Design and Analysis of Algorithms

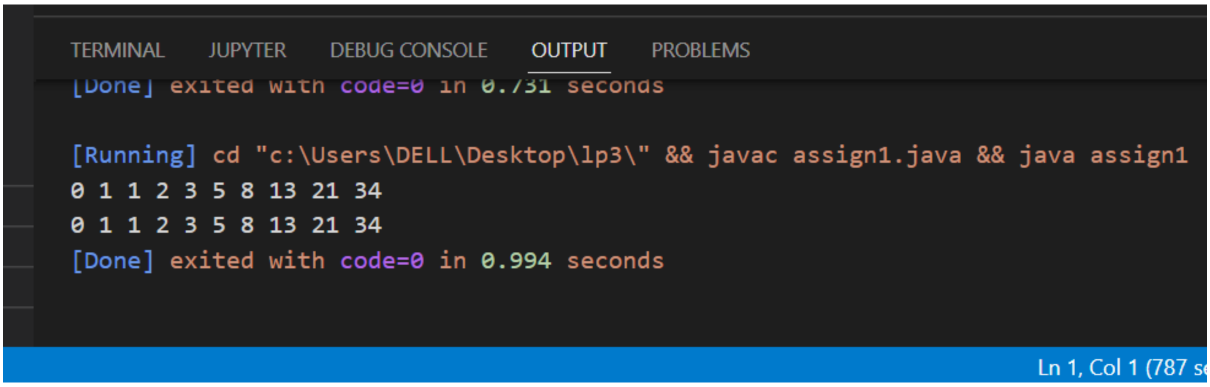
Assign.1 : Fibonacci series

Code:

```
class assign1{

   static int fib_rec(int n)
   {
      if (n ==1)
         return 0;
      else if(n==2)return 1;
      return fib_rec(n - 1) + fib_rec(n - 2);
   }
   static int fib_iter(int n)
   {
      if(n==1)return 0;
      else if(n==2)return 1;
      int f1,f2,f3=0;
      f1=0;
      f2=1;
      for(int i=3;i<=n;i++){
         f3=f1+f2;
         f1=f2;
         f2=f3;
      }
      return f3;
   }
```

```
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.print(fib_iter(i) + " " );
        }
        System.out.println();
        for (int i = 1; i <= 10; i++) {
            System.out.print(fib_rec(i) + " " );
        }




    }
}
```

Output:



```
[Done] exited with code=0 in 0.731 seconds

[Running] cd "c:\Users\DELL\Desktop\lp3\" && javac assign1.java && java assign1
0 1 1 2 3 5 8 13 21 34
0 1 1 2 3 5 8 13 21 34
[Done] exited with code=0 in 0.994 seconds
```

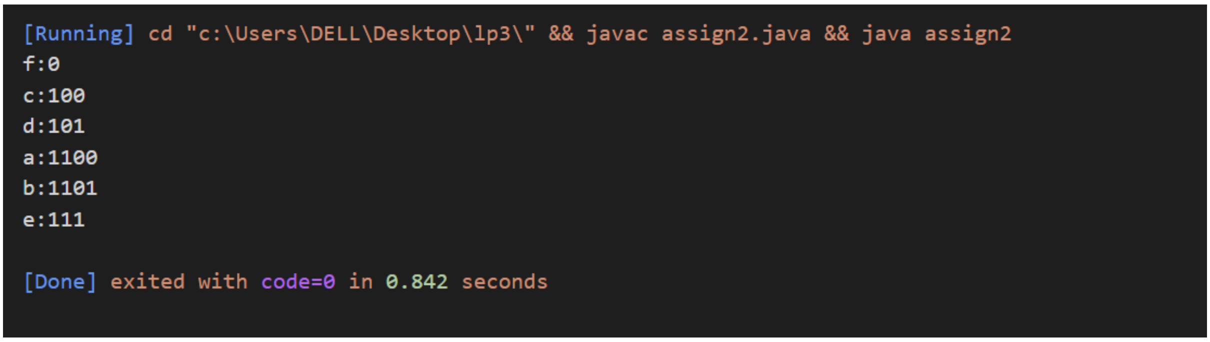
Assign.2 : Huffman encoding (greedy)

