# FE545 Final Report

**Rui Zong**

## Question 1+2:

T =1
$S_0$ = 50
K = 50
r = 0.05
$\delta$ = 0.08
$\sigma$ = 0.3
b =3
n = 100
Steps = 3

These are values for 6 prices.
Binomial Tree American Call: 5.39834
Binomial Tree American Put: 6.58504
Random High Tree American Call: 5.8449
Random High Tree American Put: 7.62524
Random Low Tree American Call: 2.80691
Random Low Tree American Put: 4.92571

Random High Estimator gives more accurate result to the Binomial Tree method for both American Put and Call options.

The prices of Binomial Tree American (call/put) is higher than the Random-Tree Low Estimator and lower than the Random-Tree High Estimator.

Line: 1 Col: 1

```
Number of steps
3
Binomial Tree: American Call Option Price =5.39834
Binomial Tree: American Put Option Price =6.58504
Random High Tree: American Call Option Price =5.8449
Random High Tree: American Put Option Price =7.62524
Random Low Tree: American Call Option Price =2.80691
Random Low Tree: American Put Option Price =4.92571
Program ended with exit code: 0
```

**For High Estimator of American Call option:**

With the underlying security prices $X_0$, $X_1$'s, $X_2$'s given and the strike price = 50, the value of the American call option **at maturity or step = 2** is $V2$ = max($X2$ − Strike, 0).

**At step 1,** the value of the American call option is:
$V1$ =max(max($X2$ −Strike,0), Average of3 successingV2

"$h_1$" in the image below is max($X_1$ − Strike, 0).

"(1/b)*sum($V_2$)" in the image below is the Average of 3 successing $V_2$.

**At step 0,** the value of the American call option is:

$V0$ = max(max($XX$ − Strike, 0) , Average of 3 successing V"$h_0$" in the image below is max($XX_0$ − Strike, 0).

"(1/b)*sum($V_1$)" in the image below is the Average of 3 successing $V_1$.

**For Low Estimator of American Call option:**

With the underlying security prices $X_0$, $X_1$'s, $X_2$'s given and the strike price = 50, the value of the American call option **at maturity or step = 2** is $V_2 = \max(X_2 - \text{Strike}, 0)$.

**At step 1,** the value of the American call option is:

$$V_1 = \text{mean}(\text{U}_1, \text{M}_1, \text{L}_1)$$

"$h_1$" in the image below is $\max(X_1 - \text{Strike}, 0)$.

Let $V_2^u, V_2^m, V_2^L$ be the $V_2$ values from the upper, middle, and lower successing nodes.

"U1" in the image below is $\begin{Bmatrix} \text{h}_1 & \mathit{if}\ \text{h}_1 \geq \frac{1}{2}(V_2^m + V_2^l) \\ V_2^u & \text{Otherwise} \end{Bmatrix}$.

"M1" in the image below is $\begin{Bmatrix} \text{h}_1 & \mathit{if}\ \text{h}_1 \geq \frac{1}{2}(V_2^u + V_2^l) \\ V_2^m & \text{Otherwise} \end{Bmatrix}$.

"L1" in the image below is $\begin{Bmatrix} \text{h}_1 & \mathit{if}\ \text{h}_1 \geq \frac{1}{2}(V_2^u + V_2^m) \\ V_2^l & \text{Otherwise} \end{Bmatrix}$.

**At step 0,** the value of the American call option is:

$$V_0 = \text{mean}(\text{U}_0, \text{M}_0, \text{L}_0)$$

"$h_0$" in the image below is $\max(X_0 - \text{Strike}, 0)$.

Let $V_1^u, V_1^m, V_1^L$ be the $V_1$ values from the upper, middle, and lower successing nodes.

"U0" in the image below is $\begin{Bmatrix} \text{h}_0 & \mathit{if}\ \text{h}_0 \geq \frac{1}{2}(V_1^m + V_1^l) \\ V_1^u & \text{Otherwise} \end{Bmatrix}$.

"M0" in the image below is $\begin{Bmatrix} \text{h}_1 & \mathit{if}\ \text{h}_0 \geq \frac{1}{2}(V_1^u + V_1^l) \\ V_1^m & \text{Otherwise} \end{Bmatrix}$.

