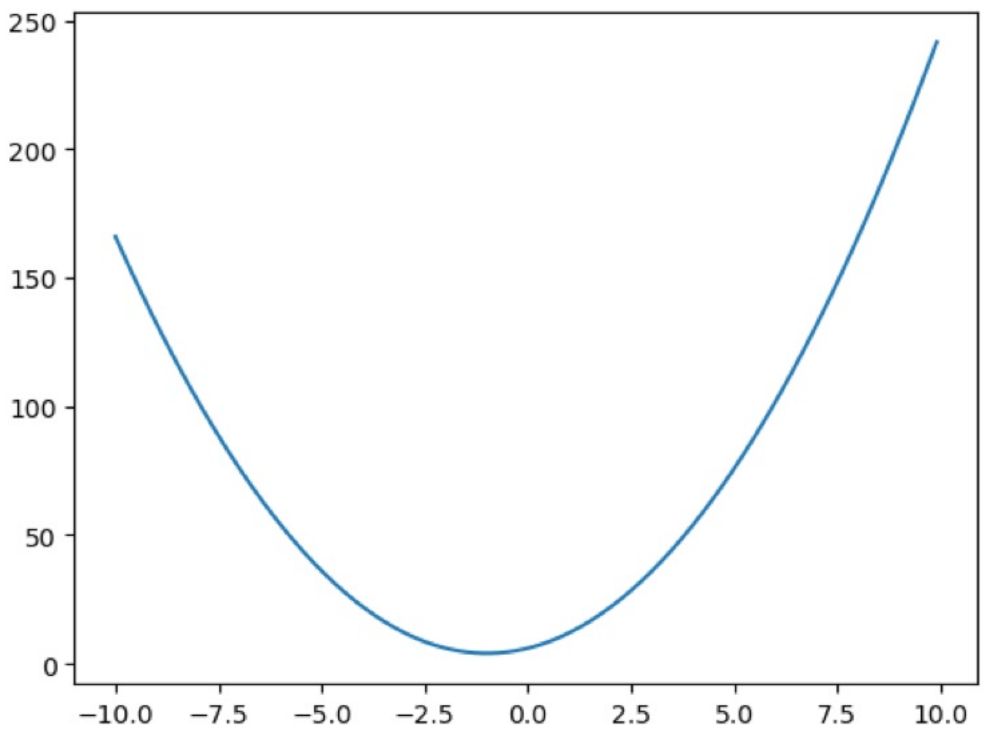
שאלה 1:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

Question 1.1

In [2]:
```
a = 6
b = 4
c = 2
def f(x):
    return a + b*x + c*(x**2)
x = np.arange(-10, 10, 0.1)
y = np.vectorize(f)(x)
plt.plot(x,y)
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x10870abd490>]
```



Question 1.2

In [3]:
```
def grad_f(x):
    return 2*c*(x) + b
```

Question 1.3

The critical point is: (-1, 4).

Question 1.4

In [4]:
```
def grad_update(grad_func, x, eta):
    return x - eta * grad_func(x)
```

Question 1.5:

In [5]:
```
def min_find(x_init, epsilon, eta):
    x_i = x_init
    x_values = []
    x_values.append(x_i)
    x_i1 = grad_update(grad_f, x_i, eta)
    while abs(x_i1 - x_i) > epsilon:
        x_i = x_i1
        x_values.append(x_i1)
        x_i1 = grad_update(grad_f, x_i, eta)
    return x_values

x_final = min_find(100, 0.001, 0.01)[-1]
print(f'({x_final}, {f(x_final)})')
```

```
(-0.9755852144560788, 4.001192163506311)
```

The x value is not the same as the one we found in question 1.3, but it is close. The reason for this being due to the epslion being not zero.

