1. Brute Force

In [1]:

```
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

class `BFPortFolio`

- gamma = γ

- if_correlated : boolean.

  False $\rightarrow$ C : general type of covariance matrix

  True $\rightarrow$ C : diagoanl

For generating whole x combinations (method combsums), I referred to the code from https://stackoverflow.com/questions/4632322/finding-all-possible-combinations-of-numbers-to-reach-a-given-sum

In [2]:

```
class BFPortFolio :

    def __init__(self, gamma, n, k, if_correlated):

        self.gamma = gamma; self.n = n ; self.k = k;self.if_correlated = if_correlated

    def datageneration(self):

        np.random.seed(2021)
        e = np.round(np.random.uniform(-0.5, 1.5, size = self.n),2)

        if self.if_correlated:

            C = np.round(np.random.uniform(-1, 1, size = (self.n, self.n)),2)
            C = np.triu(C) + np.triu(C,1).T
            for i in range(self.n):
                C[i][i] = abs(C[i][i])
            C = pd.DataFrame(C)


        else:
            C = pd.DataFrame(np.round(np.diag(np.random.uniform(0, 1,
                                                                size = self.n)) , 2))

        return e, C

    def combsums(self, n, k):

        if n == 1:

            yield (k,)
```

```python
        else:

            for i in range(k + 1):

                for j in self.combsums(n - 1, k - i):

                    yield (i,) + j

    def xgenerator(self):

        self.k = 10**self.k

        L = list(self.combsums(self.n, self.k))
        invk = 1/self.k

        return np.array([list(i) for i in L]) * invk

    def FindCombination(self):

        data = self.datageneration()
        self.e = data[0] ; self.C = data[1]

        x = self.xgenerator()
        ans_vec = np.zeros(len(x))

        if self.if_correlated:

            for i in range(len(ans_vec)):

                ans_vec[i] = self.e.T @ x[i] - self.gamma * x[i].T @ self.C @ x[i]

        for i in range(len(ans_vec)):

            ans_vec[i] = self.e.T @ x[i] - self.gamma * sum(np.diag(self.C) * x[i]**2)

        argmax = np.argmax(ans_vec)

        if ans_vec[argmax] <= 0 :

            return;

        return ans_vec[argmax], x[argmax]
```

Empirical Analysis

1^{st}) When k = 1. Had to do with small $n \in \{3, 6, 9, 12, 15\}$.

In [3]:

```python
size = [3, 6, 9, 12, 15]
time_list = [] ; k = 1

for n in size:

    port = BFPortFolio(gamma = 1, n = n, k = k, if_correlated = True) # gamma = 1
    start_time = time.time()
    port.FindCombination()
    time_list.append(time.time() - start_time)
    print('going to the next iterator')
```

```
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
```

In [4]:
```python
X = pd.DataFrame({'x1' : [n**(10**k + 2) for n in size]})

time = np.array(time_list)

regmodel = LinearRegression().fit(X, time)
time_prediction = regmodel.predict(X)

