In [1]:
```
import pandas as pd
import numpy as np
import math
import operator as op
from functools import reduce
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
def color_positive_red(val):
    """
    Takes a scalar and returns a string with
    the css property `'color: red'` for negative
    strings, black otherwise.
    """
    color = 'red' if val == 1 else 'black'
    return 'color: %s' % color
```

Below are the set of variable which have been provided as per the group work assignment

In [3]:
```
group_number = 30
current_price = 95
strike  = 90
#u = (1.1 + group_number/100)
#d = 1/u
n = 5
L = 130
```

Question 1 - American Call Option (N=5)

In [4]:
```
# Creating the function for Permuatation and combinations
def ncr(n, r):
    r = min(r, n-r)
    numer = reduce(op.mul, range(n, n-r, -1), 1)
    denom = reduce(op.mul, range(1, r+1), 1)
    return (numer // denom)
```

The below function is used to produce a **binomial lattice/tree for a fundamental security/stock** with a given price for a given parameters where:

- n is the total steps in the binomial tree
- time is the time frame over which we analyze the security
- current_price is the current price of the security/stock
- group_number is my group number

In [5]:
```
def construct_binomialtree(n, time, current_price, group_number):
    u = math.pow(((1.1 + (group_number/100))),time/n)
    d = 1/u
    binomial_tree = np.zeros((n+1,n+1))
    frequencies = np.zeros(n+1)
    for i in reversed(range(n+1)):
        for j in range(n+1):
            if((i == n) & (j == 0)):
                binomial_tree[i,j] = current_price
            elif((i+j) == n) :
                binomial_tree[i,j] = binomial_tree[i+1, j-1]* u
            elif((i+j)>n):
                binomial_tree[i,j] = binomial_tree[i,j-1] * d
        frequencies[i] = ncr(n,n-i)
    binomial_tree = binomial_tree.round(2)
    binomial_tree = pd.DataFrame(binomial_tree, index=reversed(range(n+1)))
    return(binomial_tree, frequencies)
```

Binomial Lattice of the stock price

In [6]:
```
stock_binomiallattice,_ = construct_binomialtree(n, 1, current_price, group_number)
stock_binomiallattice.columns = ['X0(w)','X1(w)','X2(w)','X3(w)','X4(w)','X5(w)']
#stock_binomiallattice.index = ['u,u,u','u,u,d','u,d']
stock_binomiallattice
```

Out[6]:

	X0(w)	X1(w)	X2(w)	X3(w)	X4(w)	X5(w)
5	0.0	0.00	0.00	0.00	0.00	133.00
4	0.0	0.00	0.00	0.00	124.34	116.25
3	0.0	0.00	0.00	116.25	108.69	101.61
2	0.0	0.00	108.69	101.61	95.00	88.82
1	0.0	101.61	95.00	88.82	83.04	77.63
0	95.0	88.82	83.04	77.63	72.58	67.86

Code below is a function will produce the **Lattice of an American/European call/put option** for a given stock price lattice, exercise price and rate of interest.

The parameter option_type of the function takes in 1 if it is a call option and -1 if it is a put option

The parameter exercise_type of the function takes in 1 if it is an European option and -1 if it is a American option
The parameter barrier of the function takes in 1 if it is a Barrier option and 0 if it is not a Barrier option

In [7]:
```
def option_price_lattice(current_price, group_number, strike, n, option_type, rate, time, exercise_type, barrier, barrier_price):
    stock_price_lattice,_ = construct_binomialtree(n, time, current_price, group_number)

    u = math.pow(((1.1 + (group_number/100))),1/n)
    d = 1/u

    binomial_tree = np.zeros(stock_price_lattice.shape)
    exercise_tree = np.array([max(0,option_type*(x-strike)) if (x>0) else 0 for x in np.ravel(stock_price_lattice)]).reshape((n+1,n+1))
    #exercise_tree = max(0,option_type*(stock_price_lattice.iloc[i,j]-strike)) for i in range(n+1) for j in range(n+1) if (i+j>=n)

    pu = (1-d)/(u-d)
    pd = 1 - pu
    t = time/n

    for j in reversed(range(n+1)):
        for i in reversed(range(n+1)):
            if(j == n):
                if barrier == 1: binomial_tree[i,j] = (stock_price_lattice.iloc[i,j] < L) * max(0,option_type*(stock_price_lattice.iloc[i,j]-strike)) * (
                else: binomial_tree[i,j] = max(0,option_type*(stock_price_lattice.iloc[i,j]-strike)) * (1/(1+rate*t))

            elif((i+j>=n)& (j!=n)) :
                #binomial_tree[i,j] = (binomial_tree[i-1, j+1]* pu + binomial_tree[i, j+1]* pd) * (1/(1+rate*t))
                binomial_tree[i,j] = (binomial_tree[i-1, j+1]* pu + binomial_tree[i, j+1]* pd) * (1/(1+rate*t)) if (exercise_type == 1) else max(exercise

    binomial_tree = np.array(binomial_tree.round(2))
    exercise_tree = np.array(exercise_tree.round(2))
    return (binomial_tree, exercise_tree)
```

