1. Heat equation: $\frac{\partial u}{\partial t} = k \frac{\partial^2 u}{\partial x^2}$

a) backward Euler difference scheme

We derive:

$$u_t = (t_{n+1}, x_j) = \frac{u_j^{n+1} - u_j^n}{\Delta t}$$

$$u_{xx} = (t_{n+1}, x_j) = \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2}$$

Let $r = \Delta x$ and $h = \Delta t$. Now we will use taylor expansion to derive our method

$$u(t_n, x_j) = u(t_{n+1}, x_j) - h\, u_t(t_{n+1}, x_j) + \frac{h^2}{2} u_{tt}(t_{n+1}, x_j) + O(h^3)$$

$$u(t_{n+1}, x_{j+1}) = u(t_{n+1}, x_j) + r\, u_x(t_{n+1}, x_j) + \frac{r^2}{2!} u_{xx}(t_{n+1}, x_j) + \frac{r^3}{3!}(t_{n+1}, x_j) + \frac{r^4}{4!} u_{xxxx}(t_{n+1}, x_j) + O(r^5)$$

$u(t_{n+1}, x_j)$ = the centered part of our expansion

$$u(t_{n+1}, x_{j-1}) = u(t_{n+1}, x_j) - r u_x(t_{n+1}, x_j) + \frac{r^2}{2!} u_{xx}(t_{n+1}, x_j) - \frac{r^3}{3!} u_{xxx}(t_{n+1}, x_j) + O(r^4)$$

Local Truncation error is denoted by:

$$\tau_j^{n+1} = \frac{u(t_{n+1}, x_j) - u(t_n, x_j)}{h} - k \frac{u(t_{n+1}, x_{j+1}) - 2u(t_{n+1}, x_j) + u(t_{n+1}, x_{j-1})}{r^2}$$

$$\frac{u(t_{n+1}, x_j) - u(t_n, x_j)}{h} = \frac{u(t_{n+1}, x_j) - (u(t_{n+1}, x_j) - h u_t(t_{n+1}, x_j) + \frac{h^2}{2} u_{tt}(t_{n+1}, x_j) + O(h)^3}{h}$$

$$= \frac{u(t_{n+1}, x_j) - u(t_{n+1}, x_j) + h u_t(t_{n+1}, x_j) + \frac{h^2}{2} u_{tt}(t_{n+1}, x_j) - O(h)^3}{h}$$

$$= \frac{\cancel{h}[u_t(t_{n+1}, x_j) - \frac{h}{2} u_{tt}(t_{n+1}, x_j) - O(h)^3]}{\cancel{h}}$$

$$= u_t(t_{n+1}, x_j) - \frac{h^2}{2} u_{tt}(t_{n+1}, x_j) + O(h)^2$$

Next, we need to solve for the local truncation error

$$-k \frac{u(t_{n+1}, x_{j-1}) - 2u(t_{n+1}, x_j) + u(t_{n+1}, x_{j-1})}{}$$

$$= -k \frac{u(t_{n+1},x_j) + r u_x(t_{n+1},x_j) + \frac{r^2}{2}u_{xx}(t_{n+1},x_j) + \frac{r^3}{6}u_{xxx}(t_{n+1},x_j) + \frac{r^4}{24}u_{xxxx} + O(r^6) - 2u(t_{n+1},x_j)}{r^2}$$

$$+ \frac{u(t_{n+1},x_j) - r u_x(t_{n+1},x_j) + \frac{r^2}{2}(t_{n+1},x_j) - \frac{r^3}{6}u_{xxx}(t_{n+1},x_j) + \frac{r^4}{24}u_{xxxx}(t_{n+1},x_j) + O(r^6)}{}$$

$$= \frac{-kr^2[u_{xx}(t_{n+1},x_j) + \frac{r^4}{12}u_{xxxx}(t_{n+1},x_j) + O(r^5)]}{r^2}$$

$$= -k u_{xx}(t_{n+1},x_j) + \frac{r^4}{12} u_{xxxx}(t_{n+1},x_j) + O(r^5)$$

From LTE: $\frac{u(t_{n+1},x_j)-u(t_n,x_j)}{h} - k\frac{u(t_{n+1},x_{j+1}) - 2u(t_{n+1},x_j) + u(t_{n+1},x_{j-1})}{r^2}$