print(regmodel.coef_, regmodel.intercept_)
```

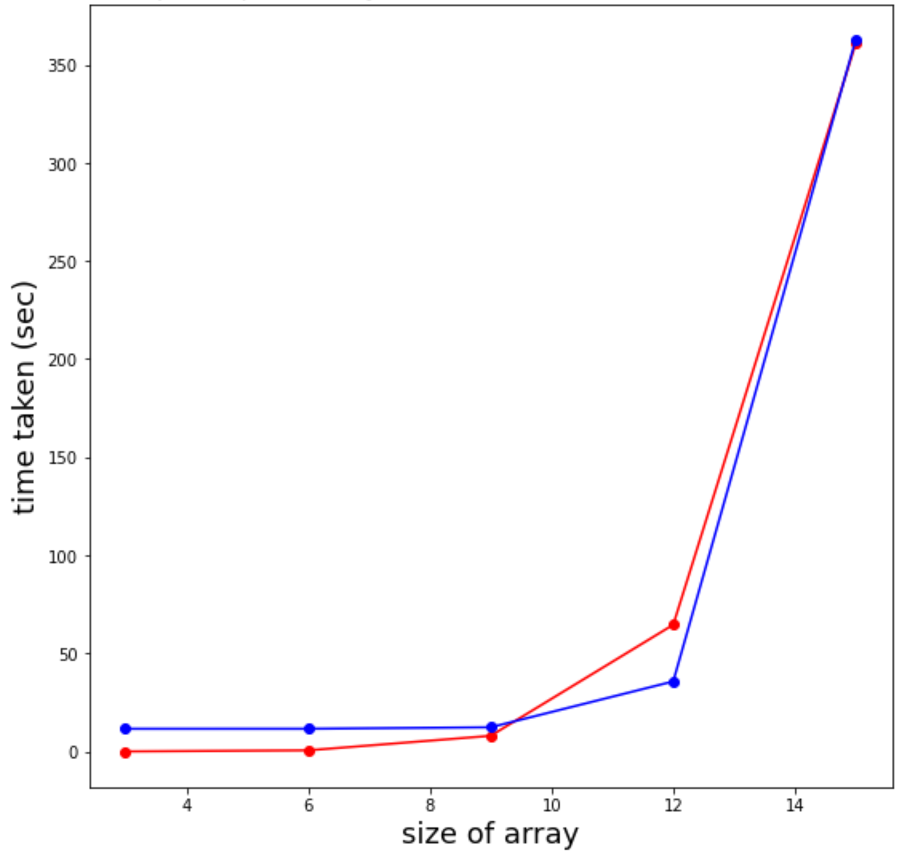
```
[2.70799034e-12] 11.560421601533903
```

$\hat{T}_n \approx (2.43e-12) \cdot n^{12} + 10.416$ for $k = 1$

In [5]:
```python
import warnings
warnings.filterwarnings('ignore')

plt.figure(figsize = (9,9))

plt.plot(np.array(size), np.array(time_list),'-ok', color = 'red')

plt.plot(np.array(size), time_prediction,'-ok', color = 'blue')

plt.xlabel('size of array', fontsize = 18); plt.ylabel('time taken (sec)',
                                                        fontsize= 18)
plt.title('Time taken to find the optimal portfolio by BF for k = 1.
          Red : real, Blue : fitted, 12th order curve fitting',
          fontsize = 15)
```

Out[5]:
```
Text(0.5, 1.0, 'Time taken to find the optimal portfolio by BF for k = 1. Red : real, Bl
ue : fitted, 12th order curve fitting')
```

Time taken to find the optimal portfolio by BF for k = 1. Red : real, Blue : fitted, 12th order curve fitting



2^{nd}) When k = 2. Had to work with even smaller $n \in \{1, 2, 3, 4, 5\}$.

In [6]:

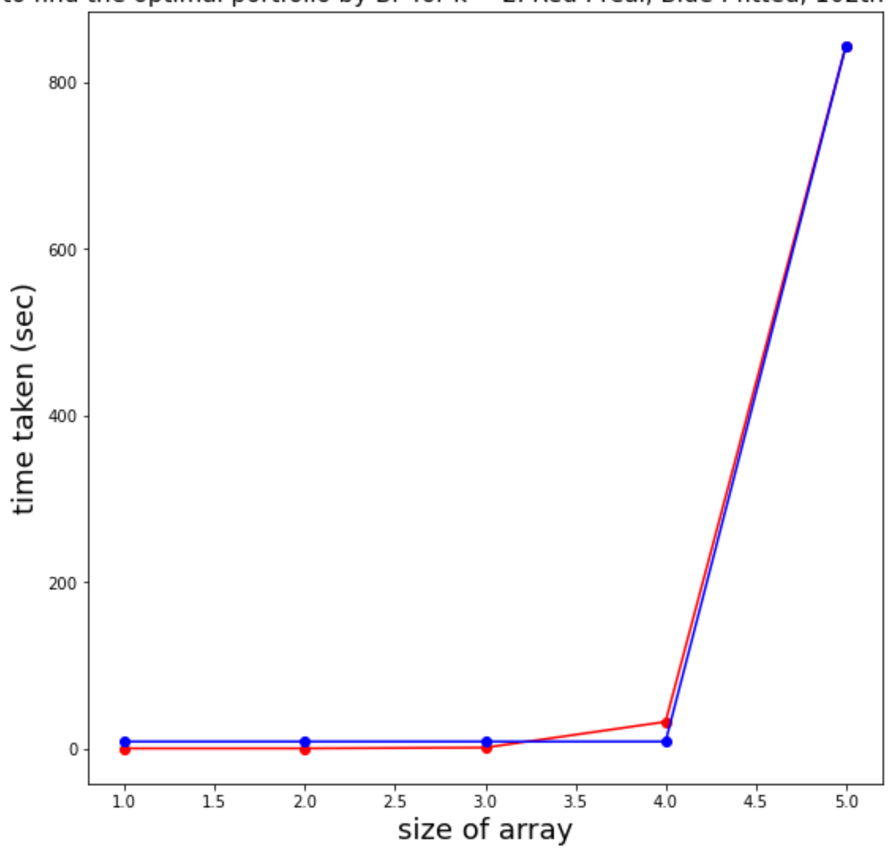
```
import time
size = [1,2,3,4,5]
time_list = [] ; k = 2

