2 (a)

In [1]:
```
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt
```

In [2]:
```
first = dict(a=1, b=2, c=3, d=4)
val = int()
item = [] or {} or str()

def includes(item,val,start_ind=None):
    if type(item) == dict:
        return val in item.values()
    if start_ind == None:
        start_ind = 0
        return val in item[start_ind:]
    else:
        return val in item[start_ind:]
```

In [3]:
```
print(includes([2, 3, 4], 2, 0)) # True
print(includes([2, 3, 4], 4, 1)) # True
print(includes({'a':1,'b':2}, 1)) # True
print(includes({'a':1,'b':2}, 'a')) # False
print(includes('abcd', 'b')) # True
```

```
True
True
True
False
True
```

2(b)

In [4]:
```
v = []
x = 0
def moving_average(x):
    v.append(x)
    result = (sum(v)/(len(v)))
    return round(result,1)
```

In [5]:
```
# mAvg = moving_average(x)
print(moving_average(10)) #10.0
print(moving_average(11)) #10.5
print(moving_average(12)) #11.0
```

```
10.0
10.5
11.0
```

2(c)

In [6]:
```
from collections import Counter
def same_frequency(num1, num2):
    n1 = str(num1)
    n2 = str(num2)

    frequency1 = Counter(n1)
    frequency2 = Counter(n2)

    if frequency1 == frequency2:
        return True
    return False
```

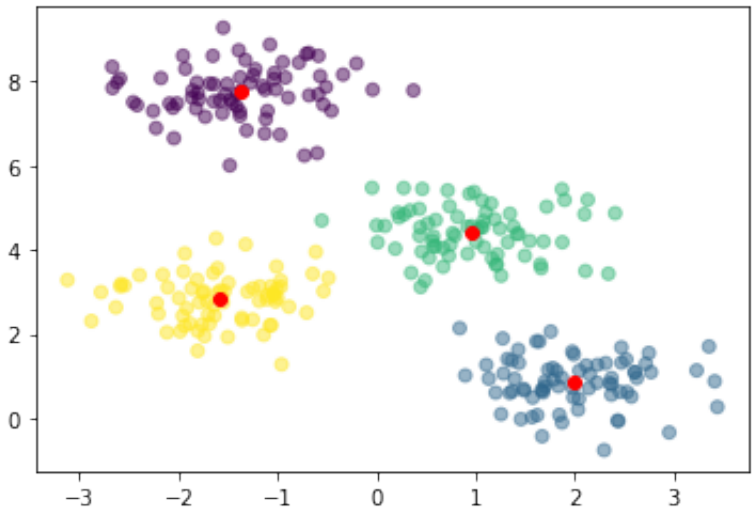
In [7]:
```
print(same_frequency(551122,221515)) # True
print(same_frequency(321142,3212215)) # False
print(same_frequency(12345,31354)) # False
print(same_frequency(1212, 2211)) # True
```

```
True
False
False
True
```

2(d)

In [8]:
```
npzfile = np.load('kmeans.npz')
data = npzfile['data']
pred = npzfile['pred']
centers = npzfile['centers']
#print(pred)
#print(data)
plt.scatter(data[:,0], data[:,1], c = pred,alpha = 0.5 )
plt.scatter(centers[:,0], centers[:,1], c = 'red')
```

Out[8]: <matplotlib.collections.PathCollection at 0x7fe5388d4dc0>



2(e)

In [9]:
```
np.random.seed(24787)
X = np.random.randint(-1000, 1000, size=3000)
Y = np.random.randint(-1000, 1000, size=3000)
# X = X.reshape(-1,1)
# Y = Y.reshape(-1,1)
#print(X)
#print(Y)
i = 0
j = 0
result = np.zeros((3000,3000))
A = np.zeros((3000,1))
B = np.zeros((3000,1))
def NUMPY_outer(A,B):
    for i in range(0,len(A)):
        for j in range(0,len(B)):
            result[i][j] = A[i]*B[j]
    return result
```

In [10]:
```
print(NUMPY_outer(X,Y))
```

```
[[ 288116.  433466.  322354. ...  234498.  459306.  323646.]
 [ 214972.  323422.  240518. ...  174966.  342702.  241482.]
 [-312200. -469700. -349300. ... -254100. -497700. -350700.]
 ...
 [ 180184.  271084.  201596. ...  146652.  287244.  202404.]
 [ -66454.  -99979.  -74351. ...  -54087. -105939.  -74649.]
 [ 203376.  305976.  227544. ...  165528.  324216.  228456.]]
```

In [11]:
```
np.outer(X,Y)
```

Out[11]:
```
array([[ 288116,  433466,  322354, ...,  234498,  459306,  323646],
       [ 214972,  323422,  240518, ...,  174966,  342702,  241482],
       [-312200, -469700, -349300, ..., -254100, -497700, -350700],
       ...,
       [ 180184,  271084,  201596, ...,  146652,  287244,  202404],
       [ -66454,  -99979,  -74351, ...,  -54087, -105939,  -74649],
       [ 203376,  305976,  227544, ...,  165528,  324216,  228456]])
```

The in-built implementation is faster than the function we wrote because numpy uses C, Fortran and and C++ as compiler and C compiles faster than Python especially within loops. Also numpy breaks tasks into fragments and executes them parallelly.