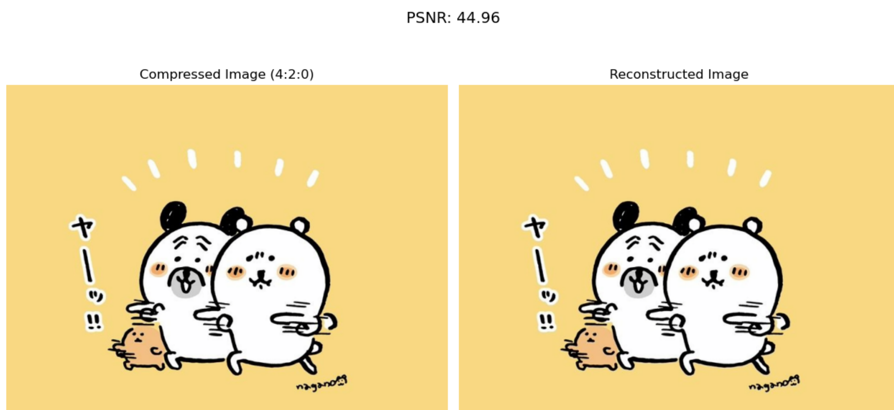
ADSP: HW4

Lo Chun, Chou
R13922136

June 5, 2025

(1)

The following image is the comparison between the 4:2:0 compressed image and the reconstructed image, and the PSNR value is shown in the title:



Since we need to do reconstruction, instead of directly finding a 4:2:0 compressed image, I first use an arbitrary image, compress it, and then reconstruct it.

For the code, please refer to the attached file on NTUCOOL.

(2)

(a)

The two reasons why DCT is used instead of DFT for transformation are:

- DCT is a real-valued transform, while DFT is a complex-valued transform, hence by using DCT, we do not need to deal with imaginary part, which would reduce the complexity of calculation, and requires less memory for storage.
- From the lecture note: "ADSP_Write4.pdf", p.296, we can see that DCT has more energy concentration at low frequencies compared to DFT, therefore, more of the data in the higher frequencies can be discarded, which would have better compression performance.

(b)

Two reasons why the input image is separated into 8x8 blocks before using DCT are:

- We can save memory by using 8x8 blocks, because each of the point in the image needed to be saved in the memory during calculation, so if we only use 8x8 block, we only need to save 64 points in the memory at once, which would reduce the usage of memory.
- Separating the image into 8x8 blocks would reduce the complexity of the calculation, since for $M \times N$ point DCT, its complexity is $\theta(MN)$.

(3)

(a)

The three conditions when two images look similar but the NRMSE is large are:

- If we increase brightness of an image and calculate the NRMSE, the resulting value would be large.

  For example, from the images in lecture note: "ADSP_Write4.pdf", p.302, if we multiply the original image by 0.5, and add with 255.5×0.5, for the pixels that originally have value 0, after the transformation, would have value 127.75, which would give great increase to the NRMSE.
- If we calculate NRMSE for a photo and its negative version, the NRMSE value would be large. (Example in "ADSP_Write4.pdf", p.306)
- If we calculate NRMSE for a photo and a photo with same shape but different intensity, the NRMSE value would be large. (Example in "ADSP_Write4.pdf", p.307)

(b)

The two conditions where two vocal signals sound similar but the NRMSE is large are:

- If the phase is different, even though the two signals have similar frequency, their NRMSE could be large. In the example during lecture, the NRMSE could be even greater than 1, which (1) represents comparing a signal to another signal with all zeros.
- Even when two signals are similar, if there's noise in the background, the NRMSE value would be large.

(4)

We're given the following:

$$P(x=n) = (1-e^{-\lambda})e^{-\lambda n}, \quad \text{for } n = 0, 1, \cdots, 5000, \quad \text{where } \lambda = 0.015$$

Also, we suppose $length(x) = 50000$.