for n in size:

    port = BFPortFolio(1, n, k, True) # gamma = 1
    start_time = time.time()
    port.FindCombination()
    time_list.append(time.time() - start_time)
    print('going to the next iterator')


X = pd.DataFrame({'x1' : [n**(10**k + 2) for n in size]})

time = np.array(time_list)

regmodel = LinearRegression().fit(X, time)
time_prediction = regmodel.predict(X)

print(regmodel.coef_, regmodel.intercept_)
```

```
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
[4.23400496e-69] 8.229680689003658
```

In [7]:
```python
import warnings
warnings.filterwarnings('ignore')

plt.figure(figsize = (9,9))

plt.plot(np.array(size), np.array(time_list),'-ok', color = 'red')

plt.plot(np.array(size), time_prediction,'-ok', color = 'blue')

plt.xlabel('size of array', fontsize = 18);plt.ylabel('time taken (sec)',fontsize= 18)
plt.title('Time taken to find the optimal portfolio by BF for k = 2.
          Red : real, Blue : fitted, 102th order curve fitting',
          fontsize = 15)
```

Out[7]:
```
Text(0.5, 1.0, 'Time taken to find the optimal portfolio by BF for k = 2. Red : real, Bl
ue : fitted, 102th order curve fitting')
```

Time taken to find the optimal portfolio by BF for k = 2. Red : real, Blue : fitted, 102th order curve fitting



Comments on Brute Force

The execution time may be different by computers but even with extremely small choice of $(n, k) = (5, 2)$, it takes more than 11 minutes.

Considering that

1) We often include more than 10 items in a portfolio

2) We often need at least an accuracy of two floating numbers,

this Brute Force algorithm is disastrous.

2. In case of C: diagonal, whether Greedy Algorithm is plausible

In [8]:
```
port = BFPortFolio(1, 8, 1, False) # gamma = 1, C is diagonal (to simplify)

data = port.datageneration()

data[0] # e
```

Out[8]:
```
array([ 0.71,  0.97, -0.22,  0.13,  1.49, -0.24, -0.14,  1.01])
```

In [9]:
```
data[1] # C
```

Out[9]:

	0	1	2	3	4	5	6	7
0	0.66	0.00	0.0	0.00	0.00	0.00	0.00	0.00
1	0.00	0.78	0.0	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.1	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.0	0.06	0.00	0.00	0.00	0.00
4	0.00	0.00	0.0	0.00	0.96	0.00	0.00	0.00
5	0.00	0.00	0.0	0.00	0.00	0.62	0.00	0.00
6	0.00	0.00	0.0	0.00	0.00	0.00	0.09	0.00
7	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.56

In [10]:
```
port.FindCombination()
```

Out[10]:
```
(0.9222000000000001, array([0.1, 0.2, 0. , 0. , 0.4, 0. , 0. , 0.3]))
```

Check if slightly changing the proportion within the selected item (proportion > 0) would output a different objective function value.

Trial 1) Slight change in proportions of two best items (best : highest proportions output by the function above)

In [11]:
```
x = np.array([0.1, 0.2, 0, 0, 0.3, 0, 0, 0.4])

C = np.diag([0.66, 0.78, 0.1, 0.06, 0.96, 0.62, 0.09, 0.56])

e = np.array([0.71, 0.97,-0.22, -0.13, 1.49, -0.24, -0.14, 1.01])

print("When I slightly change the proportions of two best items,
      the objective function value gets smaller to ",
      e.T @ x - 1 * x.T @ C @ x)
```

```
When I slightly change the proportions of two best items, the objective function value g
ets smaller to  0.9022000000000001
```

Trial 2) When I only buy the best item : $i = 4$

In [12]:
```
print("If I only invest on the best item,
      the objective function value gets smaller to ",
      e[4] - 1* C[4][4])
```