$$= u_t(t_{n+1},x_j) - k u_{xx}(t_{n+1},x_j) - \frac{h}{2}u_{tt}(t_{n+1},x_j) + O(h^2) - k\frac{r^2}{12}u_{xxxx}(t_{n+1},x_j) + O(r^3)$$

$$= 0 - \frac{h}{2}u_{tt}(t_{n+1},x_j) + O(h^2) - k\cdot\frac{r^2}{12}u_{xxxx}(t_{n+1},x_j) + O(r^3)$$

When we subtract our Taylor approx, we get

$$= \frac{h}{2}u_{tt}(t_{n+1},x_j) + O(h^2) - k\frac{r^2}{12}u_{xxxx}(t_{n+1},x_j) + O(r^3)$$

So, our local truncation error is

$$T_j^{n+1}(h,r) = \frac{h}{2!}u_{tt}(t_{n+1},x_j) + O(h^2) + \frac{r^2}{4!}u_{xxx}(t_{n+1},x_j) + O(r^3)$$

where $h=\Delta t$, $r=\Delta x$

The time $h$ is order 2 and space $r$ is order 3

From definition 14.2, we know that a PDE is consistent if $T_j^{n+1}(\Delta t,\Delta x) \to 0$ as $\Delta t,\Delta x \to 0$. We can clearly see

$$T_j^{n+1}(\Delta t,\Delta x) = O(\Delta t)^2 + O(\Delta x)^3$$

$$\lim_{\Delta t,\Delta x} T_j^{n+1}(\Delta t,\Delta x) = 0$$

Therefore, this is consistent

Now, we need to check for stability, let $u_m^n = \xi^n e^{ikjr}$

where $m = 1,2,\dots,M-1$ and $\lambda = \frac{hk}{r^2}$

For our scheme:

$$\frac{u_m^{n+1} - u_m^n}{h} = k\,\frac{u_{m+1}^{n+1} - 2u_m^{n+1} + u_{m-1}^{n+1}}{r^2}$$

We have: $\xi^{n+1} e^{ikmr} - \xi^n e^{ikmr} = \lambda \xi^{n+1} e^{ik(m+1)r}$

$$e^{ikmr}(\xi^{n+1} - \xi^n) = \lambda \xi^{n+1}(e^{ik(m+1)r} - 2e^{ikmr} + e^{ik(m-1)r})$$

$$= \lambda \xi^{n+1}(e^{ikmr}e^{kir} - 2e^{ikmr} + e^{ikmr}e^{-kir})$$

$$\cancel{e^{ikmr}}(\xi^{n+1} - \xi^n) = \lambda \xi^{n+1}\cancel{e^{ikmr}}(e^{kir} - e^{ikmr} + e^{-kir})$$

$$\xi^{n+1} - \xi^n = \xi^{n+1}\lambda(e^{kir} - 2 + e^{-kir})$$

$$\xi - 1 = \xi\lambda(e^{kir} - 2 + e^{-kir})$$

$$\Rightarrow \quad 1 = \xi - \lambda\xi(e^{kir} - 2 + e^{-kir})$$

$$= \xi(1 - \lambda\xi(e^{kir} - 2 + e^{-kir}))$$

$$= \xi(1 + 2\lambda(1 - \cos(kr)))$$

$$= \xi(1 + 2\lambda \sin^2(\tfrac{kr}{2}))$$

$$\Rightarrow \quad \xi = \frac{1}{(1 + 2\lambda \sin^2(\frac{kr}{2}))}$$

Since $\lambda > 0$ and $\sin(x) \in [0,1]$ when $x \in \mathbb{R}$ we can say that $\sin^2(x) \in [0,1]$

$$\Rightarrow \quad 1 + 2\lambda \sin^2(\tfrac{kr}{2}) \geq 1$$

Since our denominator is always positive and the numerator is one, we can say $\xi = \frac{1}{1 + 2\lambda \sin^2(\frac{kr}{2})} \leq 1$ and $0 < \xi \leq 1$

Therefore, this is stable for any value $h = \Delta t$, $r = \Delta x$

$\Rightarrow$ Unconditionally Stable with time order 2 and space order 3.

