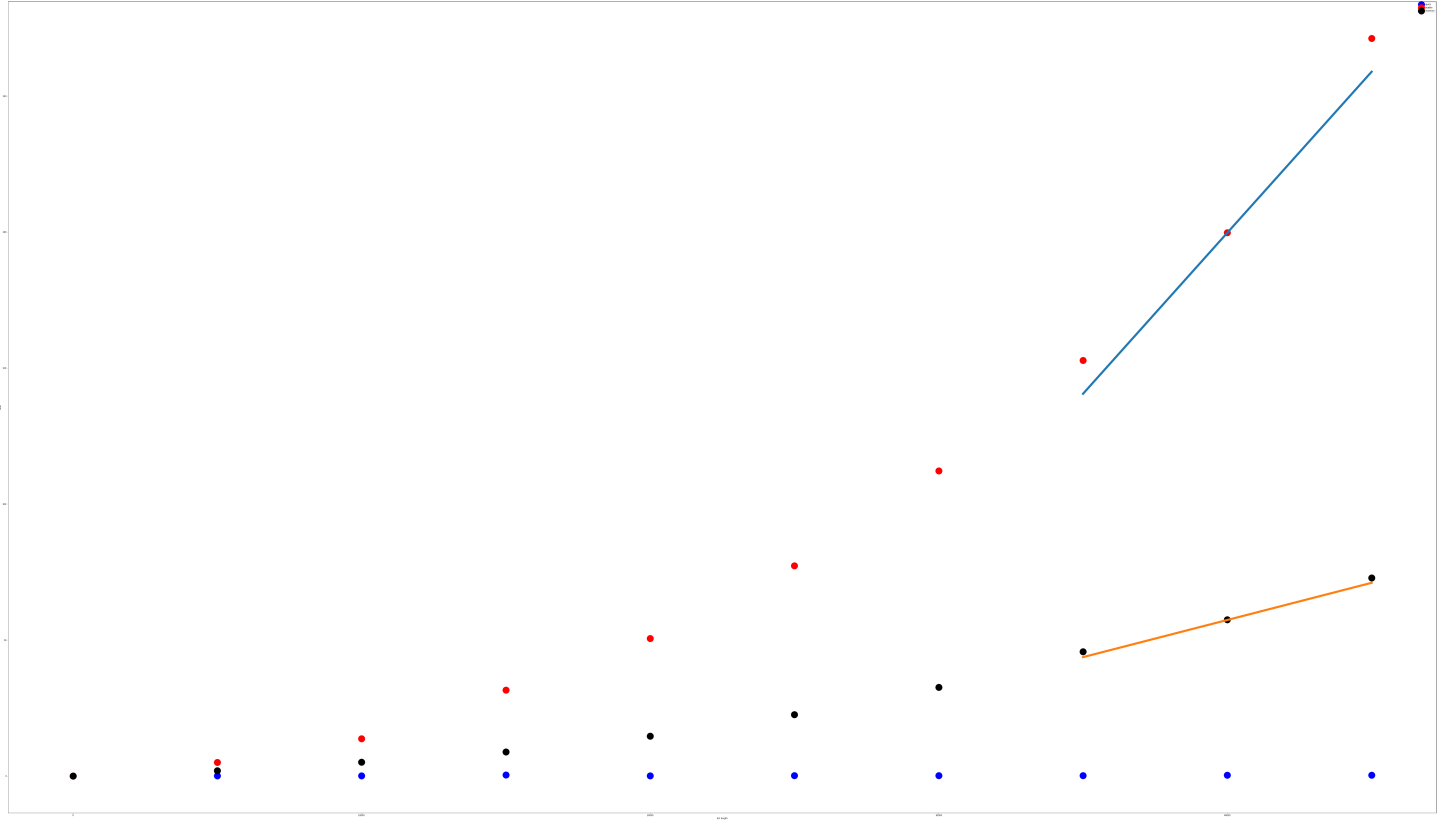
	Quick Sort	Bubble Sort	Insertion Sort
2	0.000018	0.000006	0.000006
5002	0.042743	4.967089	1.930441
10002	0.046710	13.705083	5.089626
15002	0.391331	31.601418	8.817992
20002	0.100364	50.548779	14.631031
25002	0.128250	77.269351	22.564628
30002	0.154332	112.188078	32.565380
35002	0.181586	152.822460	45.695084
40002	0.266142	199.794240	57.489762
45002	0.303592	271.214856	72.818693