```
If I only invest on the best item, the objective function value gets smaller to  0.53
```

Thus, Greedy Algorithm does not work even for C : Diagonal

Dynamic Programming results in the optimal answer and DP starts from the next report.

In [1]:
```python
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

In [2]:
```python
class BFPortFolio :

    def __init__(self, gamma, n, k, if_correlated):
        self.gamma = gamma; self.n = n ; self.k = k; self.if_correlated =if_correlated

    def datageneration(self):
        np.random.seed(2021)
        e = np.round(np.random.uniform(-0.5, 1.5, size = self.n),2)

        if self.if_correlated:

            C = np.round(np.random.uniform(-1, 1, size = (self.n, self.n)),2)
            C = np.triu(C) + np.triu(C,1).T
            for i in range(self.n):
                C[i][i] = abs(C[i][i])
            C = pd.DataFrame(C)

        else:
            C = pd.DataFrame(np.round(np.diag(np.random.uniform(0,
                                                     1, size = self.n)) , 2)

        return e, C

    def combsums(self, n, k):
        if n == 1:
            yield (k,)
        else:
            for i in range(k + 1):
                for j in self.combsums(n - 1, k - i):
                    yield (i,) + j

    def xgenerator(self):
        self.k = 10**self.k
        L = list(self.combsums(self.n, self.k))
        invk = 1/self.k
        return np.array([list(i) for i in L]) * invk

    def FindCombination(self):
        data = self.datageneration()
        self.e = data[0] ; self.C = data[1]
        x = self.xgenerator()
        ans_vec = np.zeros(len(x))

        if self.if_correlated:
            for i in range(len(ans_vec)):
                ans_vec[i] = self.e.T @ x[i] - self.gamma * x[i].T @ self.C @ x[i]

        for i in range(len(ans_vec)):
            ans_vec[i] = self.e.T @ x[i] - self.gamma * sum(np.diag(self.C) * x[i]**2)
        argmax = np.argmax(ans_vec)

        if ans_vec[argmax] <= 0 :
            return;
```

```python
        return ans_vec[argmax], x[argmax]
```

3. Dynamic Programming

3.1. First, for diagonal C case

In [3]:

```python
class DPPortFolio_diagonal:

    """
    Deals with diagonal C so that the parameter if_correlated is unnecessary
    """

    def __init__(self, gamma, n, k):

        self.gamma = gamma; self.n = n ; self.k = k

    def datageneration(self):

        np.random.seed(2021)
        e = np.round(np.random.uniform(-0.5, 1.5, size = self.n),2)
        C = pd.DataFrame(np.round(np.diag(np.random.uniform(0, 1, size = self.n)) , 2))

        return e, C

    def create_record_table(self):

        mark = pd.DataFrame(np.zeros((self.n, 10**self.k)))
        mark.index = np.arange(1, self.n + 1)
        mark.columns = np.arange(1, 10**self.k + 1) / 10**self.k
        return mark


    def FindCombination(self):

        """returns 1) the record table of utility & 2) optimal proportion vector"""

        data = self.datageneration()
        mark = self.create_record_table()
        optimal_prop = np.zeros(self.n)
        self.e = data[0] ; self.C = data[1]
        invfrac = 1/10**self.k

        for p in mark.columns:    # initialization for 1st row

            mark.loc[1,p] = max(p * self.e[0] - self.gamma * p**2 * self.C[0][0] , 0)


        for i in range(1, self.n - 1): # initialization for 1st col

            mark.iloc[i,0] = max(mark.iloc[i-1, 0], invfrac*self.e[i] -
                                 self.gamma * invfrac**2 * self.C[i][i])

        # recursive equation

        for i in range(1, self.n - 1): # 1,2,3,...,n-2

            for j in range(1, 10**self.k): # 1,2,3,...,10^k - 1
```

```
        lst = []

        lst.append(mark.iloc[i-1, j])

        for l in range(1,j+1): # 1,2,...,j

            z = j - l

            weight = l / 10**self.k

            lst.append(mark.iloc[i-1,z] +  (weight * self.e[i] -
                                             self.gamma * weight**2 *
                                             self.C[i][i]))

        weight = (j+1) / 10**self.k

        lst.append(weight * self.e[i] - self.gamma *
                   weight**2 * self.C[i][i])


        argmax = np.argmax(lst)
        mark.iloc[i,j] = lst[argmax]

# For Loop for mark[n, 1] = mark.iloc[n-1,-1]!

lst = []
lst.append(mark.iloc[self.n-2, -1])

for l in range(1, 10**self.k): # 1,..., 10^k - 1

    z = 10**self.k - l - 1

    weight = l / 10**self.k

    lst.append(mark.iloc[self.n-2, z] + (weight * self.e[self.n-1] -
                                         self.gamma * weight**2 *
                                         self.C[self.n-1][self.n-1]))

lst.append(self.e[self.n-1] - self.gamma * self.C[self.n-1][self.n-1])


argmax = np.argmax(lst)   #newly added
mark.iloc[self.n-1, -1] = lst[argmax]


# Procedure to find the optimal proportion using the record table.

optimal_prop[self.n-1] = argmax / 10**self.k  #newly added


for i in range(self.n-2 ,  -1, -1):

    j = (1 - sum(optimal_prop)) * 10**self.k - 1
    j = round(j) # for iloc indexing. e.g, 6.0 -> 6


    if (mark.iloc[i,j] == mark.iloc[i-1,j]) :

        pass
```

```
            if (mark.iloc[i,j] != mark.iloc[i-1,j]):


                lst = []
                lst.append(mark.iloc[i-1, j])

                for l in range(1, j+1):



                    z = j - l

                    weight = l / 10**self.k

                    lst.append(mark.iloc[i-1,z] +  (weight * self.e[i] -
                                                    self.gamma * weight**2 *
                                                    self.C[i][i]))

                weight = (j+1) / 10**self.k

                lst.append(weight * self.e[i] - self.gamma *
                           weight**2 * self.C[i][i])


                argmax = np.argmax(lst)


                optimal_prop[i] = argmax / 10**self.k

        return mark, optimal_prop
```