Question 1.6

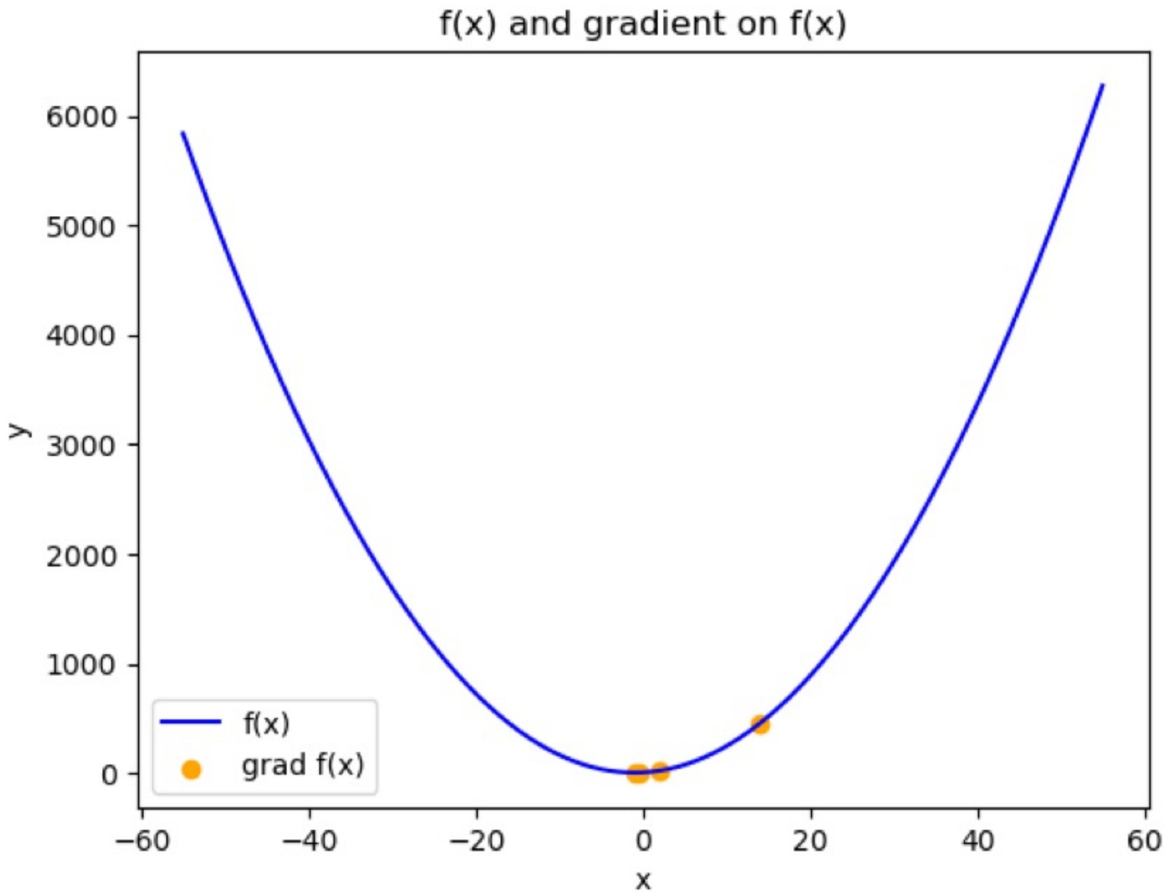
In [6]:
```
T_min = np.inf
T_min_x = None
T_min_parameters = None

# Loop over the range of epsilon, eta, and x_0
for epsilon in np.arange(0.1, 0.0, -0.01):
    for eta in np.arange(0.2, 0.0, -0.01):
        for x_0 in np.arange(50, 5, -1):
            ret = min_find(x_0, epsilon, eta)
            if len(ret) < T_min:
                T_min = len(ret)
                T_min_x = ret
                T_min_parameters = (x_0, epsilon, eta)

# Print the results
print(f'The algorithm is fastest with the following values:\n'
      f'x_0 = {T_min_parameters[0]}\n'
      f'epsilon = {T_min_parameters[1]}\n'
      f'eta = {T_min_parameters[2]:.2f}\n'
      f'With T = {T_min}')
```

```
The algorithm is fastest with the following values:
x_0 = 14
epsilon = 0.1
eta = 0.20
With T = 4
```

Question 1.7

In [7]:
```
x = np.arange(-55,55,0.01)
y = np.vectorize(f)(x)
plt.plot(x,y,color='blue',label='f(x)')
y_min = np.vectorize(f)(T_min_x)
plt.scatter(T_min_x, y_min,color='orange',
            label='grad f(x)')
plt.title('f(x) and gradient on f(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

Out[7]: `<matplotlib.legend.Legend at 0x10870b0ba90>`



שאלה 2:

1. ראשית נשים לב כי: $1-y_i(\langle w,x_i\rangle+b)$ הינה פו' לינארית
ולכן קמורה. $\Rightarrow$ $\max\{0,1-y_i\langle w,x_i\rangle+b\}$ קמורה
(מקס' של פו' קמורה היא קמורה גם כן).
$\Rightarrow \frac{1}{m}\sum_{i=1}^{m}\max\{0,1-y_i(\langle w,x_i\rangle+b)\}$ צירוף לינארי של פו' קמורה
והכפלה בקבוע אי-שלילי תשמור את התוצאה קמורה.
בנוסף $\lambda\|w\|^2$ - צירוף לינארי של קבוע אי-שלילי ונורמה בריבוע
(קמורה כיוון ש- $\|w\|$ קמורה ואי-שלילית מהגדרתה) הינו קמור.
$\Rightarrow$ $\underbrace{\frac{1}{m}\sum_{i=1}^{m}\max\{0,1-y_i(\langle w,x_i\rangle+b)\}+\lambda\|w\|^2}_{\text{צ"ל של פו' קמורות}}$ קמור.

2. נסמן $f(w)=l(w,y_i,x_i)$:
מתקיים:

$$\|f(w_1)-f(w_2)\|=\begin{cases}\|0-0\|\le R\|w_1-w_2\| & \forall R, f(w_1)=f(w_2)=0\\ \mathrm{II}\quad \|1-y_i\langle w_2,x_i\rangle\|\le R\|w_1-w_2\| & \text{בה"כ } f(w_1)=0\wedge f(w_2)>0\\ \mathrm{III}\quad \|1-y_i\langle w_1,x_i\rangle-(1-y_i\langle w_2,x_i\rangle)\|\le R\|w_1-w_2\| & f(w_1),f(w_2)>0\end{cases}$$

$\mathrm{II}\Rightarrow 1-y_i\langle w_1,x_i\rangle\le 0\Rightarrow 1\le y_i\langle w_1,x_i\rangle$

$\Rightarrow \|1-y_i\langle w_2,x_i\rangle\|\le\|y_i\langle w_1,x_i\rangle-y_i\langle w_2,x_i\rangle\|=$

$\|\langle w_1-w_2,y_ix_i\rangle\|\underset{\text{אי' קושי-שוורץ}}{\le}\|w_1-w_2\|\cdot\max_i\|y_ix_i\|=$

$=\|w_1-w_2\|\cdot\max_i\|x_i\|$

בנוסף מתקיים: $(1-y_i\langle w_2,x_i\rangle)>0$ ולכן עבור III:

$\|1-y_i\langle w_1,x_i\rangle-(1-y_i\langle w_2,x_i\rangle)\|=$

$=\|y_i\langle w_1,x_i\rangle-y_i\langle w_2,x_i\rangle\|=$

$=\|\langle w_1-w_2,y_ix_i\rangle\|\underset{\text{אי' קושי-שוורץ}}{\le}\|w_1-w_2\|\cdot\max_i\|y_ix_i\|=\|w_1-w_2\|\cdot\max_i\|x_i\|$

3. ראשית מתמטית, ה- subgradient הינו הנגזרת לפי המשתנה:

$$\text{subgradient by } w = \begin{cases} 2\lambda w & 1-y_i(\langle w,x_i\rangle+b)\leq 0 \\ -y_i x_i + 2\lambda w & 1-y_i(\langle w,x_i\rangle+b)>0 \end{cases}$$

$$\text{Subgradient by } b = \begin{cases} 0 & 1-y_i(\langle w,x_i\rangle+b)\leq 0 \\ -y_i & 1-y_i(\langle w,x_i\rangle+b)>0 \end{cases}$$

In [8]:
```
def subgradient(w, x, y, b, lam):
    if 1 - y * (w.dot(x) + b) <= 0:
        return (2 * lam * w,0)
    else:
        return (-y * x + 2 * lam * w, (-1) * y)
```

Question 2.4

In [46]:
```
def svm_with_sgd(X, y, lam=0, epochs=1000, l_rate=0.01,
                 sgd_type='practical'):
    np.random.seed(2)
    m, d = X.shape
    b = np.random.normal(0, 1, 1)
    w = np.random.normal(0, 1, d)

    if sgd_type == 'practical':
        for epoch in range(epochs):
            for i in np.random.permutation(m):
                subgrad_w, subgrad_b =
                subgradient(w, X[i], y[i], b, lam)
                w -= l_rate * subgrad_w
                b -= l_rate * subgrad_b
        return w, b
    if sgd_type == 'theory':
        ws, bs = [w.copy()], [b.copy()]
        # Use copy to avoid in-place modifications
        for _ in range(epochs * m):
            i = np.random.randint(m)
            subgrad = subgradient(w, X[i], y[i], b, lam)
            w -= l_rate * subgrad[0]
            ws.append(w.copy())
            b -= l_rate * subgrad[1]
            bs.append(b.copy())
        return np.mean(ws, axis=0), np.mean(bs, axis=0)
```

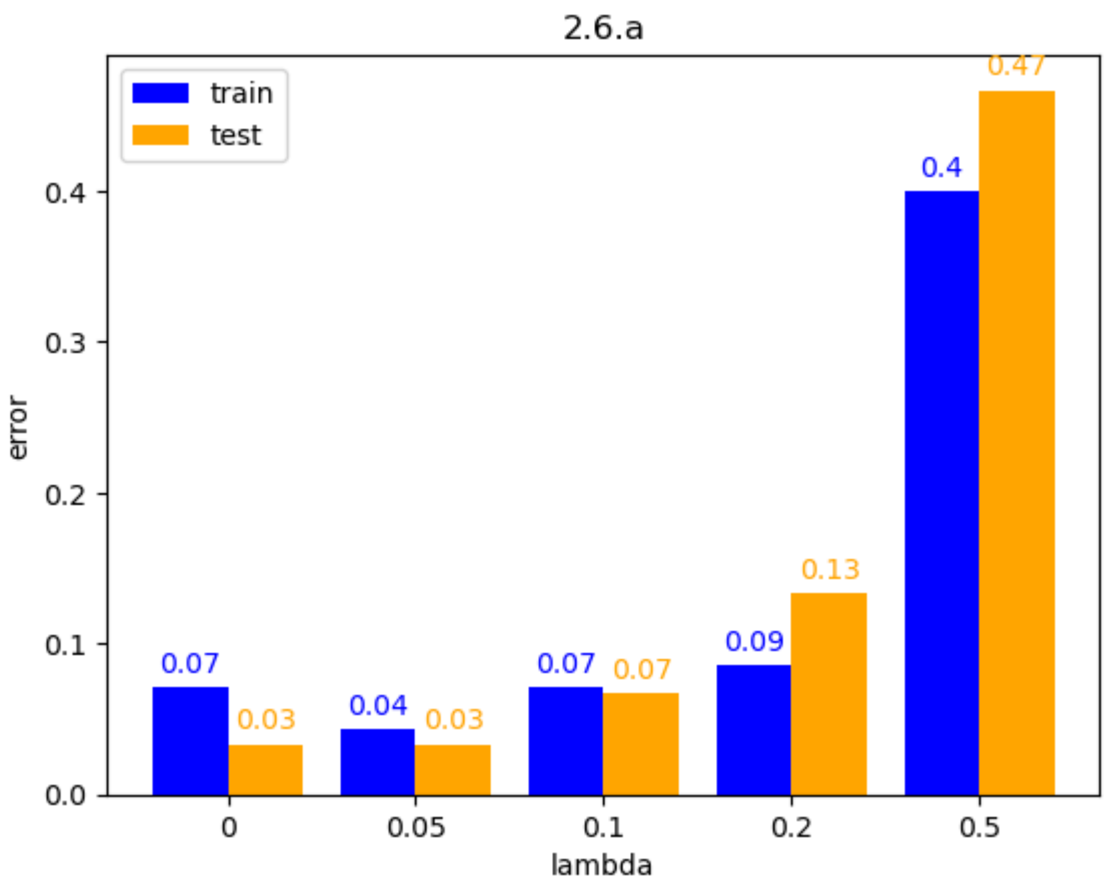
Question 2.5

In [38]:
```
def calculate_error(w, bias, X, y):
    pred = []
    M = X.shape[0]

    for i in range(X.shape[0]):
        if w.dot(X[i]) + bias > 0:
            pred.append(1)
        else:
            pred.append(-1)
    return np.sum(pred != y) / M
```

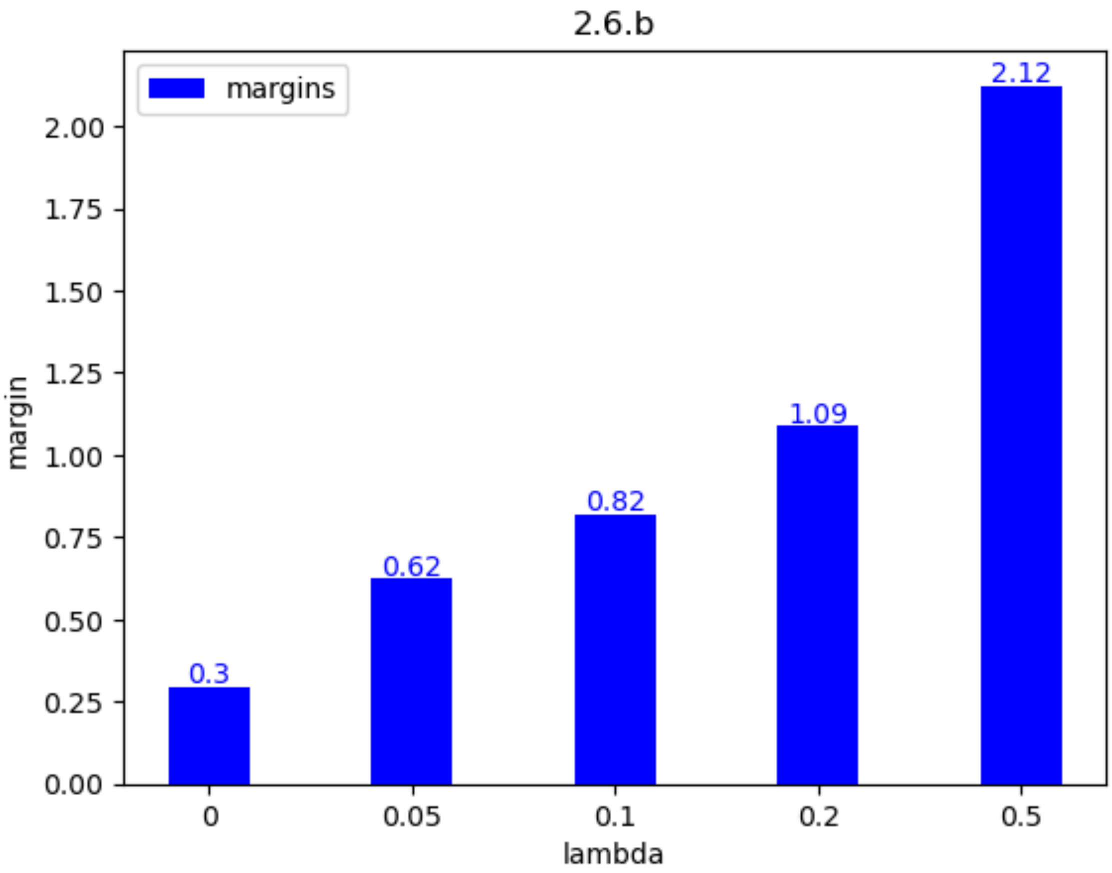
Question 2.6

In [39]:
```
# 2
X, y = load_iris(return_X_y=True)
X = X[y != 0]
y = y[y != 0]
y[y==2] = -1
X = X[:, 2:4]