In [192]:
```
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy
```

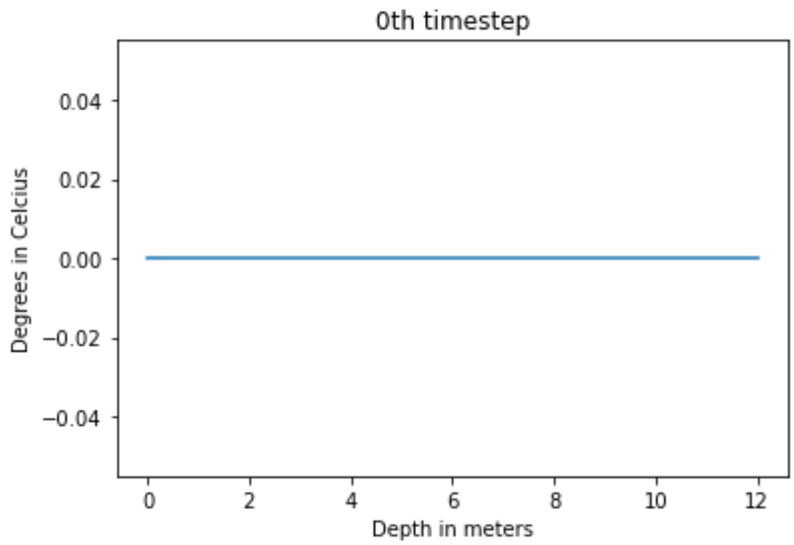
In [223]:
```
#1b

def tridiag(n, alpha):  #define our tri diagonal matrix, we implemented
    result = 2 * np.eye(n)

    for i in range(n - 1):
        result[i + 1][i] = -1
        result[i][i + 1] = -1

    return np.eye(n) + alpha * result
```

In [224]:
```
def backward_findiff(t0, uinit, alpha, delta_t, N, timesteps): #define
    t_curr = t0
    u_curr = uinit[1 : -1]
    M = len(uinit)
    A = tridiag(M - 2, alpha) #initialize our matrix to be used in equa
    results = []
    if 0 in timesteps:
        results.append(deepcopy(uinit))
    for i in range(1, N + 1): #this implements each time step, current
        t_curr += delta_t #adds delta t to create the next sep
        b = u_curr
        b[0] += alpha * u_t(t_curr)
        u_curr = np.linalg.solve(A, b) #uses a matrix solver to define
        if i in timesteps:
            result = np.concatenate(([u_t(t_curr)],u_curr,  [0.0]))
            results.append(result)
    return results
```

In [225]:
```
q1 = 0.71 #0.71 m is the initial condition
t0 = 0.0 #our initial time
K = 2e-3 * 1e-4 #convert from cm^2/s to m^2/s^2, our initial condition
Y = 3.15e7 #convert one year into seconds so our time steps match
A = 20.0 #define A as described in the problem

u_t = lambda t: A * np.sin(2 * np.pi * t / Y) #solve for the sinusoidal
uinit_calc = lambda t, xval: u_t(t) * np.exp(-q1 * xval) #this is used

xval = np.linspace(0, 12, 51) #define the x values
uinit = uinit_calc(t0, xval) #define our initial u value
delta_x = xval[1] - xval[0] #delta x is distance betweeen our consecuti
delta_t = Y / 250 #define the number of timesteps we want to observe
alpha = K * delta_t / (delta_x ** 2) #define our alpha value


timesteps = [i * 5 for i in range(101)] #20 time steps displayed with 5
u_approxs = backward_findiff(t0, uinit, alpha, delta_t, 1000, timesteps
```