In [8]:
```
calloptionlattice, callexercise_tree = option_price_lattice(current_price, group_number, strike, n = 5 , option_type= 1, rate = 0, time = 1, exercise_typ
pd.DataFrame(callexercise_tree)
```

Out[8]:

	0	1	2	3	4	5
0	0.0	0.00	0.00	0.00	0.00	43.00
1	0.0	0.00	0.00	0.00	34.34	26.25
2	0.0	0.00	0.00	26.25	18.69	11.61
3	0.0	0.00	18.69	11.61	5.00	0.00
4	0.0	11.61	5.00	0.00	0.00	0.00
5	5.0	0.00	0.00	0.00	0.00	0.00

1(a) - Find the value of the derivative at each node.

In [9]:
```
pd.DataFrame(calloptionlattice)
```

Out[9]:

	0	1	2	3	4	5
0	0.00	0.00	0.00	0.00	0.00	43.00
1	0.00	0.00	0.00	0.00	34.34	26.25
2	0.00	0.00	0.00	26.25	18.69	11.61
3	0.00	0.00	18.85	11.93	5.61	0.00
4	0.00	12.81	7.17	2.71	0.00	0.00
5	8.33	4.14	1.31	0.00	0.00	0.00

In [10]:
```
def early_exercise_chart(optionlattice,exercise_tree):
    early_exercise = np.zeros((n+1, n+1))
    for i in range(len(optionlattice)):
        for j in range(len(optionlattice)):
            #print(i,j)
            if(optionlattice[i,j] == exercise_tree[i,j] and exercise_tree[i,j]!= 0 and j!= len(optionlattice)-1):
                early_exercise[i,j] = 1
            else:
                early_exercise[i,j] = 0
    return pd.DataFrame(early_exercise.round(0))
```

1(b) - Is there any point time where we, as buyers of the option, benefit from early exercise?

The **red fonts** below indicate the point or nodes for early exercise

In [11]:
```
dfcall = early_exercise_chart(calloptionlattice, callexercise_tree)
dfcall.style.applymap(color_positive_red)
```

Out[11]:

	0	1	2	3	4	5
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
2	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Question 2 - American Put Option (N=5)

2. Consider the same parameters and setting as in (1), but now for pricing an American Put Option.

In [12]:
```
putoptionlattice, putexercise_tree = option_price_lattice(current_price, group_number, strike, n = 5 , option_type= -1, rate = 0, time = 1, exercise_type
pd.DataFrame(putexercise_tree)
```

Out[12]:

	0	1	2	3	4	5
0	0.0	0.00	0.00	0.00	0.00	0.00
1	0.0	0.00	0.00	0.00	0.00	0.00
2	0.0	0.00	0.00	0.00	0.00	0.00
3	0.0	0.00	0.00	0.00	0.00	1.18
4	0.0	0.00	0.00	1.18	6.96	12.37
5	0.0	1.18	6.96	12.37	17.42	22.14

2(a) - Find the value of the derivative at each node.

In [13]:
```
pd.DataFrame(putoptionlattice)
```

Out[13]:

	0	1	2	3	4	5
0	0.00	0.00	0.00	0.00	0.00	0.00
1	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.16	0.32	0.61	1.18
4	0.00	1.20	2.16	3.89	6.96	12.37
5	3.33	5.32	8.27	12.37	17.42	22.14

2(b) - Is there any point time where we, as buyers of the option, benefit from early exercise?