# 3
X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  test_size=0.3, random_sta

# 4
lambdas = [0, 0.05, 0.1, 0.2, 0.5]
train_error_array = []
test_error_array = []
margins_array = []

for lam in lambdas:
    model = svm_with_sgd(X_train, y_train, lam)
    train_error = calculate_error(model[0],model[1],
                                  X_train,y_train)
    test_error = calculate_error(model[0],model[1],
                                 X_val,y_val)
    margin = 1 / np.linalg.norm(model[0])
    train_error_array.append(train_error)
    test_error_array.append(test_error)
    margins_array.append(margin)
```

In [40]:
```python
# 5
x = np.arange(len(lambdas))
width = 0.4

plt.bar(x, train_error_array, width=width,
        label='train', color='blue')
plt.bar(np.add(x, width), test_error_array,
        width=width, label='test', color='orange')

for i in range(len(lambdas)):
    plt.text(i, train_error_array[i] + 0.01,
             round(train_error_array[i], 2), ha='center', color='blue')
    plt.text(i + width, test_error_array[i] + 0.01,
             round(test_error_array[i], 2), ha='center', color='orange')

plt.xlabel('lambda')
plt.ylabel('error')
plt.title('2.6.a')
plt.xticks(np.add(x, width / 2), lambdas)
plt.legend()
plt.show()
```



In [41]:
```
x = np.arange(len(lambdas))
width = 0.4

plt.bar(x, margins_array, width=width, label='margins', color='blue')

for i in range(len(lambdas)):
    plt.text(i, margins_array[i] + 0.01,
             round(margins_array[i], 2), ha='center', color='blue')