Definir l'expression des courbes

Definir l'expression des derivees

> La derivee de la fonction de la courbe C_{bubble} est la fonction qui pour chaque abcisse *x*, associe *y*
> le coefficient directeur de C_{bubble} au point *(x,y)*
> Tracons les tangentes en chaque point *p*
> Pour tracer la tangente en un point ***p*** a la courbe ***C***, on calcule le coefficient directeur de la droite passant par les points ***p-1*** et ***p+1*** (points adjacents sur la courbe).
> Cette tangente est de la forme ***ax+b= y*** ou ***a =*** $\frac{y_{p+1}-y_{p-1}}{x_{p+1}-x_{p-1}}$

In [12]:
```
plt.plot([35002, 45002],[float64(152.822460-12.3),float64(271.214856-12.3)], linewidth=10)
plt.plot([35002, 45002],[float64(45.489762-1.8),float64(72.818693-1.8)], linewidth=10)
plt.scatter(range(2, 50000,5000), quick_time_list,label="quick", marker='o',color="blue", s=1000)
plt.scatter(range(2, 50000, 5000), bubble_time_list, marker='o', label="bubble", color="red", s=1000)
plt.scatter(range(2, 50000, 5000), insertion_time_list, marker='o', label="insertion", color='black', s=1000)
plt.xlabel("list length")
plt.ylabel("time")
plt.legend()
plt.rcParams['figure.figsize'] = [120, 70]
plt.show()
```



Nous remplissons les tableaux suivants; en utilisant l'approximation ci-dessous:
exemple QuickSort pour x=5002:

> $f'(5002) \approx \frac{f(10002)-f(2)}{10002-2}$

Quick Sort

x	f(x)=y	f'(x)
2	0.000018	-----
5002	0.042743	4.6692E-6
10002	0.046710	-3.6099E-6
15002	0.391331	-3.66736E-5
20002	0.100364	
25002	0.128250	
30002	0.154332	
35002	0.181586	
40002	0.266142	
45002	0.303592	-----

Bubble Sort

x	f(x)=y	f'(x)
2	0.000006	-----
5002	4.967089	0.0013705077
10002	13.705083	0.0026634329
15002	31.601418	0.0036843696
20002	50.548779	0.0045667933
25002	77.269351	0.0061639299
30002	112.188078	0.0075553109
35002	152.822460	0.0087606162
40002	199.794240	0.0118392396
45002	271.214856	-----

Insertion Sort

x	y	f'(x)
2	0.000006	-----
5002	1.930441	
10002	5.089626	
15002	8.817992	

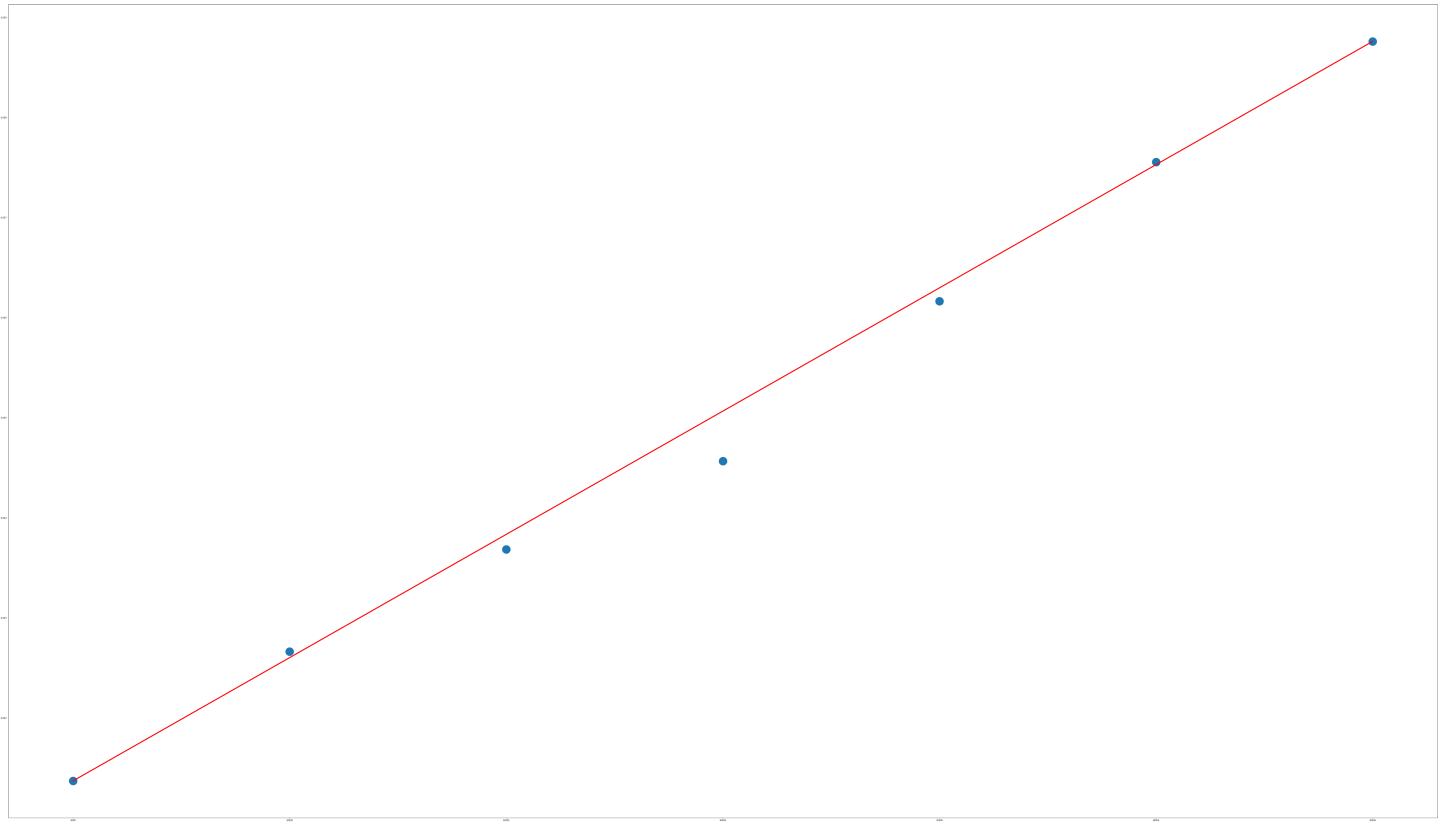
x	y	f'(x)
20002	14.631031	
25002	22.564628	
30002	32.565380	
35002	45.695084	
40002	57.489762	
45002	72.818693	-----

Prediction BubbleSort

> dans un premier temps nous etudierons l'evolution du temps d'execution de l'algorithme BubbleSort. Pour cela nous utilisons les valeurs de *f'(x)* du tableau ci-dessus.

In [38]:

```
derivee_bubble=[
0.0013705077,
0.0026634329,
0.0036843696,
0.0045667933,
0.0061639299,
0.0075553109,
0.0087606162,
0.0118392396]
#0.0118392396

x_derivee_bubble = [
5002 ,
10002,
15002,
20002,
25002,
30002,
35002,
40002]
#40002
plt.plot([5002,35002],[0.0013705077,0.0087606162], linewidth=5, color='red')

plt.scatter(x_derivee_bubble,derivee_bubble,marker='o', s=1500)
plt.show()
```



En placant sur un graphe les differentes valeurs de *f'(x)* (Bubble Sort); on remarque que les points semblent se distribuer autour d'une droite.
On peut alors dire que f'(x) est une fonction affine qui peut s'ecrire sous la forme *ax+b*.

La droite passant par le point (5002, 0.0013705077) et le point (35002, 0.0087606162) est la meilleure approximation.

a est le coefficient directeur de cette droite:

> a ≈ $\frac{0.018392396-0.0013705077}{40002-5002}$ = 2.991169229e-07
> *b* est l'ordonnee du point d'abcisse 0 qui appartient a cette droite (point d'intersection de la droite avec l'axe des ordonnees).
> a ≈ $\frac{0.0013705077-b}{5002-0}$ ==> b ≈ 0.0013705077 - (a * 5002) ≈ 1.25675148e-4

f'(x) ≈ 2.1114595714285714e-07*x* + 1.25675148e-4

Quelle fonction a pour derive f'(x)=2.1114595714285714e-07*x* + 1.25675148e-4 ; ou plus generalement, quelle fonction a pour derive ax + b ?

hypothese:

($\frac{1}{2}$x^2)' = x

k' = kx

donc par identification: *f(x)* = 1.495584615e-7*x*^2 + 1.25675148e-4*x* + C

exemple: f(22000) = 75.151149

In [43]:

```
from numpy import float64
from time import perf_counter as now
from json import loads
from sys import setrecursionlimit
setrecursionlimit(10**6)

with open("numbers.json","r") as d:
    data = loads(d.read())
b = float64(now())
bubbleSort(data[:22000])
b2 = float64(now())
t = b2 - b
t
```

Out[43]: 77.9425115050035

La marge d'erreur est de ((77.9425115050035 - 75.151149)/77.9425115050035) * 100 = 2.565993784%