The **red fonts** below indicate the point or nodes for early exercise

In [14]:
```
dfput = early_exercise_chart(putoptionlattice, putexercise_tree)
dfput.style.applymap(color_positive_red)
```

Out[14]:

	0	1	2	3	4	5
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
5	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000

Question 3 - Barrier Call Option (N=3)

3(a) - Value of the European Up-and-Out (UAO) Call Option

In [15]:
```
euro_uao_stock_price_lattice, euro_uao_stock_price_freq = construct_binomialtree(n = 5, time = 1, current_price = 100, group_number = 30)
pd.DataFrame(euro_uao_stock_price_lattice)
```

Out[15]:

	0	1	2	3	4	5
5	0.0	0.00	0.00	0.00	0.00	140.00
4	0.0	0.00	0.00	0.00	130.89	122.37
3	0.0	0.00	0.00	122.37	114.41	106.96
2	0.0	0.00	114.41	106.96	100.00	93.49
1	0.0	106.96	100.00	93.49	87.41	81.72
0	100.0	93.49	87.41	81.72	76.40	71.43

In [16]:
```
european_uao_call_lattice, european_uao_exercise_tree = option_price_lattice(100, group_number, strike, n = 5, option_type = 1, rate = 0, time = 1, exerc
pd.DataFrame(european_uao_call_lattice)
```

Out[16]:

	0	1	2	3	4	5
0	0.00	0.00	0.00	0.00	0.00	0.00
1	0.00	0.00	0.00	0.00	16.73	32.37
2	0.00	0.00	0.00	20.70	24.41	16.96
3	0.00	0.00	18.77	16.96	10.00	3.49
4	0.00	14.83	11.14	5.70	1.69	0.00
5	10.79	7.03	3.18	0.81	0.00	0.00

3(b) - Which option is more expensive: the European call, or the UAO European call?

In [17]:
```
european_vanilla_call_lattice, european_vanilla_exercise_tree = option_price_lattice(100, group_number, strike, n = 5, option_type = 1, rate = 0, time =
pd.DataFrame(european_vanilla_call_lattice)
```

Out[17]:

	0	1	2	3	4	5
0	0.00	0.00	0.00	0.00	0.00	50.00
1	0.00	0.00	0.00	0.00	40.89	32.37
2	0.00	0.00	0.00	32.37	24.41	16.96
3	0.00	0.00	24.41	16.96	10.00	3.49
4	0.00	17.55	11.14	5.70	1.69	0.00
5	12.11	7.03	3.18	0.81	0.00	0.00

Clearly, the European call option is more expensive. This is because when a barrier is introduced to a vanilla European call, its pay-off reduces to zero when the stock price crosses this barrier. For a vanilla call whose value increases with an increase in the underlying's price, the introduction of a barrier prevents it from realising this value once it is breached. For this reason, a European Up-and-Out Call option will typically trade at a discount to the value of a plain Vanilla European Call option.

3(b) - Value of Up-and-In (UAI) European Call option

Being long a KO option and a KI option with the same features is equivalent to owning a comparable vanilla option independently from the behaviour of the spot with respect to the barrier level.

Knock-In (K,T,B) + Knock-Out (K,T,B) = Vanilla (K,T)

It is very easy to see that, for any given scenario of the underlying asset path before maturity, the portfolio (KI + KO) will always have the same payoff as the corresponding vanilla option. This relationship holds for both the put and call options in the absence of rebates.

Source: https://bookdown.org/maxime_debellefroid/MyBook/barrier-options.html

In [18]:
```
european_uai_call_lattice = pd.DataFrame(european_vanilla_call_lattice) - pd.DataFrame(european_uao_call_lattice)
european_uai_call_lattice
```

Out[18]:

	0	1	2	3	4	5
0	0.00	0.00	0.00	0.00	0.00	50.0
1	0.00	0.00	0.00	0.00	24.16	0.0
2	0.00	0.00	0.00	11.67	0.00	0.0
3	0.00	0.00	5.64	0.00	0.00	0.0
4	0.00	2.72	0.00	0.00	0.00	0.0
5	1.32	0.00	0.00	0.00	0.00	0.0

In [ ]:

In [ ]: