**Step 1)** load the colored image

In [3]:
```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, img_as_float

# Load the image
image = io.imread('Bhpd8.jpg')
image = img_as_float(image)  # Convert image to float

# Display the original image
plt.imshow(image)
plt.title("Original Image")
plt.axis('off')
plt.show()
```

Original Image

**step 2)** Perform SVD on each color channel

In [4]:
```python
def svd_decompose(image_channel):
    U, S, Vt = np.linalg.svd(image_channel, full_matrices=False)
    return U, S, Vt

R, G, B = image[:, :, 0], image[:, :, 1], image[:, :, 2]
print("Performing SVD on the Red channel...")
U_R, S_R, Vt_R = svd_decompose(R)
print("Done.")

print("Performing SVD on the Green channel...")
U_G, S_G, Vt_G = svd_decompose(G)
print("Done.")

print("Performing SVD on the Blue channel...")
U_B, S_B, Vt_B = svd_decompose(B)
print("Done.")
```

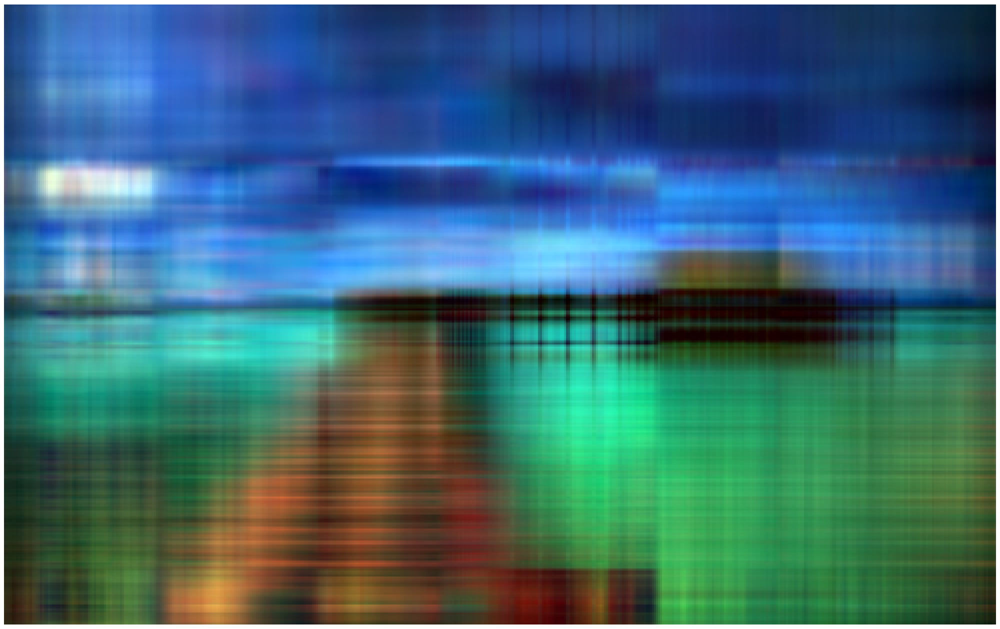
```
Performing SVD on the Red channel...
Done.
Performing SVD on the Green channel...
Done.
Performing SVD on the Blue channel...
Done.
```

**step 3)** Reconstruct the Image Using a Limited Number of Singular Values

```
In [5]: def svd_reconstruct(U, S, Vt, k):
            return np.dot(U[:, :k], np.dot(np.diag(S[:k]), Vt[:k, :]))

        def reconstruct_image(U_R, S_R, Vt_R, U_G, S_G, Vt_G, U_B, S_B, Vt_B, k):
            R_reconstructed = svd_reconstruct(U_R, S_R, Vt_R, k)
            G_reconstructed = svd_reconstruct(U_G, S_G, Vt_G, k)
            B_reconstructed = svd_reconstruct(U_B, S_B, Vt_B, k)
            return np.stack([R_reconstructed, G_reconstructed, B_reconstructed], axis=2)

        k_values = [5, 20, 50, 100, 150]
        for k in k_values:
            reconstructed_image = reconstruct_image(U_R, S_R, Vt_R, U_G, S_G, Vt_G, U_B, S_B, Vt
            plt.imshow(reconstructed_image)
            plt.title(f"Reconstructed Image with {k} Singular Values")
            plt.axis('off')
            plt.show()
```

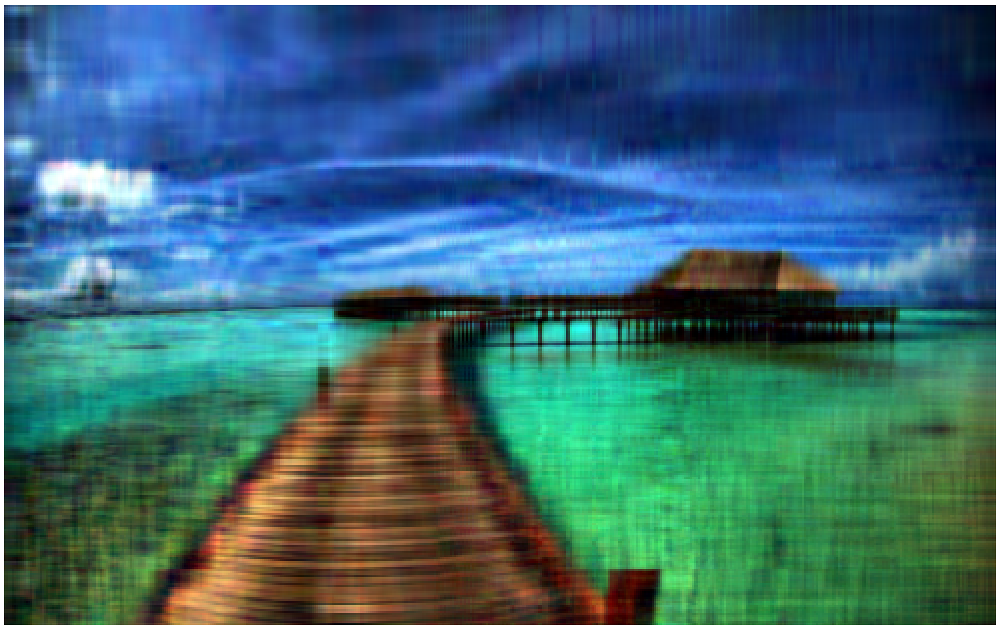
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Reconstructed Image with 5 Singular Values



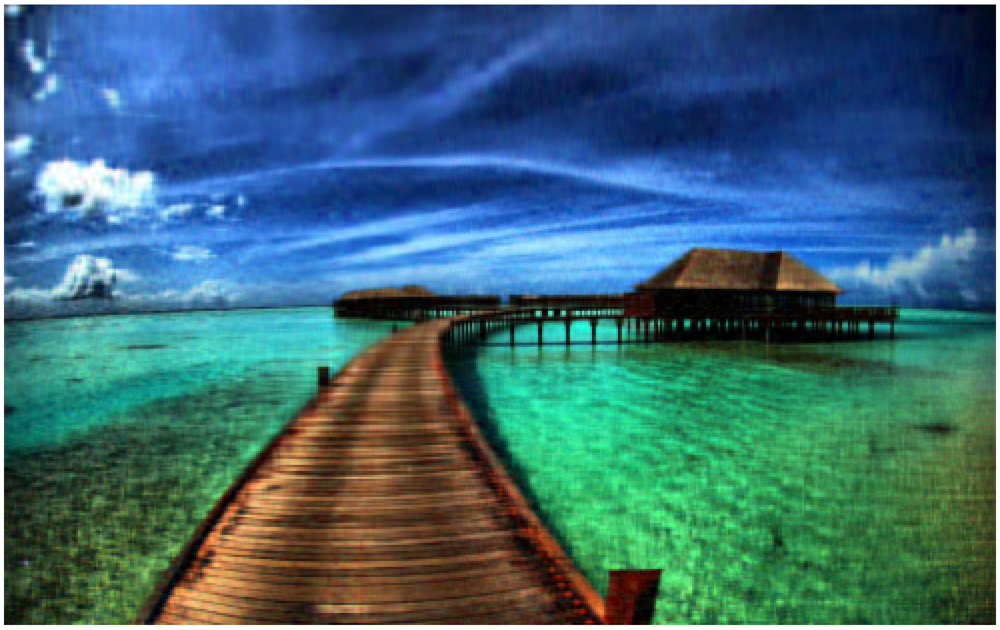
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Reconstructed Image with 20 Singular Values



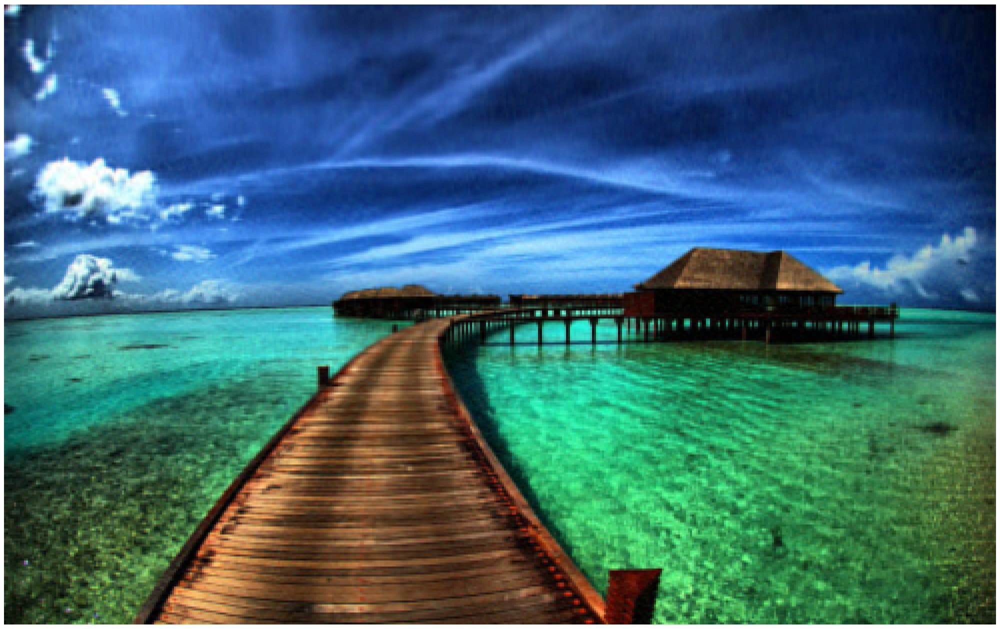
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Reconstructed Image with 50 Singular Values



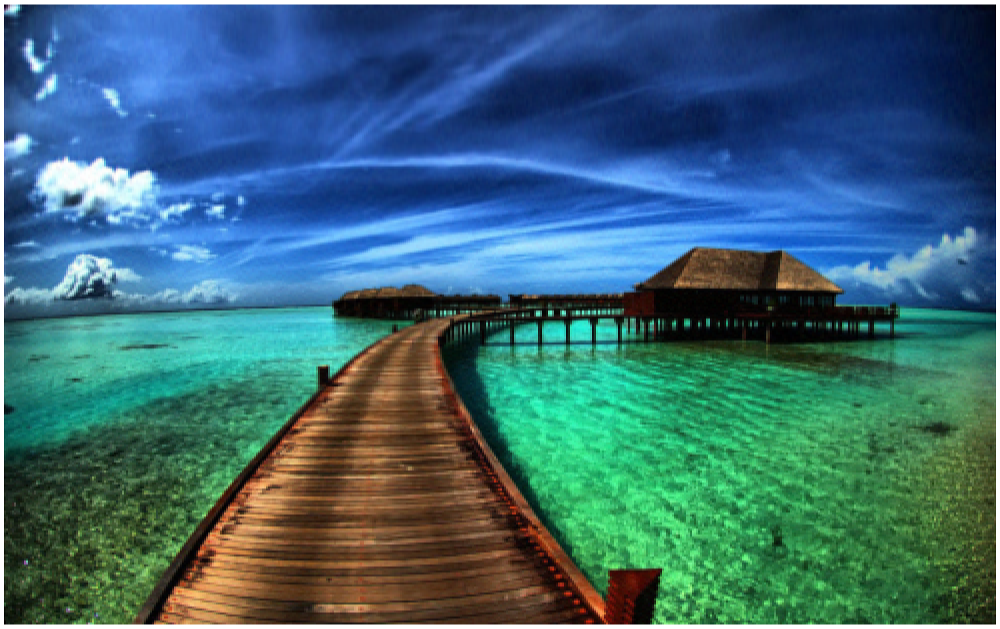
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Reconstructed Image with 100 Singular Values



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Reconstructed Image with 150 Singular Values



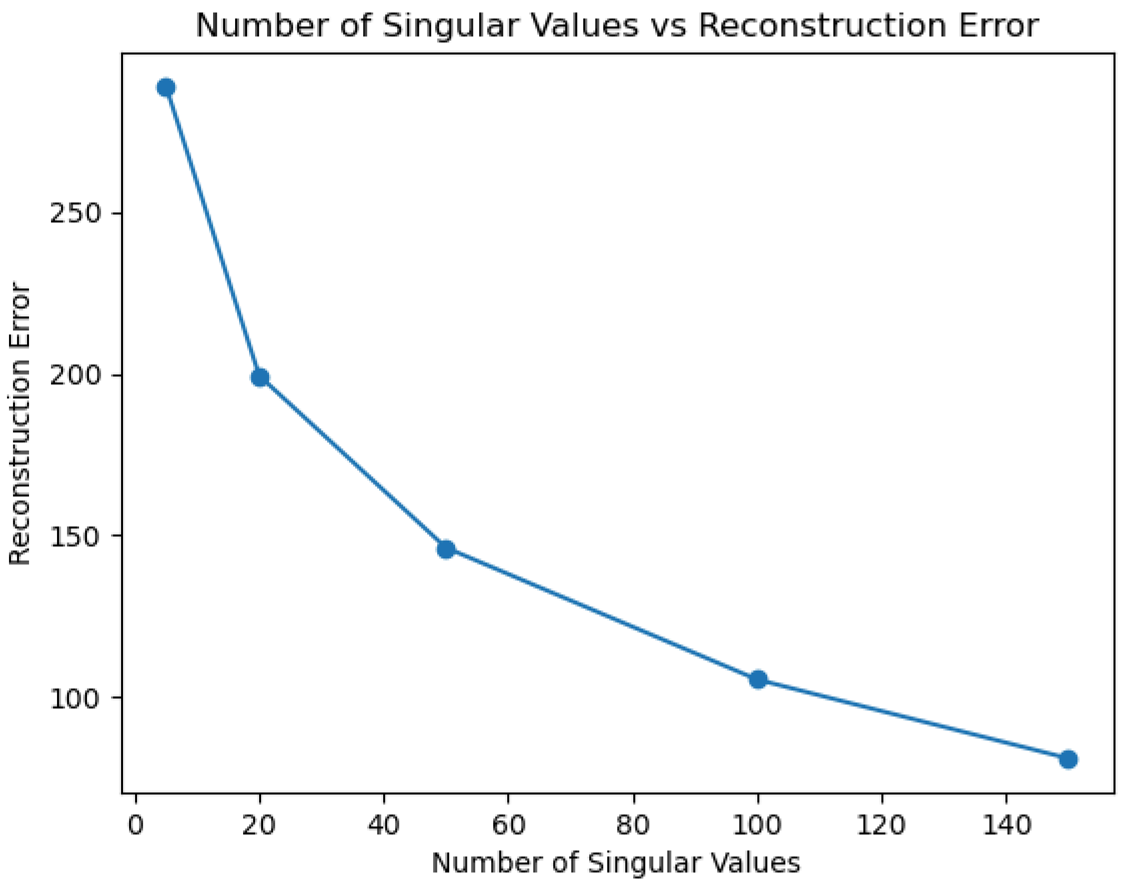
**step 4)** Calculate the Reconstruction Error and Compression Ratio