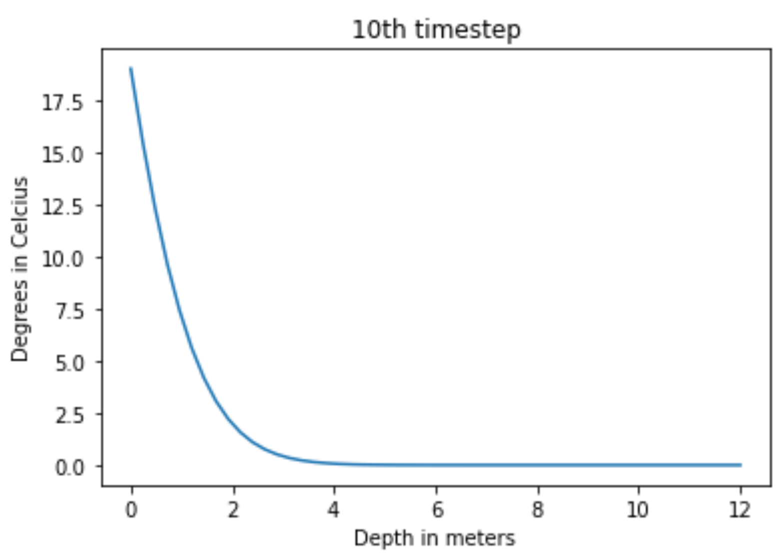
In [226]:
```
# Plotting 0 timestep
plt.plot(xval, u_approx[0])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("0th timestep")

```

Out[226]: Text(0.5, 1.0, '0th timestep')




In [227]:
```
# Plotting 10 timestep
plt.plot(xval, u_approxs[10])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("10th timestep")
```

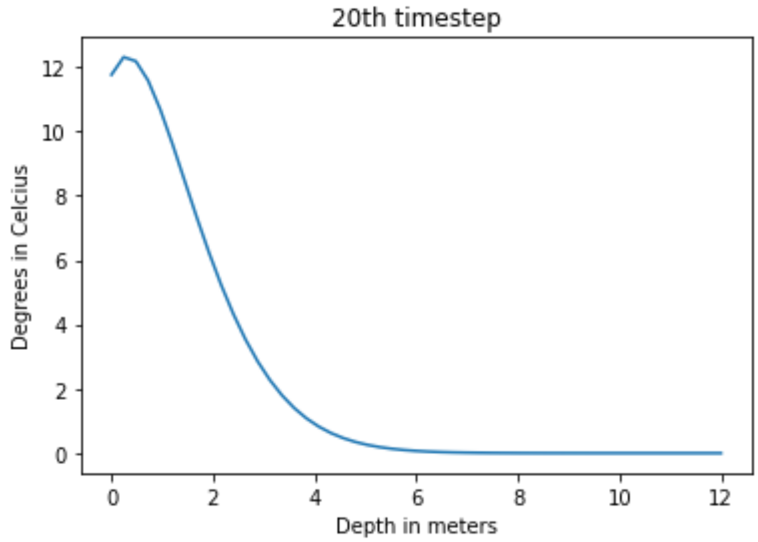
Out[227]: Text(0.5, 1.0, '10th timestep')



In [228]:

```
# Plotting 20 timestep
plt.plot(xval, u_approxs[20])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("20th timestep")
```

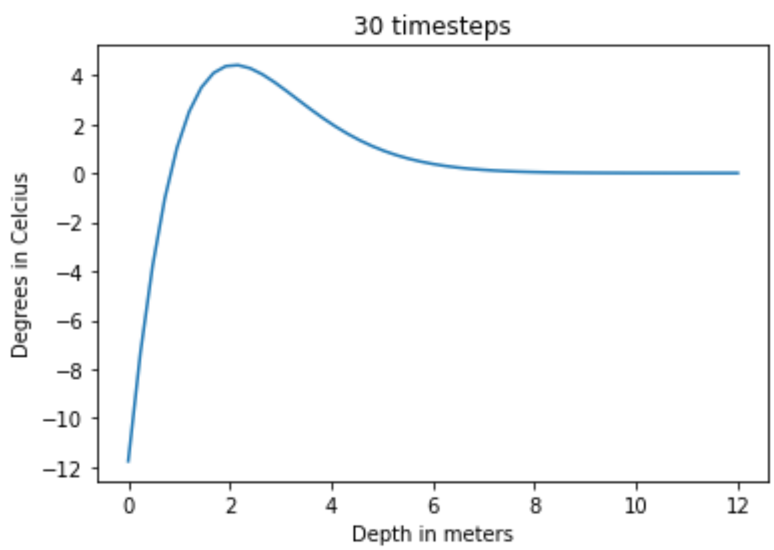
Out[228]: Text(0.5, 1.0, '20th timestep')