"L0" in the image below is $\begin{Bmatrix} \text{h}_1 & \mathit{if}\ \text{h}_0 \geq \frac{1}{2}(V_1^u + V_1^m) \\ V_1^l & \text{Otherwise} \end{Bmatrix}$.

Here is the implementation of the Random High Tree C++ file with the HW4 algorithm.

```cpp
#include "RandomHighTree.h"
#include "Arrays.h"
#include <cmath>
#if !defined(_MSC_VER)
using namespace std;
#endif
RandomHighTree1::RandomHighTree1(double Spot_,
                                 double r_,
                                 double d_,
                                 double Volatility_,
                                 unsigned long Steps_,
                                 double Time_,
                                 double b_,
                                 unsigned long NumberOfPaths_)
                                 :Spot(Spot_),
                                 r(r_),
                                 d(d_),
                                 Volatility(Volatility_),
                                 Steps(Steps_),
                                 Time(Time_),
                                 b(b_),
                                 NumberOfPaths(NumberOfPaths_){
    TreeBuilt=false;
}
void RandomHighTree1::BuildTree() {
    TreeBuilt = true;
    TheTree.resize(Steps+1);
    TheTree[0].resize(1);
    TheTree[0][0].first = Spot;

    double delta_t = Time / Steps;

    for (unsigned long i=1; i <=Steps; i++){

        TheTree[i].resize(pow(b,i));

        for (unsigned long j = 0; j < pow(b,i); j++ ){
            unsigned long index = j / static_cast<unsigned long>(b);
            TheTree[i][j].first = GetGBMNextPrice(TheTree[i-1][index].first,r,d,Volatility,delta_t);
        }
    }
}
```

```cpp
double RandomHighTree1::GetThePrice(const TreeProducts &TheProduct) {

    double Sum = 0;

    for (unsigned long i = 0; i < NumberOfPaths; ++i) {
        if (!TreeBuilt)
            BuildTree();

        for (unsigned long j = 0; j < pow(b,Steps); ++j) {
            TheTree[Steps][j].second = TheProduct.FinalPayOff(TheTree[Steps][j].first);
        }

        for (unsigned long j = 1; j <= Steps ; ++j) {
            unsigned long x = Steps-j;
            double RHTtime = x*Time/Steps;

            for (unsigned long k = 0; k < pow(b,x); k++ ){
                double Spot = TheTree[x][k].first;
                double current_sum = 0;
                for (int p = 0; p < b; p++) {
                    current_sum += (TheTree[x + 1][b * k + p]).second;

                }

                double mean = (1.0 / b) * current_sum;
                TheTree[x][k].second = TheProduct.PreFinalValue(Spot,RHTtime,mean);
            }
        }
        Sum += TheTree[0][0].second;
        TreeBuilt = false;
    }
    return Sum/NumberOfPaths;
}
```

Here is the implementation of the Random Low Tree C++ file with the HW4 algorithm.
In this class we set b=3 is fixed

```cpp
#include "RandomLowTree.h"
#include "Arrays.h"
#include <cmath>

// the basic math functions should be in namespace std but aren't in VCPP6
#if !defined(_MSC_VER)
using namespace std;
#endif

RandomLowTree::RandomLowTree(double Spot_,
                             double r_,
                             double d_,
                             double Volatility_,
                             unsigned long Steps_,
                             double Time_,
                             double b_,
                             unsigned long NumberOfPaths_)
        :Spot(Spot_),
         r(r_),
         d(d_),
         Volatility(Volatility_),
         Steps(Steps_),
         Time(Time_),
         b(b_),
         NumberOfPaths(NumberOfPaths_){
    TreeBuilt=false;
}

void RandomLowTree::BuildTree() {
    TreeBuilt = true;
    TheTree.resize(Steps+1);
    TheTree[0].resize(1);
    TheTree[0][0].first = Spot;

    double delta_t = Time / Steps;

    for (unsigned long i=1; i <=Steps; i++){

        TheTree[i].resize(pow(b,i));
        for (unsigned long j = 0; j < pow(b,i); j++ ){
            unsigned long index = j / static_cast<unsigned long>(b);
            TheTree[i][j].first = GetGBMNextPrice(TheTree[i-1][index].first,r,d,Volatility,delta_t);
        }
    }
}
```

```cpp
double RandomLowTree::GetThePrice(const TreeProducts& TheProduct) {

    double Sum = 0;

    for (unsigned long i = 0; i < NumberOfPaths; ++i) {
        if (!TreeBuilt)
            BuildTree();

        if (TheProduct.GetFinalTime() != Time)
            throw("mismatched product in SimpleTrinomialTree");

        if (b != 3)
            throw("b has to be 3");

        for (unsigned long k = 0; k < pow(b, Steps); ++k) {
            TheTree[Steps][k].second = TheProduct.FinalPayOff(TheTree[Steps][k].first);
        }

        for (unsigned long j = 1; j <= Steps; ++j) {
            unsigned long x = Steps - j;
            double ThisTime = x * Time / Steps;

            for (unsigned long k = 0; k < pow(b, x); k++) {
                double h = TheProduct.FinalPayOff(TheTree[x][k].first);
                double V_u = TheTree[x + 1][3 * k].second;
                double V_m = TheTree[x + 1][3 * k + 1].second;
                double V_l = TheTree[x + 1][3 * k + 2].second;
                double U = h >= 0.5 * (V_m + V_l) ? h : V_u;
                double M = h >= 0.5 * (V_u + V_l) ? h : V_m;
                double L = h >= 0.5 * (V_u + V_m) ? h : V_l;

                TheTree[x][k].second = (U + M + L) / b;
            }
        }
        Sum += TheTree[0][0].second;
        TreeBuilt = false;
    }
    return Sum / NumberOfPaths;
}
```

# Question 3:

I added the PayOff-Factory class, including 6 PayOff objects.

Binomial Tree American Call: BTCall
Binomial Tree American Put: BTPut
Random High Tree American Call: RTCallH
Random High Tree American Put: RTPutH
Random Low Tree American Call: RTCallL
Random Low Tree American Put: RTPutL

| BTCall | BTPut | RTCallH | RTCallL | RTPutH | RTPutL |
|---|---|---|---|---|---|
| 5.39834 | 6.58504 | 5.8449 | 2.74067 | 7.41372 | 5.16863 |

BTCall: 5.39834

```
545final                                              Line: 31  Col: 9

FE545 Number of steps Input
3

6 PayOff names (BTCall,BTPut,RTCallH,RTCallL,RTPutH,RTPutL)
BTCall

5.39834
```

BTPut:6.58504

```
545final                                        Line: 31  Col: 9

FE545 Number of steps Input
3

6 PayOff names (BTCall,BTPut,RTCallH,RTCallL,RTPutH,RTPutL)
BTPut

6.58504
```

RTCallH: 5.8449

```
545final                                        Line: 31  Col: 9

FE545 Number of steps Input
3

6 PayOff names (BTCall,BTPut,RTCallH,RTCallL,RTPutH,RTPutL)
RTCallH

5.8449
```

RTCallL:2.74067

```
545final                                        Line: 31  Col: 9

FE545 Number of steps Input
3

6 PayOff names (BTCall,BTPut,RTCallH,RTCallL,RTPutH,RTPutL)
RTCallL

2.74067
```

RTPutH:7.41372

```
545final                                        Line: 31  Col: 9

FE545 Number of steps Input
3

6 PayOff names (BTCall,BTPut,RTCallH,RTCallL,RTPutH,RTPutL)
RTPutH

7.41372
```

RTPutL: 5.16863

```
545final                                        Line: 31  Col: 9

FE545 Number of steps Input
3

6 PayOff names (BTCall,BTPut,RTCallH,RTCallL,RTPutH,RTPutL)
RTPutL

5.16863
```

# Question 4:

In this question. I must move to the windows platform in order to set up the QuantLib environment.Then, we can get 6 prices together with their option id.

The OptionPricingWriter is to call those PayOff objects in the factory with their ID using QuantLib Package. After the environment was settled, the Obeservable pattern can gather 6 Objects' names and their prices in the factory, send them to the QuantLib Observer. Finally, the Observer can control the OptionPricingWriter to print the pricing report.