(i)

When using the Huffman code, we have the formula in lecture note: "ADSP_Write4.pdf", p.326:

- Huffman Coding 的 total coding length $b = mean(L)N$ $\quad N$: data length

$$\boxed{ceil\left(N\frac{entropy}{\ln k}\right) \leq b \leq floor\left(N\frac{entropy}{\ln k} + N\right)}$$

Thus, we first calculate the entropy by the formula from lecture note: "ADSP_Write4.pdf", p.325:

- Entropy 熵；亂度 (Information Theory)

$$entropy = \sum_{j=1}^{J} P(S_j)\ln\frac{1}{P(S_j)}$$

using the scipy library in python, we would calculate the entropy and the range of total coding length:

$$ceil(50000 \times \frac{\text{entropy (nats)}}{ln2}) \leq b \leq floor(50000 \times \frac{\text{entropy (nats)}}{ln2} + 50000)$$

The code is as the following:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW4 git:(main)±2 (0.094s)
bat 4_huffman.py

File: 4_huffman.py

import numpy as np
from scipy.stats import entropy

# Parameters
lambda_val = 0.015
n_max = 5000

# aim: Calculate entropy
# Generate n values from 0 to 5000
n = np.arange(n_max + 1)

# Calculate probabilities: P(x=n) = (1-e^(-lambda))e^(-lambda * n)
p = (1 - np.exp(-lambda_val)) * np.exp(-lambda_val * n)

# Calculate entropy (in nats by default)
H = entropy(p)
# Calculate entropy in bits (by dividing by ln(2))
H_bits = H / np.log(2)
print(f"Entropy (nats): {H:.6f}")
print(f"Entropy (bits): {H_bits:.6f}\n")

# aim: Calculate the bounds to get the range of total coding length
N = 50000

# Lower bound: ceil(50000 x 7.501602 / ln2)
lower_bound = np.ceil(N * H_bits)

# Upper bound: floor(50000 x 7.501602 / ln2 + 50000)
upper_bound = np.floor(N * H_bits + N)

print(f"\nBounds calculation:\n")
print(f"Lower bound: {lower_bound:.0f}")
print(f"Upper bound: {upper_bound:.0f}")
```

and the result is:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW4 git:(main)±2 (4.773s)
python3 4.py
Entropy (nats): 5.199714
Entropy (bits): 7.501602


Bounds calculation:

Lower bound: 375081
Upper bound: 425080
```

Thus, the range of total coding length is:

$$375081 \leq b \leq 425080$$

(ii)

If we use the arithmetic code, the range of total coding length is given by the following formula, which is from lecture note: "ADSP_Write4.pdf", p.338:

Total coding length b 的範圍是

$$ceil\left(-N\sum_{m=1}^{M} P_m \log_k P_m\right) \le b \le floor\left(-N\sum_{m=1}^{M} P_m \log_k P_m + \log_k 2\right) + 1$$

binary : $k = 2$

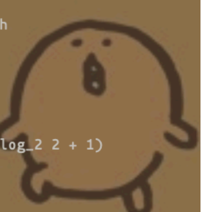
upper bound : $floor\left(N \frac{entropy}{\ln 2} + 2\right)$

$$\boxed{ceil\left(N \cdot \frac{entropy}{\ln k}\right) \le b \le floor\left(N \cdot \frac{entropy}{\ln k} + \log_k 2 + 1\right)}$$

(Compared to page 326)

Similarly, I use the following code to calculate the range of total coding length:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW4 git:(main)±2 (0.123s)
bat 4_arithmetic.py

File: 4_arithmetic.py

import numpy as np
from scipy.stats import entropy

# aim: Calculate entropy
lambda_val = 0.015
n_max = 5000

n = np.arange(n_max + 1)

# Calculate probabilities: P(x=n) = (1-e^(-λ))e^(-λn)
p = (1 - np.exp(-lambda_val)) * np.exp(-lambda_val * n)

# Calculate entropy (in nats by default)
H = entropy(p)
# Calculate entropy in bits (by dividing by ln(2))
H_bits = H / np.log(2)

print(f"Entropy (nats): {H:.6f}")
print(f"Entropy (bits): {H_bits:.6f}\n")

# aim: Calculate bounds to get the range of total coding length
N = 50000  # length(X)
k = 2      # binary coding

# Calculate the range of total coding length b
# For binary coding (k=2):
# ceil(N·entropy / ln k) \leq b \leq floor(N entropy / ln k + log_2 2 + 1)

# Lower bound: ceil(N entropy / ln k)
lower_bound = np.ceil(N * H_bits)

# Upper bound: floor(N·entropy / ln k + log_2 2 + 1)
upper_bound = np.floor(N * H_bits + np.log2(k) + 1)

print(f"Arithmetic Coding Length Bounds:")
print(f"Lower bound: {lower_bound:.0f}")
print(f"Upper bound: {upper_bound:.0f}")

```



and the result is:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW4 git:(main)±2 (1.032s)
python3 4_arithmetic.py
Entropy (nats): 5.199714
Entropy (bits): 7.501602

Arithmetic Coding Length Bounds:
Lower bound: 375081
Upper bound: 375082
```

Thus, the range of total coding length is:

$$375081 \leq b \leq 375082$$

(5)

Let:

$$a = 0.7010,\ b = 0.9239,\ c = 0.3827$$

Then the given system can be written as:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

We can then formulate the system as:

$$\begin{aligned} \begin{bmatrix} y_0 \\ y_2 \end{bmatrix} &= \begin{bmatrix} a & a & a & a \\ a & -a & -a & a \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ &= \begin{bmatrix} a & a \\ a & -a \end{bmatrix} \begin{bmatrix} x_0 + x_3 \\ x_1 + x_2 \end{bmatrix} \end{aligned}$$

which would result in 2 multiplications in this step (by "ADSP_Write5.pdf", p.354).

Then the other part can be written as:

$$
\begin{bmatrix} y_1 \\ y_3 \end{bmatrix} = \begin{bmatrix} b & c & -c & -b \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}
$$

$$
= \begin{bmatrix} b & c \\ c & -b \end{bmatrix} \begin{bmatrix} x_0 - x_3 \\ x_1 - x_2 \end{bmatrix}
$$

which is case 4, and would result in 3 multiplications in this step (by "ADSP_Write5.pdf", p.360).

In total, we need 5 multiplications.

(6)

Consider the case that $\sin\theta = 0$, then the given system (representing the rotation operation):

$$
\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}
$$

would become:

$$
\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}
$$

Similarly, if $\cos\theta = 0$, then the given system would be:

$$
\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0 & \sin\theta \\ -\sin\theta & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}
$$

These two cases happen when:

$$
\begin{aligned}
&\theta = \frac{\pi}{2}k, \quad \text{for } \cos\theta = 0 \\
&\theta = \pi k, \quad \text{for } \sin\theta = 0 \quad \text{where } k \in \mathbb{Z}
\end{aligned}
$$

Hence, we can combine these cases and say that when:

$$
\theta = \frac{\pi}{2}k, \quad \text{for any } k \in \mathbb{Z}
$$

If k is odd:

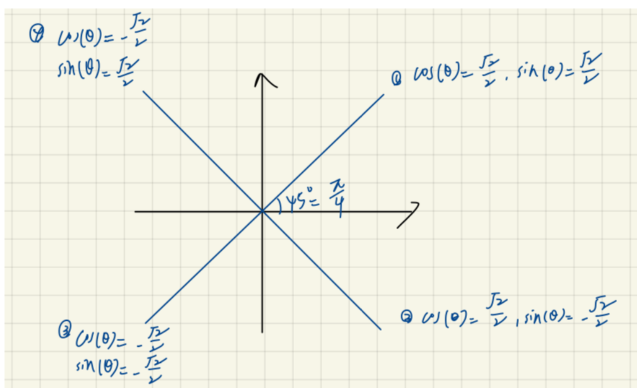
$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

If k is even:

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

As we could see from the above, we only need two multiplications to multiply with ± 1.

Another situation is that when $|\cos\theta| = |\sin\theta|$, which is as the following image:



the given system:

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

would become:

$$
\begin{aligned}
\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} &= \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2}(x_0 + x_1) \\ -\frac{\sqrt{2}}{2}(x_0 + x_1) \end{bmatrix} \qquad \text{or} \\
&= \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2}(x_0 - x_1) \\ \frac{\sqrt{2}}{2}(x_0 + x_1) \end{bmatrix} \qquad \text{or} \\
&= \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} -\frac{\sqrt{2}}{2}(x_0 + x_1) \\ \frac{\sqrt{2}}{2}(x_0 - x_1) \end{bmatrix} \qquad \text{or} \\
&= \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} -\frac{\sqrt{2}}{2}(x_0 - x_1) \\ -\frac{\sqrt{2}}{2}(x_0 + x_1) \end{bmatrix}
\end{aligned}
$$

after calculating $x_0 + x_1$ and $x_0 - x_1$, we only need 2 multiplications, and this happens when:

$$\theta = \frac{\pi}{4} + \frac{\pi}{2}k, \quad \text{for any } k \in \mathbb{Z}$$

(7)

We're given that:

$$
\begin{aligned}
N &= \text{length}(x[n]) = 63 \\
M &= \text{length}(h[n]) = 35
\end{aligned}
$$

To find the optimal number of points for the DFT, I followed the steps in the lecture note: "ADSP_12.pdf", p.445, which is shown as below:

How do we estimate the optimal L? 445

(1) Find L_0 to minimize $\frac{N}{L}3(L+M-1)\left[\log_2(L+M-1)+1\right]$

(2) Estimate P_0 from $P_0 = L_0$+M-1. Then several values of P around P_0 to make MUL_P smaller

(3) Calculate L, S, and the number of real multiplications for each possible P to find the optimal P and L.

I wrote a python code to implement the above steps, and the code is as follows:

```python
import numpy as np
import math
from scipy.optimize import minimize_scalar

# N is the number of points in the DFT, and muls is the number of real multiplications
# This is part of the given table, which can be found in lecture note "ADSP_Write5.pdf", p.377-379
Pvalues_N = [
    {'N': 63, 'muls': 256}, {'N': 64, 'muls': 204}, {'N': 66, 'muls': 284},
    {'N': 70, 'muls': 300}, {'N': 72, 'muls': 164}, {'N': 80, 'muls': 260},
    {'N': 81, 'muls': 480}, {'N': 84, 'muls': 248}, {'N': 88, 'muls': 364},
    {'N': 90, 'muls': 340}, {'N': 96, 'muls': 280}, {'N': 104, 'muls': 468},
    {'N': 108, 'muls': 456}, {'N': 112, 'muls': 396}, {'N': 120, 'muls': 380},
    {'N': 128, 'muls': 560}, {'N': 144, 'muls': 436}, {'N': 160, 'muls': 680},
    {'N': 168, 'muls': 580}, {'N': 180, 'muls': 680}, {'N': 192, 'muls': 752},
    {'N': 204, 'muls': 976}, {'N': 216, 'muls': 1020}, {'N': 224, 'muls': 1016},
    {'N': 240, 'muls': 940}, {'N': 252, 'muls': 1024}, {'N': 256, 'muls': 1308},
    {'N': 288, 'muls': 1160}, {'N': 312, 'muls': 1608}, {'N': 336, 'muls': 1412},
    {'N': 360, 'muls': 1540}, {'N': 420, 'muls': 2080}, {'N': 480, 'muls': 2360},
    {'N': 504, 'muls': 2300}, {'N': 512, 'muls': 3180}, {'N': 560, 'muls': 3100},
    {'N': 672, 'muls': 3496}, {'N': 720, 'muls': 3620}, {'N': 784, 'muls': 4412},
    {'N': 840, 'muls': 4580}
]

def calculate_step1_func(L, N, M):
    """
    Calculate the function in step one: (N / L) 3 (L + M - 1)[log_2 ( L + M - 1) + 1]
    """
    if L <= 0:
        return float('inf')
    return (N / L) * 3 * (L + M - 1) * (np.log2(L + M - 1) + 1)

def find_L0(N, M):
    """
    Step 1: Find L0 that minimizes the function in step one
    note that we return the rounded value in order to get the integer value of L0
    """
    result = minimize_scalar(
        lambda L: calculate_step1_func(L, N, M),
        bounds=(1, N),
        method='bounded'
    )

    L0 = int(round(result.x))
    return L0

def calculate_real_multiplications(L, P, N):
    """
    Calculate the number of real multiplications using the formula:
    2S x [(3P / 2) log_2(P)] + 3SP, where S approx N / L
    The formula can be found in lecture note "ADSP_12.pdf", p.444
    """
    S = math.ceil(N / L)
    return 2 * S * ((3 * P / 2) * np.log2(P)) + 3 * S * P

def find_nearest_P_values(P0, Pvalues_N, num_neighbors = 4):
    """
    Find the nearest P values from the table around P0
    """
    # Get all N values from the table
    N_values = [entry['N'] for entry in Pvalues_N]

    # Find the nearest indices
    N_array = np.array(N_values)
    idx = np.searchsorted(N_array, P0)

    # Get range of indices around the insertion point
    start_idx = max(0, idx - num_neighbors)
    end_idx = min(len(N_values), idx + num_neighbors + 1)

    return N_values[start_idx:end_idx]

def main():
    # Given parameters
    N = 63  # length(x[n])
    M = 35  # length(h[n])

    # Step 1: Find L0
    L0 = find_L0(N, M)
    print(f"\nStep 1:")
    print(f"L0 = {L0}")

    # Step 2: Estimate P0 and find nearest values from the table
    P0 = L0 + M - 1
    P_values = find_nearest_P_values(P0, Pvalues_N)
    print(f"\nStep 2:")
    print(f"P0 = L0 + M - 1 = {P0}")
    print(f"Testing P values from table: {P_values}")

    # Step 3: Calculate L, S, and number of real multiplications for each P
    print(f"\nStep 3:")
    print("P\tL\tS\tReal Multiplications\tTable Multiplications")
    print("-" * 70)

    min_muls = float('inf')
    optimal_P = None
    optimal_L = None
    optimal_S = None

    results = []
    for P in P_values:
        # L must be less than or equal to P-M+1
        L = min(L0, P - M + 1)
        S = math.ceil(N / L)
        muls = calculate_real_multiplications(L, P, N)

        # Find the corresponding table value
        table_muls = next((entry['muls'] for entry in Pvalues_N if entry['N'] == P), None)

        results.append((P, L, S, muls, table_muls))
        print(f"{P}\t{L}\t{S}\t{muls:.0f}\t\t{table_muls if table_muls else 'N/A'}")

        if muls < min_muls:
            min_muls = muls
            optimal_P = P
            optimal_L = L
            optimal_S = S

    print(f"\nOptimal values:")
    print(f"P = {optimal_P}")
    print(f"L = {optimal_L}")
    print(f"S = {optimal_S}")
    print(f"Calculated number of real multiplications = {min_muls:.0f}")

    # Find the corresponding table value for optimal P
    optimal_table_muls = next((entry['muls'] for entry in Pvalues_N if entry['N'] == optimal_P), None)
    if optimal_table_muls:
        print(f"Table value for N={optimal_P}: {optimal_table_muls}")

if __name__ == "__main__":
    main()
```