Empirical Analysis a) Time Complexity Analysis

$k = 1$

In [4]:

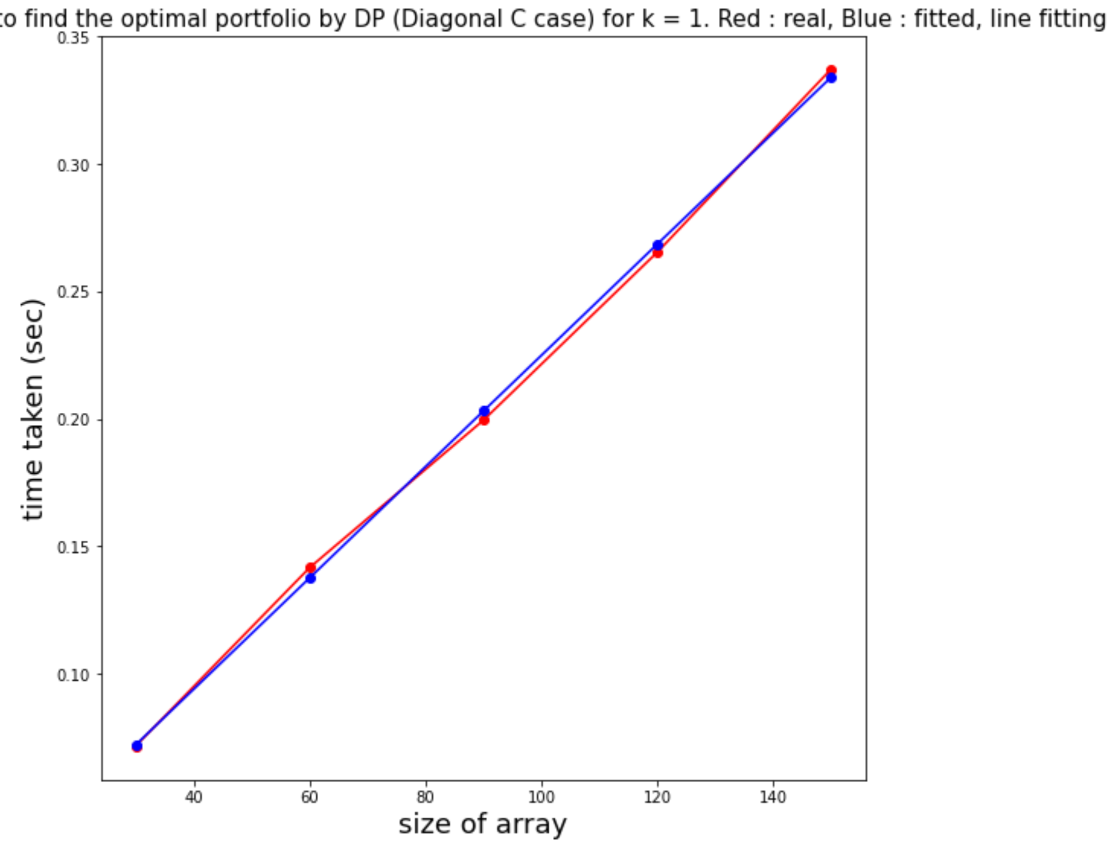
```
import time
size = [30,60,90,120,150]
time_list = [] ; k = 1

for n in size:

    port = DPPortFolio_diagonal(1, n, k) # gamma = 1
    start_time = time.time()
    port.FindCombination()
    time_list.append(time.time() - start_time)
    print('going to the next iterator')


X = pd.DataFrame({'x1' : [n for n in size]})

time = np.array(time_list) ; print(time)

regmodel = LinearRegression().fit(X, time)
time_prediction = regmodel.predict(X)

print(regmodel.coef_, regmodel.intercept_)
```

```
import warnings
warnings.filterwarnings('ignore')

plt.figure(figsize = (9,9))

plt.plot(np.array(size), np.array(time_list),'-ok', color = 'red')

plt.plot(np.array(size), time_prediction,'-ok', color = 'blue')

plt.xlabel('size of array', fontsize = 18); plt.ylabel('time taken (sec)',fontsize=18)
plt.title('Time taken to find the optimal portfolio by DP (Diagonal C case) for k = 1.
          fontsize = 15)
```

```
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
[0.07177973 0.14165044 0.1994586  0.26529074 0.33705926]
[0.00218066] 0.006787943840026872
```