In [229]:

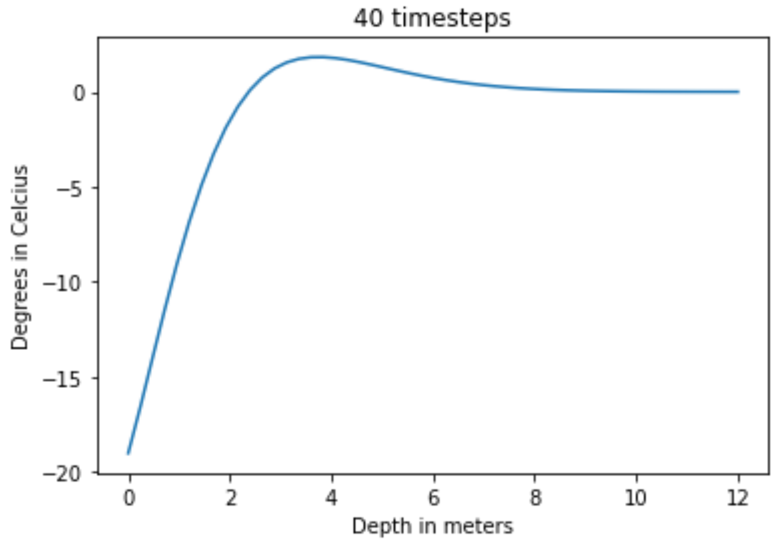
```
# Plotting 30 timestep
plt.plot(xval, u_approxs[30])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("30 timesteps")
```

Out[229]: Text(0.5, 1.0, '30 timesteps')



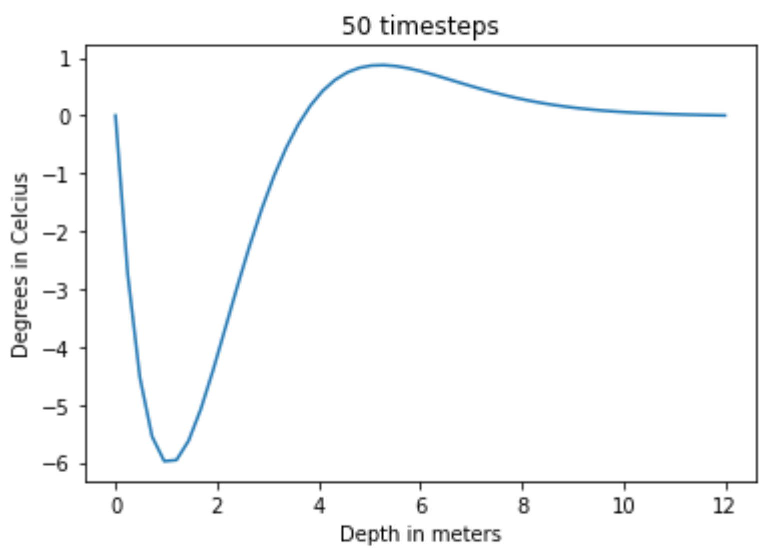
In [230]:
```
# Plotting 40 timestep
plt.plot(xval, u_approxs[40])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("40 timesteps")
```

Out[230]: Text(0.5, 1.0, '40 timesteps')




In [231]:
```
# Plotting 50 timestep
plt.plot(xval, u_approxs[50])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("50 timesteps")
```

Out[231]: Text(0.5, 1.0, '50 timesteps')