Code:

```
import java.util.PriorityQueue;

import java.util.Comparator;


class assign2 {



        public static void printCode(HuffmanNode root, String s)

        {



                if (root.left== null&& root.right== null && Character.isLetter(root.c)) {


                        System.out.println(root.c + ":" + s);

                        return;

                }

                printCode(root.left, s + "0");

                printCode(root.right, s + "1");

        }

        public static void main(String[] args)

        {


                int n = 6;

                char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };

                int[] charfreq = { 5, 9, 12, 13, 16, 45 };

                PriorityQueue<HuffmanNode> q

                        = new PriorityQueue<HuffmanNode>(n, new MyComparator());


                for (int i = 0; i < n; i++) {
```

```
            HuffmanNode hn = new HuffmanNode();

            hn.c = charArray[i];
            hn.data = charfreq[i];
   hn.left = null;
            hn.right = null;
            q.add(hn);
        }

        HuffmanNode root = null;

        while (q.size() > 1) {
            HuffmanNode x = q.peek();
            q.poll();
            HuffmanNode y = q.peek();
            q.poll();
            HuffmanNode f = new HuffmanNode();
            f.data = x.data + y.data;
            f.c = '-';
            f.left = x;
            f.right = y;
            root = f;
            q.add(f);
        }

        printCode(root, "");
    }
}

class HuffmanNode {
    int data;
    char c;
    HuffmanNode left;
    HuffmanNode right;
}


class MyComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode x, HuffmanNode y)
    {

        return x.data - y.data;
    }
}
```

Output:



```
[Running] cd "c:\Users\DELL\Desktop\lp3\" && javac assign2.java && java assign2
f:0
c:100
d:101
a:1100
b:1101
e:111

[Done] exited with code=0 in 0.842 seconds
```

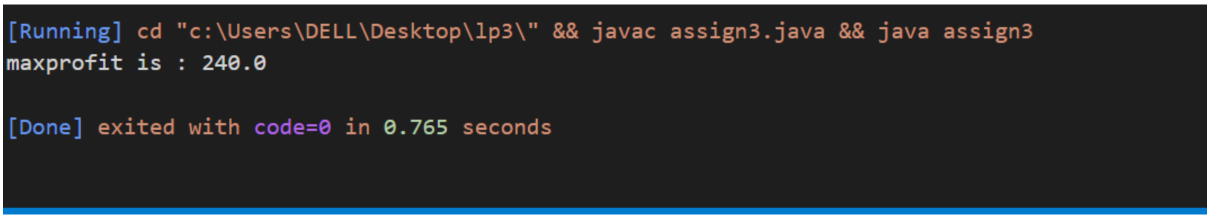
Assign.3 : Fractional knapsack (greedy)

Code:

```
import java.util.Arrays;
import java.util.Comparator;
class item{
   float wt;
   float profit;

   public item(int p,int w){
      this.wt=w;
      this.profit=p;
   }

}

public class assign3 {

   static void fracKnapsack(item [] items ,float bagsize ){

         float maxprofit=0;
         Arrays.sort(items,new MyComparator() );
         for (int i = 0; i < items.length; i++) {

            if(bagsize>=items[i].wt){
               maxprofit=maxprofit +items[i].profit;
               bagsize=bagsize-items[i].wt;

            }else{

               float x= (items[i].profit/items[i].wt )*bagsize;
```

```
                    maxprofit=maxprofit+x;
                    bagsize=0;
                    break;
                }

            }
            System.out.println(maxprofit);

    }
    public static void main(String[] args) {

        item[] arr = { new item(60, 10), new item(100, 20),
            new item(120, 30)
            };

        int bagsize= 50;

        System.out.print("maxprofit is : ");
        fracKnapsack(arr, bagsize);


    }

}
```

Output:



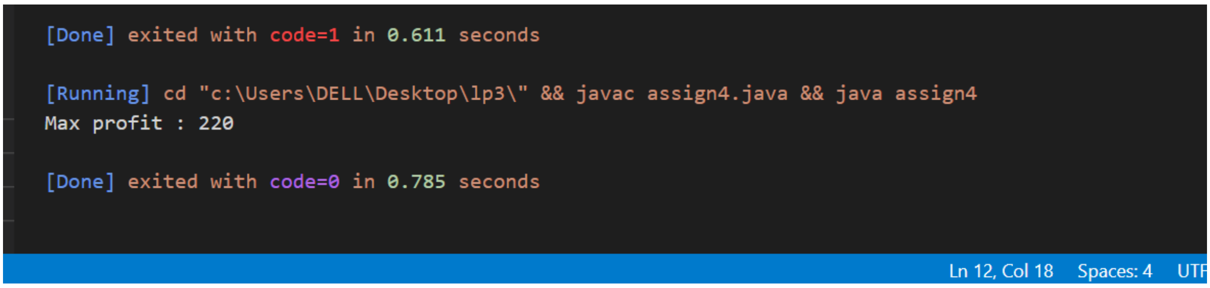
```
[Running] cd "c:\Users\DELL\Desktop\lp3\" && javac assign3.java && java assign3
maxprofit is : 240.0

[Done] exited with code=0 in 0.765 seconds
```