plt.xlabel('lambda')
plt.ylabel('margin')
plt.title('2.6.b')
plt.xticks(x, lambdas)
plt.legend()
plt.show()
```



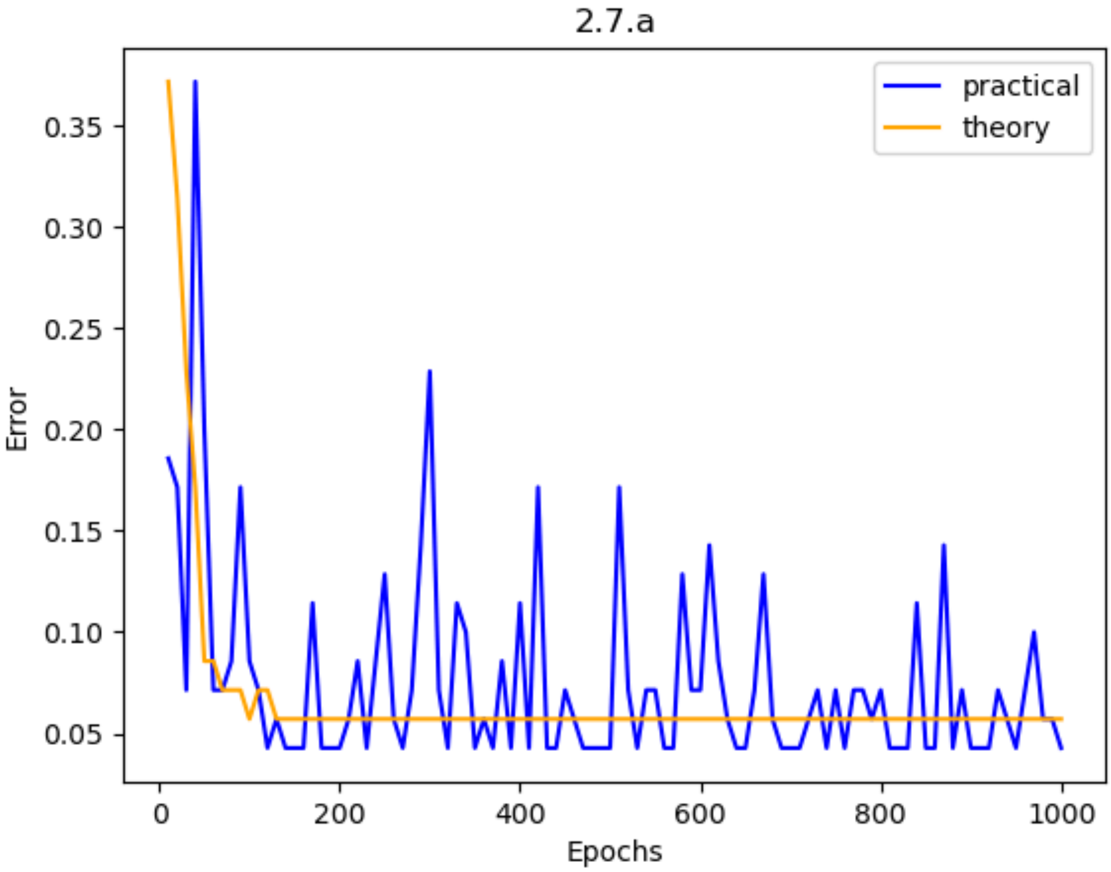
We choose lambda = 0.05 since it is the lambda with the lowest error rate for both the train and test sets.

Question 2.7

In [47]:
```
epochs = np.arange(10, 1001, 10) # including 1000

train_error_practical = []
test_error_practical = []
train_error_theory = []
test_error_theory = []

for epoch in epochs:
    # Practical SGD
    model = svm_with_sgd(X_train, y_train,
                         lam=0.05, epochs=epoch, sgd_type='practical')
    train_error = calculate_error(model[0],
                                  model[1], X_train, y_train)
    test_error = calculate_error(model[0],
                                 model[1], X_val, y_val)
    train_error_practical.append(train_error)
    test_error_practical.append(test_error)