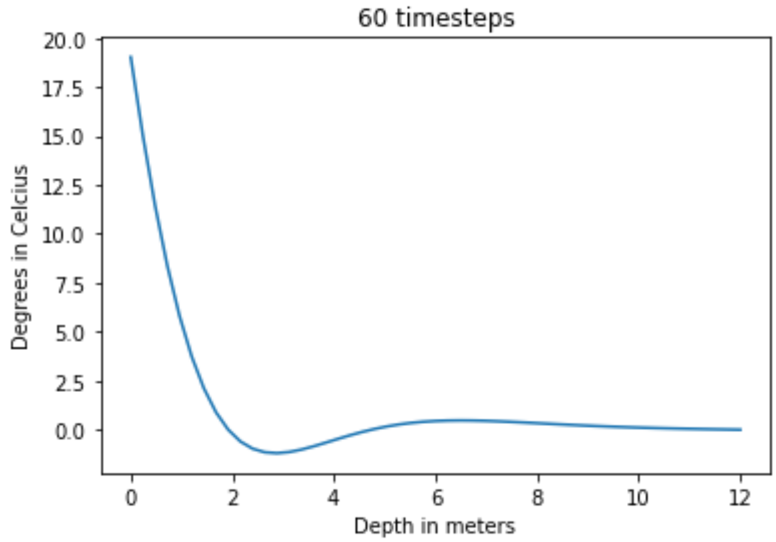


In [232]:

```
# Plotting 60 timestep
plt.plot(xval, u_approxs[60])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("60 timesteps")
```

Out[232]:

```
Text(0.5, 1.0, '60 timesteps')
```

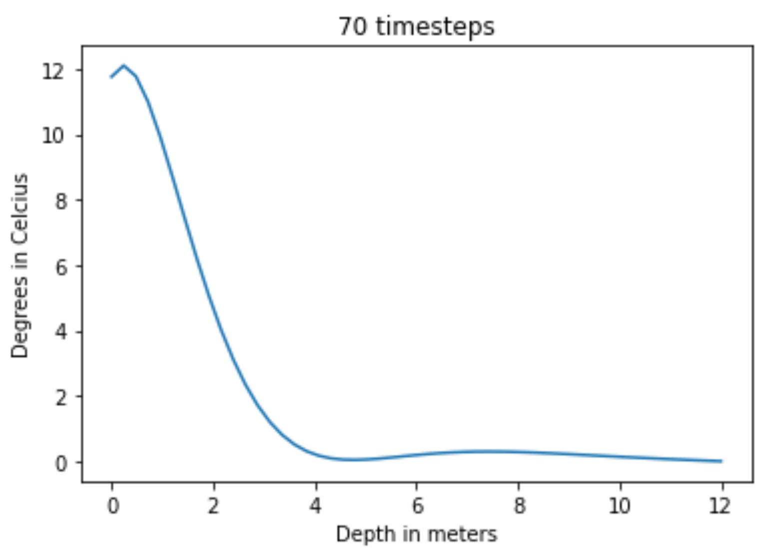



In [233]:

```
# Plotting 70 timestep
plt.plot(xval, u_approxs[70])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("70 timesteps")
```

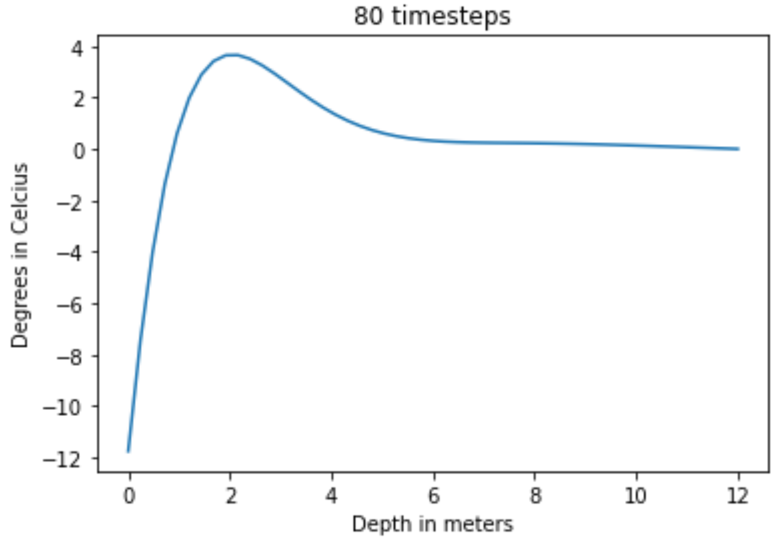
Out[233]:

```
Text(0.5, 1.0, '70 timesteps')
```