<u>Assign.4 : zero one knapsack</u>

<u>Code:</u>

```
public class assign4 {



  static int zeroneKnapsack(int wt[],int prof[],int bagsize,int n){


    int  [][] dp = new int [n+1][ bagsize +1];


    for(int i=0;i<=n;i++){
      for(int j=0;j<=bagsize;j++){
        if(i==0 || j==0){
          dp[i][j]=0;
        }
       else if( wt[i-1] <= j){

              dp[i][j]=assign4.max( dp[ i-1][j- wt[i-1]] + prof[i-1] , dp[i-1][j]);
        }
        else{
          dp[i][j]= dp[i-1][j];

        }
      }
    }



      return dp[n][bagsize];
  }


  public static void main(String[] args) {
        int prof[]={60,100,120};

        int wt[]={10,20,30};

        int bag=50;


        System.out.println( "Max profit : "+zeroneKnapsack(wt, prof, bag,3));

    }

}
```

Output:



```
[Done] exited with code=1 in 0.611 seconds

[Running] cd "c:\Users\DELL\Desktop\lp3\" && javac assign4.java && java assign4
Max profit : 220

[Done] exited with code=0 in 0.785 seconds
```

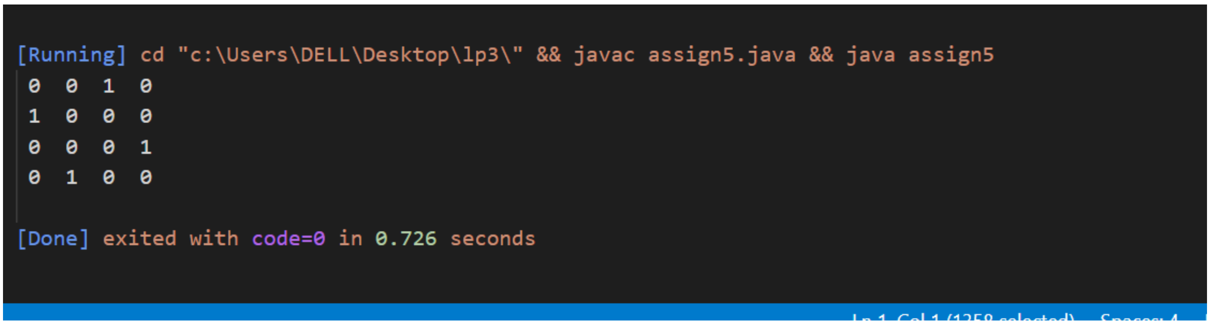
Assign.5 : N queens (backtracking)

Code:

```
public class assign5{

        final int N = 4;

        void printSolution(int board[][])

        {

                for (int i = 0; i < N; i++) {

                        for (int j = 0; j < N; j++)

                                System.out.print(" " + board[i][j]

                                                                + " ");

                        System.out.println();

                }

        }



        boolean isSafe(int board[][], int row, int col)

        {

                int i, j;



                for (i = 0; i < col; i++)

                        if (board[row][i] == 1)

                                return false;



                for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

                        if (board[i][j] == 1)

                                return false;

                for (i = row, j = col; j >= 0 && i < N; i++, j--)

                        if (board[i][j] == 1)

                                return false;
        return true;
}


boolean solveNQUtil(int board[][], int col)
{

        if (col >= N)
                return true;


        for (int i = 0; i < N; i++) {

                if (isSafe(board, i, col)) {

                        board[i][col] = 1;


                        if (solveNQUtil(board, col + 1) == true)
                                return true;

                        board[i][col] = 0;
                }
        }


        return false;
}
```

```
        boolean solveNQ()
        {
                int board[][] = { { 0, 0, 0, 0 },
                                  { 0, 0, 0, 0 },
                                  { 0, 0, 0, 0 },
                                  { 0, 0, 0, 0 } };

                if (solveNQUtil(board, 0) == false) {
                        System.out.print("Solution does not exist");
                        return false;
                }

                printSolution(board);
                return true;
        }

        public static void main(String args[])
        {
                assign5 Queen = new assign5();
                Queen.solveNQ();
        }
}
```

Output:



```
[Running] cd "c:\Users\DELL\Desktop\lp3\" && javac assign5.java && java assign5
 0  0  1  0
 1  0  0  0
 0  0  0  1
 0  1  0  0

[Done] exited with code=0 in 0.726 seconds
```