Out[4]:

```
Text(0.5, 1.0, 'Time taken to find the optimal portfolio by DP (Diagonal C case) for k =
1. Red : real, Blue : fitted, line fitting')
```



$k = 2$

In [5]:

```
import time
size = [10,20,30,40,50]
time_list = [] ; k = 2

for n in size:
```

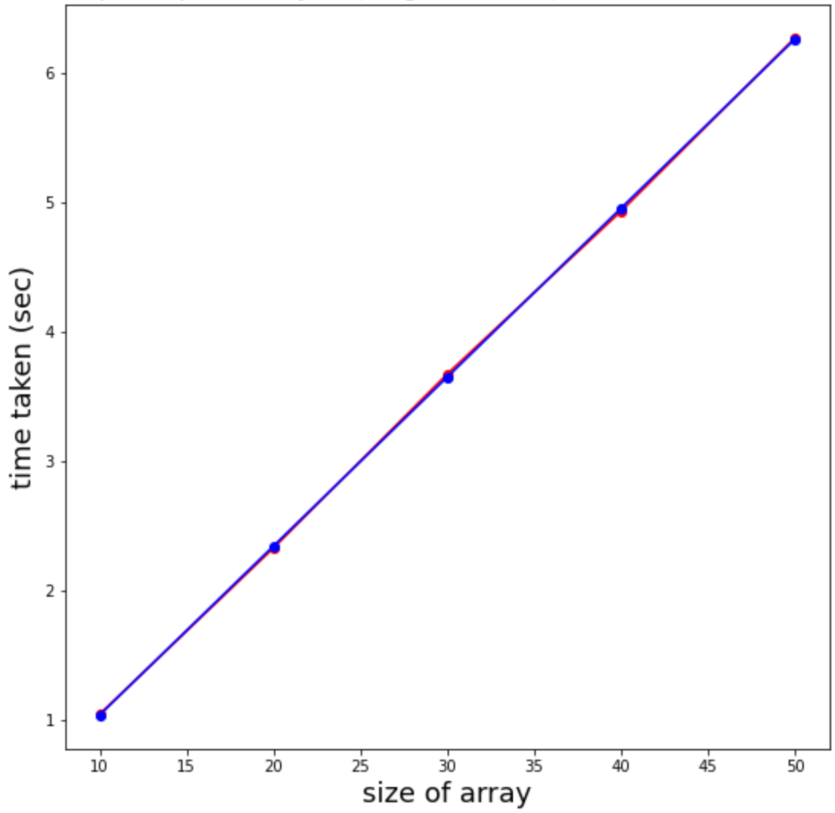
```python
    port = DPPortFolio_diagonal(1, n, k) # gamma = 1
    start_time = time.time()
    port.FindCombination()
    time_list.append(time.time() - start_time)
    print('going to the next iterator')


X = pd.DataFrame({'x1' : [n for n in size]})

time = np.array(time_list) ; print(time)

regmodel = LinearRegression().fit(X, time)
time_prediction = regmodel.predict(X)

print(regmodel.coef_, regmodel.intercept_)



warnings.filterwarnings('ignore')

plt.figure(figsize = (9,9))

plt.plot(np.array(size), np.array(time_list),'-ok', color = 'red')

plt.plot(np.array(size), time_prediction,'-ok', color = 'blue')

plt.xlabel('size of array', fontsize = 18); plt.ylabel('time taken (sec)',fontsize=18)
plt.title('Time taken to find the optimal portfolio by DP (Diagonal C case) for k = 2.
          fontsize = 15)
```

```
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
going to the next iterator
[1.04156494 2.32677031 3.67269588 4.93100286 6.27108312]
[0.13063269] -0.2703572511672987
```