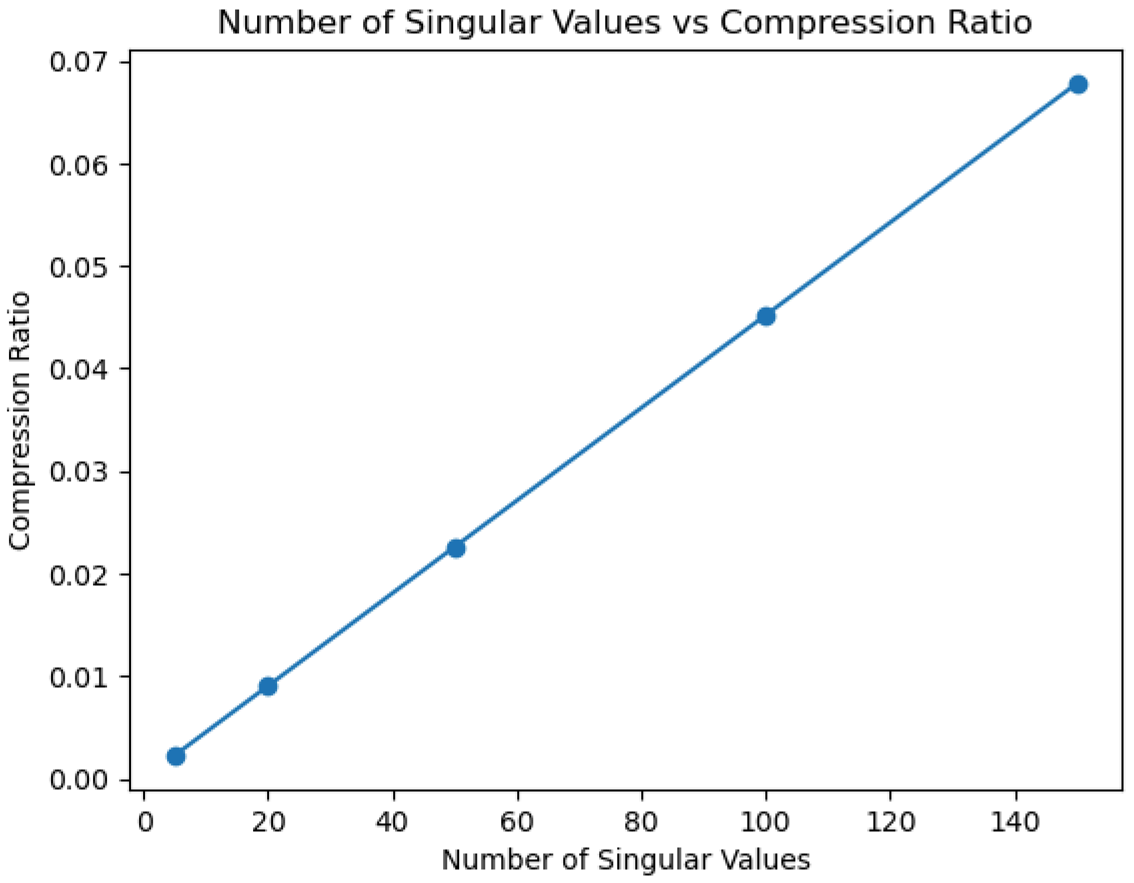
In [6]:
```
def calculate_error(original, reconstructed):
    return np.linalg.norm(original - reconstructed)

def calculate_compression_ratio(original_shape, k):
    return (k * (1 + sum(original_shape))) / np.prod(original_shape)

errors = []
compression_ratios = []
for k in k_values:
    reconstructed_image = reconstruct_image(U_R, S_R, Vt_R, U_G, S_G, Vt_G, U_B, S_B, Vt
    error = calculate_error(image, reconstructed_image)
    compression_ratio = calculate_compression_ratio(image.shape, k)
    errors.append(error)
    compression_ratios.append(compression_ratio)
    print(f"For {k} singular values: Error = {error}, Compression Ratio = {compression_r
```

```
For 5 singular values: Error = 288.7966971261218, Compression Ratio = 0.00225983796296296
3
For 20 singular values: Error = 199.31827526220144, Compression Ratio = 0.009039351851851
852
For 50 singular values: Error = 146.17753910075186, Compression Ratio = 0.022598379629629
628
For 100 singular values: Error = 105.41739003364806, Compression Ratio = 0.04519675925925
9256
For 150 singular values: Error = 81.03532868895753, Compression Ratio = 0.067795138888888
88
```

**step 5)** Generate the Plots and Tables

In [7]:
```
# Plot number of eigenvalues vs error
plt.plot(k_values, errors, marker='o')
plt.title("Number of Singular Values vs Reconstruction Error")
plt.xlabel("Number of Singular Values")
plt.ylabel("Reconstruction Error")
plt.show()

# Plot number of eigenvalues vs compression ratio
plt.plot(k_values, compression_ratios, marker='o')
plt.title("Number of Singular Values vs Compression Ratio")
plt.xlabel("Number of Singular Values")
plt.ylabel("Compression Ratio")
plt.show()
```




Number of Singular Values vs Compression Ratio
Compression Ratio
Number of Singular Values
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0
20
40
60
80
100
120
140

```
In [ ]:
```