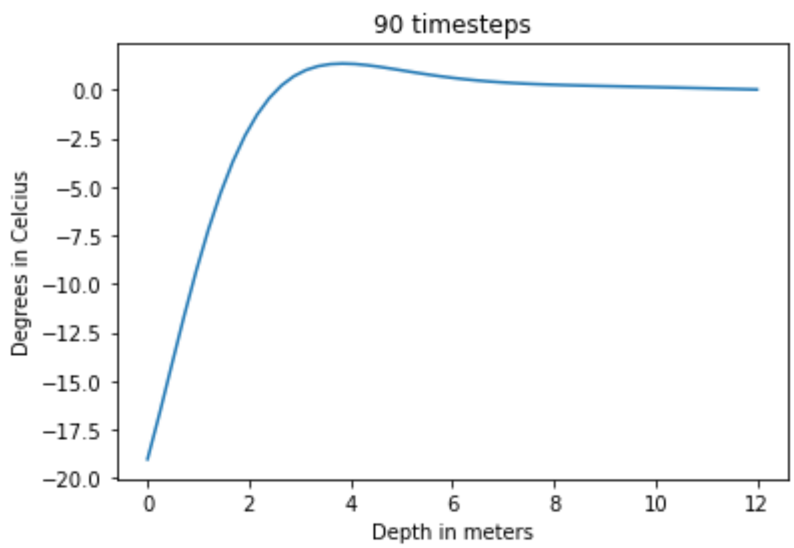
In [234]:
```
# Plotting 80 timestep
plt.plot(xval, u_approxs[80])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("80 timesteps")
```

Out[234]: Text(0.5, 1.0, '80 timesteps')




In [235]:
```
# Plotting 90 timestep
plt.plot(xval, u_approxs[90])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("90 timesteps")
```

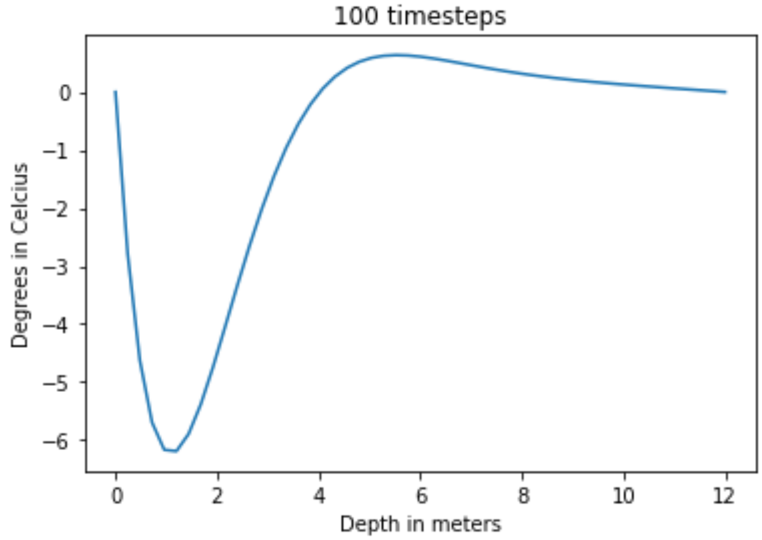
Out[235]: Text(0.5, 1.0, '90 timesteps')



In [236]:

```
# Plotting 100 timestep
plt.plot(xval, u_approxs[100])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("100 timesteps")
```