and by running the code, we can get the following result:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW4 git:(main)±2 (0.548s)
python3 7.py

Step 1:
L0 = 63

Step 2:
P0 = L0 + M - 1 = 97
Testing P values from table: [84, 88, 90, 96, 104, 108, 112, 120, 128]

Step 3:
P       L       S       Real Multiplications    Table Multiplications
----------------------------------------------------------------------
84      50      2       3726            248
88      54      2       3939            364
90      56      2       4046            340
96      62      2       4369            280
104     63      1       2403            468
108     63      1       2513            456
112     63      1       2623            396
120     63      1       2846            380
128     63      1       3072            560

Optimal values:
P = 104
L = 63
S = 1
Calculated number of real multiplications = 2403
Table value for N=104: 468
```

Therefore, we can see that the optimal number of points for the DFT is $P = 104$.

(Since the image might be not that clear, if the original code file is needed, please inform me.)

(8)

We're required to determine the number of real multiplications for the following x-point DFTs:

Our approach is first decomposing the given x-points into factors, then use the table in lecture note: "ADSP_Write5.pdf", p.377-379, to find the corresponding value of MUL_x, except the subproblem (c), a little modification is needed to deal with $121 = 11^2$.

(a)

For 154 points DFT:

$$154 = 11 \times 14$$

So we have:

$$
\begin{aligned}
& 11 \times \mathrm{MUL}_{14} + 14 \times \mathrm{MUL}_{11} \\
= & 11 \times 32 + 14 \times 40 \\
= & 912
\end{aligned}
$$

(b)

For 165 points DFT:

$$165 = 11 \times 15$$

So we have:

$$
\begin{aligned}
& 11 \times \mathrm{MUL}_{15} + 15 \times \mathrm{MUL}_{11} \\
= & 11 \times 40 + 15 \times 40 \\
= & 1040
\end{aligned}
$$

(c)

For 242 points DFT:

$$242 = 2 \times 11^2$$

So we have:

$$
\begin{aligned}
& 2 \times \mathrm{MUL}_{121} + 121 \times \mathrm{MUL}_{2} \\
= & 2 \times \mathrm{MUL}_{121} + 121 \times 0 \\
= & 2 \times (11 \times \mathrm{MUL}_{11} + 11 \times \mathrm{MUL}_{11} + 3 \times 10 \times 10) \\
= & 2 \times (11 \times 40 + 11 \times 40 + 300) \\
= & 2 \times 1180 \\
= & 2360
\end{aligned}
$$