Out[5]:

```
Text(0.5, 1.0, 'Time taken to find the optimal portfolio by DP (Diagonal C case) for k =
2. Red : real, Blue : fitted, line fitting')
```

Time taken to find the optimal portfolio by DP (Diagonal C case) for k = 2. Red : real, Blue : fitted, line fitting



Empirical Analysis b) If results from Brute Force and DP are consistent

For checking if I implemented two algorithms correctly, I compare the results of BF and DP for the same data.

(i) $n = 8, k = 1$ with C : diagonal.

Step1) Data Comparison For Brute Force and DP (have to be the same because I set random seed = 2021)

In [6]:

```
portBF = BFPortFolio(1, 8, 1, False) # gamma = 1, C is diagonal (to simplify)

dataBF = portBF.datageneration()

dataBF
```

Out[6]:

```
(array([ 0.71,  0.97, -0.22,  0.13,  1.49, -0.24, -0.14,  1.01]),
      0     1    2     3     4     5     6     7
 0  0.66  0.00  0.0  0.00  0.00  0.00  0.00  0.00
 1  0.00  0.78  0.0  0.00  0.00  0.00  0.00  0.00
 2  0.00  0.00  0.1  0.00  0.00  0.00  0.00  0.00
 3  0.00  0.00  0.0  0.06  0.00  0.00  0.00  0.00
 4  0.00  0.00  0.0  0.00  0.96  0.00  0.00  0.00
 5  0.00  0.00  0.0  0.00  0.00  0.62  0.00  0.00
 6  0.00  0.00  0.0  0.00  0.00  0.00  0.09  0.00
 7  0.00  0.00  0.0  0.00  0.00  0.00  0.00  0.56)
```

In [7]:
```python
portDP = DPPortFolio_diagonal(1, 8, 1)

dataDP = portDP.datageneration()

dataDP
```

Out[7]:
```
(array([ 0.71,  0.97, -0.22,  0.13,  1.49, -0.24, -0.14,  1.01]),
       0     1    2     3     4     5     6     7
 0  0.66  0.00  0.0  0.00  0.00  0.00  0.00  0.00
 1  0.00  0.78  0.0  0.00  0.00  0.00  0.00  0.00
 2  0.00  0.00  0.1  0.00  0.00  0.00  0.00  0.00
 3  0.00  0.00  0.0  0.06  0.00  0.00  0.00  0.00
 4  0.00  0.00  0.0  0.00  0.96  0.00  0.00  0.00
 5  0.00  0.00  0.0  0.00  0.00  0.62  0.00  0.00
 6  0.00  0.00  0.0  0.00  0.00  0.00  0.09  0.00
 7  0.00  0.00  0.0  0.00  0.00  0.00  0.00  0.56)
```

Step2) Optimal Portfolio utility for Brute Force and DP

In [8]:
```python
BF_Answer = portBF.FindCombination()
DP_Answer = portDP.FindCombination()

display(BF_Answer[0], DP_Answer[0].iloc[-1,-1]) #bottom right of record table

display(BF_Answer[1], DP_Answer[1])
```

```
0.9222000000000001
0.9221999999999999
array([0.1, 0.2, 0. , 0. , 0.4, 0. , 0. , 0.3])
array([0.1, 0.2, 0. , 0. , 0.4, 0. , 0. , 0.3])
```

The answers are consistent.