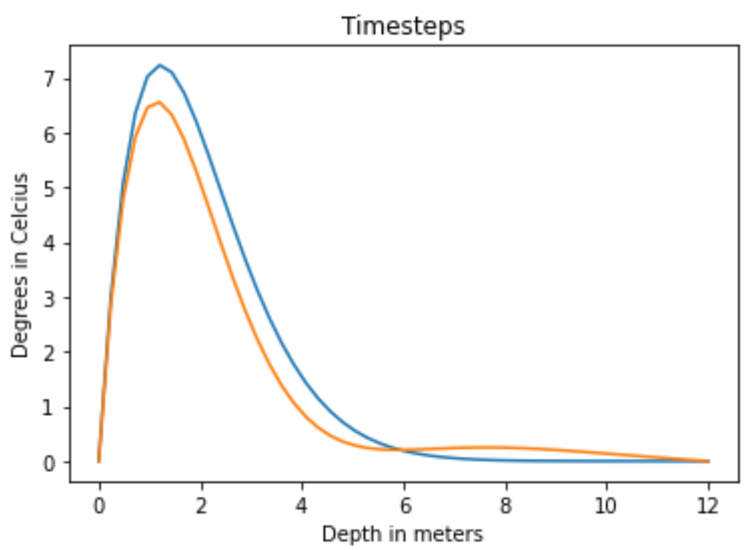
Out[236]: Text(0.5, 1.0, '100 timesteps')




In [237]:

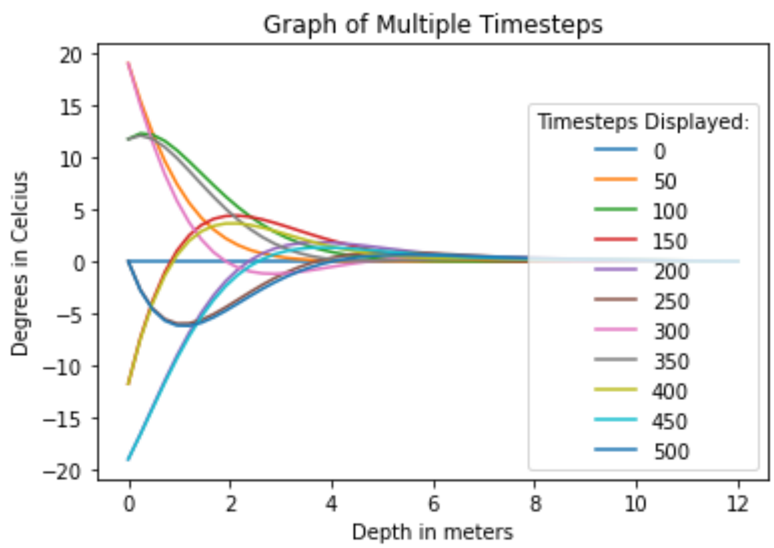
```
# Plotting 100 timestep
plt.plot(xval, u_approxs[25])
plt.plot(xval, u_approxs[75])
plt.xlabel("Depth in meters")
plt.ylabel("Degrees in Celcius")
plt.title("Timesteps")
```

Out[237]: Text(0.5, 1.0, 'Timesteps')



In [212]:

```
timestep_graphs = [0,10,20,30,40,50,60,70,80,90,100] #define our timest
timesteps_guide = [0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500]
for i in range(0, len(timestep_graphs)):
    plt.plot(xval, u_approx[timestep_graphs[i]])
    plt.legend((timesteps_guide), title = "Timesteps Displayed:", loc =
    plt.xlabel("Depth in meters")
    plt.ylabel("Degrees in Celcius")
    plt.title("Graph of Multiple Timesteps")
```




In [238]:

```
#c
print("The optimal for x* is when the temperature is the opposite of th
print("In our plot of multiple timesteps, we observe that temperature i
print("After our depth of 4m, we see our temperature begin to fluctuate
print("We see that the optimal x* value is approximately 3.7m for the m
print("This is optimal as you can cool wine during the hot summer and c
```

```
The optimal for x* is when the temperature is the opposite of the surface
temperature.
In our plot of multiple timesteps, we observe that temperature is opposit
e of the surface between 3 and 5.
After our depth of 4m, we see our temperature begin to fluctuate much les
s, this tends to zero.
We see that the optimal x* value is approximately 3.7m for the majority o
f time steps that we have plotted
This is optimal as you can cool wine during the hot summer and cultivate
vegetables during the cold winter
```

In [ ]:

```

```