    # Theoretical SGD
    model = svm_with_sgd(X_train, y_train,
                         lam=0.05, epochs=epoch, sgd_type='theory')
    train_error = calculate_error(model[0],
                                  model[1], X_train, y_train)
    test_error = calculate_error(model[0],
                                 model[1], X_val, y_val)
    train_error_theory.append(train_error)
    test_error_theory.append(test_error)
```

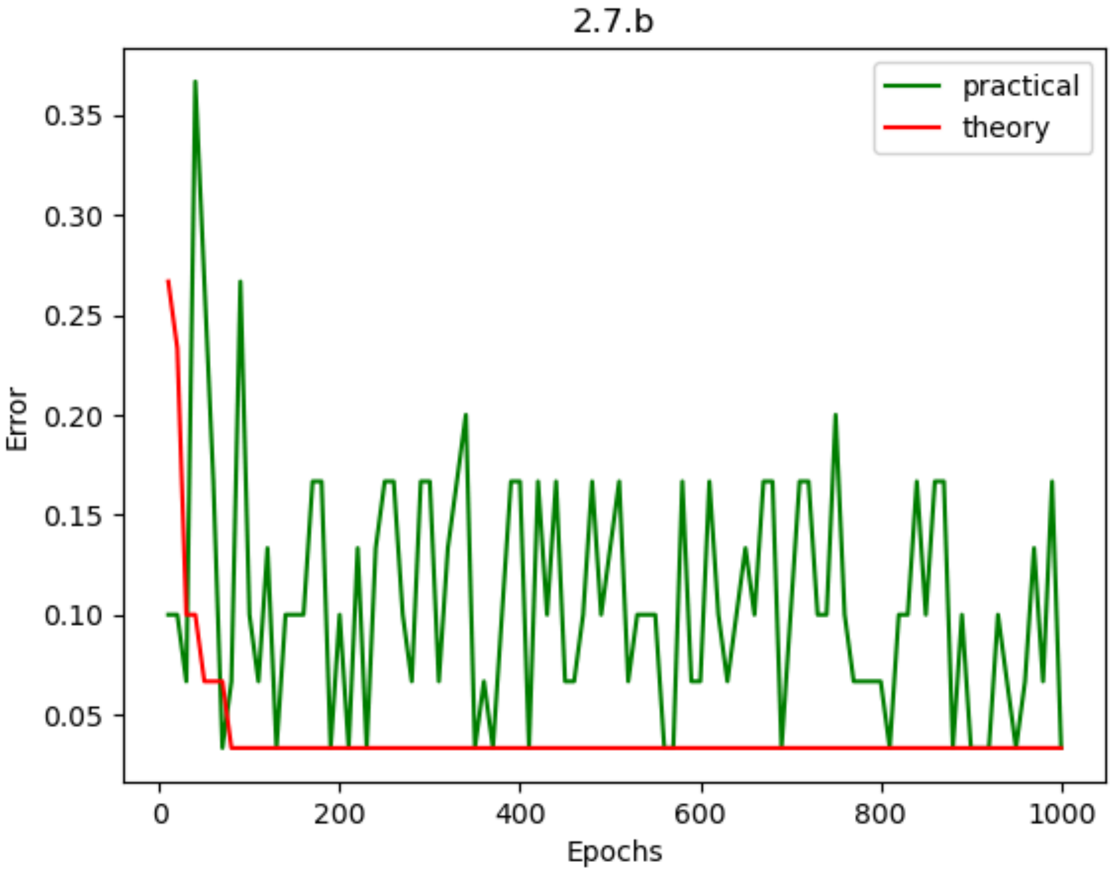
In [48]:
```
plt.plot(epochs, train_error_practical,
         label='practical', color='blue')
plt.plot(epochs, train_error_theory,
         label='theory', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.title('2.7.a')
plt.legend()
```

Out[48]: <matplotlib.legend.Legend at 0x10877805670>



In [49]:
```
plt.plot(epochs,test_error_practical,
         label='practical', color='g')
plt.plot(epochs,test_error_theory,
         label='theory', color='r')
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.title('2.7.b')
plt.legend()
```

Out[49]: <matplotlib.legend.Legend at 0x108793d0cd0>



Theoretical SGD assumes noise-free, ideal data, leading to stable errors after a certain number of epochs. In contrast, practical SGD deals with real, noisy data, so errors can continue to change with more epochs due to inherent data imperfections. This explains why practical errors fluctuate while theoretical errors stabilize.

Question 3.1

In [17]:
```
import numpy as np
```

In [18]:
```
def cross_validation_error(X, y, model, n_folds):
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    folds = np.array_split(indices, n_folds)

    train_errors = []
    validation_errors = []

    for fold in range(n_folds):
        val_indices = folds[fold]
        train_indices = np.concatenate([folds[i] for i in range(n_folds) if

        X_train, y_train = X[train_indices], y[train_indices]
        X_val, y_val = X[val_indices], y[val_indices]

        model.fit(X_train, y_train)
        train_preds = model.predict(X_train)
        val_preds = model.predict(X_val)

        train_errors.append(np.mean(train_preds != y_train))
        validation_errors.append(np.mean(val_preds != y_val))

    avg_train_error = np.mean(train_errors)
    avg_val_error = np.mean(validation_errors)