(ii) $n = 9, k = 1$ with C : diagonal.

Step1) Data Comparison For Brute Force and DP (have to be the same)

In [9]:
```python
portBF = BFPortFolio(1, 9, 1, False) # gamma = 1, C is diagonal (to simplify)

dataBF = portBF.datageneration()

dataBF
```

Out[9]:
```
(array([ 0.71,  0.97, -0.22,  0.13,  1.49, -0.24, -0.14,  1.01,  0.82]),
       0    1     2     3     4     5     6     7     8
 0  0.78  0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 1  0.00  0.1  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 2  0.00  0.0  0.06  0.00  0.00  0.00  0.00  0.00  0.00
 3  0.00  0.0  0.00  0.96  0.00  0.00  0.00  0.00  0.00
 4  0.00  0.0  0.00  0.00  0.62  0.00  0.00  0.00  0.00
 5  0.00  0.0  0.00  0.00  0.00  0.09  0.00  0.00  0.00
 6  0.00  0.0  0.00  0.00  0.00  0.00  0.56  0.00  0.00
```

```
7  0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.62  0.00
8  0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.96)
```

In [10]:
```python
portDP = DPPortFolio_diagonal(1, 9, 1)

dataDP = portDP.datageneration()

dataDP
```

Out[10]:
```
(array([ 0.71,  0.97, -0.22,  0.13,  1.49, -0.24, -0.14,  1.01,  0.82]),
      0    1     2     3     4     5     6     7     8
0  0.78  0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.00
1  0.00  0.1  0.00  0.00  0.00  0.00  0.00  0.00  0.00
2  0.00  0.0  0.06  0.00  0.00  0.00  0.00  0.00  0.00
3  0.00  0.0  0.00  0.96  0.00  0.00  0.00  0.00  0.00
4  0.00  0.0  0.00  0.00  0.62  0.00  0.00  0.00  0.00
5  0.00  0.0  0.00  0.00  0.00  0.09  0.00  0.00  0.00
6  0.00  0.0  0.00  0.00  0.00  0.00  0.56  0.00  0.00
7  0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.62  0.00
8  0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.96)
```

Step2) Optimal Portfolio utility for Brute Force and DP

In [11]:
```python
BF_Answer = portBF.FindCombination()
DP_Answer = portDP.FindCombination()

display(BF_Answer[0], DP_Answer[0].iloc[-1,-1]) #bottom right of record table

display(BF_Answer[1], DP_Answer[1])
```

```
1.0568
1.0568
array([0. , 0.4, 0. , 0. , 0.5, 0. , 0. , 0.1, 0. ])
array([0. , 0.4, 0. , 0. , 0.5, 0. , 0. , 0.1, 0. ])
```

The answers are consistent.

(iii) $n = 3, k = 2$ with C : diagonal.

Step1) Data Comparison For Brute Force and DP (have to be the same)

In [12]:
```python
portBF = BFPortFolio(1, 3, 2, False) # gamma = 1, C is diagonal (to simplify)

dataBF = portBF.datageneration()

dataBF
```

Out[12]:
```
(array([ 0.71,  0.97, -0.22]),
      0    1     2
0  0.31  0.0  0.00
1  0.00  1.0  0.00
2  0.00  0.0  0.13)
```

In [13]:
```
portDP = DPPortFolio_diagonal(1, 3, 2)

dataDP = portDP.datageneration()

dataDP
```

Out[13]:
```
(array([ 0.71,  0.97, -0.22]),
       0    1     2
 0  0.31  0.0  0.00
 1  0.00  1.0  0.00
 2  0.00  0.0  0.13)
```

Step2) Optimal Portfolio utility for Brute Force and DP

In [14]:
```
BF_Answer = portBF.FindCombination()
DP_Answer = portDP.FindCombination()

display(BF_Answer[0], DP_Answer[0].iloc[-1,-1]) #bottom right of record table

display(BF_Answer[1], DP_Answer[1])
```

```
0.5477639999999999
0.5477639999999999
array([0.66, 0.34, 0.  ])
array([0.66, 0.34, 0.  ])
```

The answers are consistent.