    return avg_train_error, avg_val_error
```

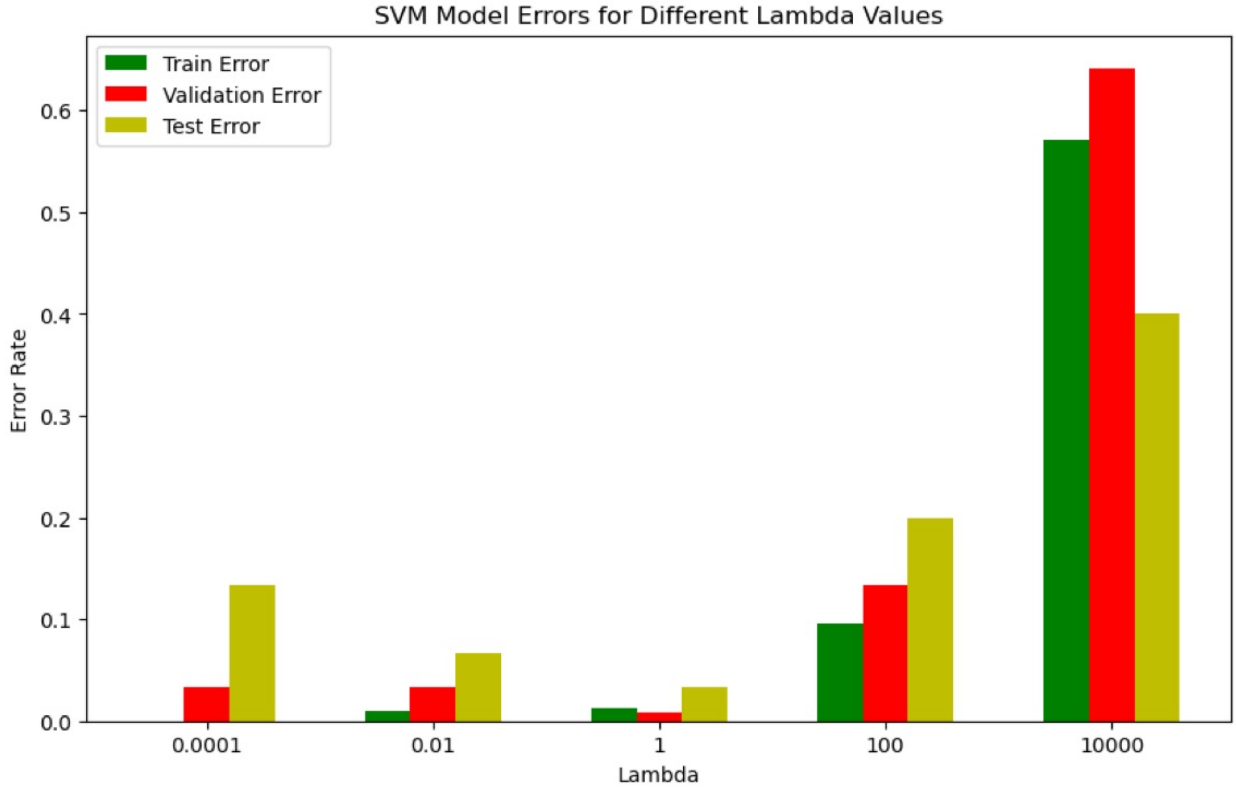
Question 3.2

In [19]:
```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

In [20]:
```
# Define lambda values
lambda_values = [10**(-4), 10**(-2), 1, 10**2, 10**4]

def svm_results(X_train, y_train, X_test, y_test):
    results = {}
    n_folds = 5

    for lam in lambda_values:
        c_value = 1 / lam
        model = SVC(kernel='linear', C=c_value)

        train_error, val_error =
        cross_validation_error(X_train, y_train, model, n_folds)

        # Compute test error
        test_preds = model.predict(X_test)
        test_error = np.mean(test_preds != y_test)

        results[f'SVM_lambda_{lam}'] =
        (train_error, val_error, test_error)

    return results
```

Question 3.3

In [21]:
```
# Load data
iris_data = load_iris()
X, y = iris_data['data'], iris_data['target']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_s

# Evaluate models
results = svm_results(X_train, y_train, X_test, y_test)

# Plot results
x = np.arange(len(lambda_values))
train_errors = [results[f'SVM_lambda_{lam}'][0] for lam in lambda_values]
val_errors = [results[f'SVM_lambda_{lam}'][1] for lam in lambda_values]
test_errors = [results[f'SVM_lambda_{lam}'][2] for lam in lambda_values]

plt.figure(figsize=(10, 6))
plt.bar(x - 0.2, train_errors, width=0.2,
        label='Train Error', color='g')
plt.bar(x, val_errors, width=0.2,
        label='Validation Error', color='r')
plt.bar(x + 0.2, test_errors, width=0.2,
        label='Test Error', color='y')
plt.xlabel('Lambda')
plt.ylabel('Error Rate')
plt.title('SVM Model Errors for Different Lambda Values')
plt.xticks(x, lambda_values)
plt.legend()
plt.show()
```



The best model for the CV is lambda = 0.0001. The best model for the test set is lambda = 1. The lambda which is best for CV is not the same as the one that is best for the test set, this is because for smaller lambdas there is a significant difference between the train error and the test error - this is caused by overfitting. Since CV is based on the training set, the model that works best for it will be with smaller lambdas. For a model which wishes to minimize the train error, lambda 1 is the most balanced between the smaller lambdas (explained above) and larger lambdas (which leads to underfitting).

שאלה 4:

$$g(w)+\langle u-w,\nabla g_j(w)\rangle = g_j(w)+\langle u-w,\nabla g_j(w)\rangle \le g_j(u) \le \max_i g_i(u) = g(u),\ \forall u\in\mathbb{R}^d$$

מהגדרת $g(w)=\max_i g_i(w)=g_j(w)$

לפי אי שוויון שטוען בתרגול
ובהרצאה עבור פו׳ קמורה

מהגדרת $g_j(w)$ בשאלה