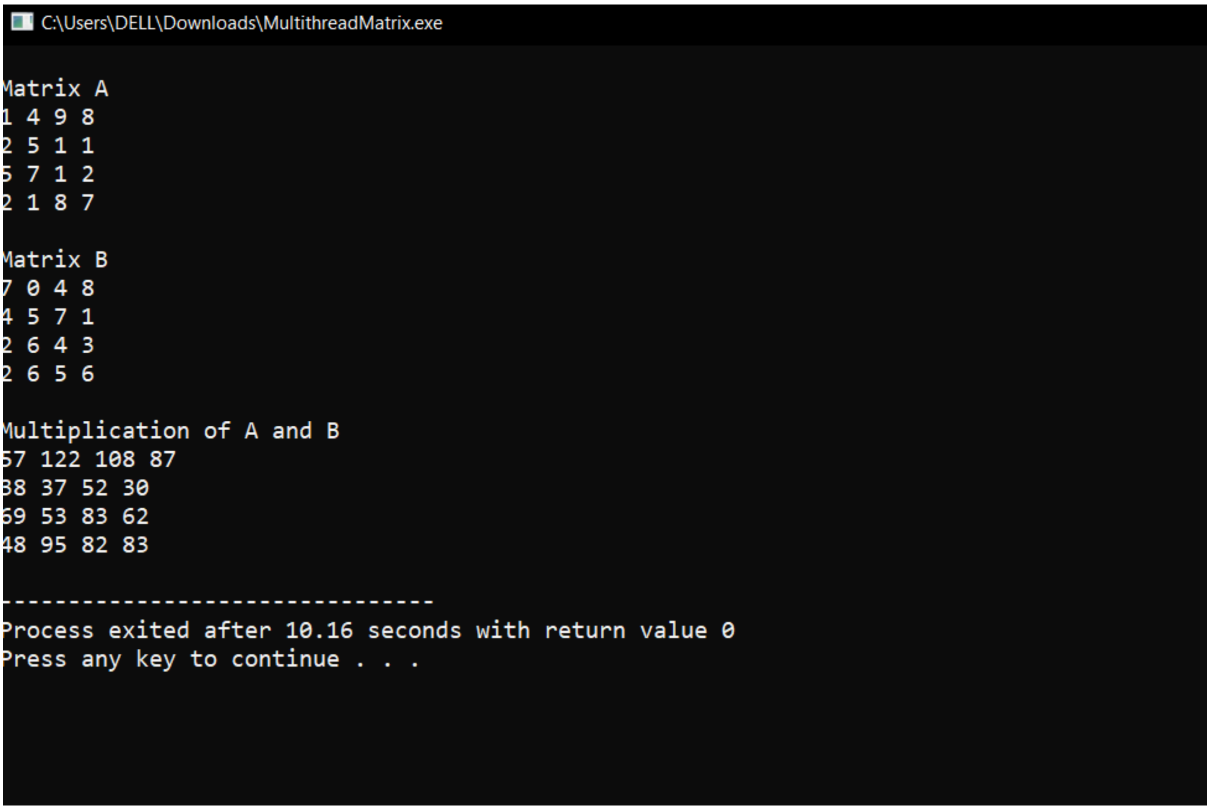
Mini-project

Code:

```
#define MAX 4

// maximum number of threads
#define MAX_THREAD 4

int matA[MAX][MAX];
int matB[MAX][MAX];
int matC[MAX][MAX];
int step_i = 0;

void* multi(void* arg)
{
	int i = step_i++; //i denotes row number of resultant matC

	for (int j = 0; j < MAX; j++)
	for (int k = 0; k < MAX; k++)
		matC[i][j] += matA[i][k] * matB[k][j];
}

// Driver Code
int main()
{
	// Generating random values in matA and matB
	for (int i = 0; i < MAX; i++) {
		for (int j = 0; j < MAX; j++) {
			matA[i][j] = rand() % 10;
			matB[i][j] = rand() % 10;
		}
}

// Displaying matA
cout << endl
          << "Matrix A" << endl;
for (int i = 0; i < MAX; i++) {
          for (int j = 0; j < MAX; j++)
                    cout << matA[i][j] << " ";
          cout << endl;
}

// Displaying matB
cout << endl
          << "Matrix B" << endl;
for (int i = 0; i < MAX; i++) {
          for (int j = 0; j < MAX; j++)
                    cout << matB[i][j] << " ";
          cout << endl;
}

// declaring four threads
pthread_t threads[MAX_THREAD];

// Creating four threads, each evaluating its own part
for (int i = 0; i < MAX_THREAD; i++) {
          int* p;
          pthread_create(&threads[i], NULL, multi, (void*)(p));
}

// joining and waiting for all threads to complete
for (int i = 0; i < MAX_THREAD; i++)
```

```
        pthread_join(threads[i], NULL);

    // Displaying the result matrix
    cout << endl
        << "Multiplication of A and B" << endl;
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++)
            cout << matC[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

Output:



```
Matrix A
1 4 9 8
2 5 1 1
5 7 1 2
2 1 8 7

Matrix B
7 0 4 8
4 5 7 1
2 6 4 3
2 6 5 6

Multiplication of A and B
57 122 108 87
38 37 52 30
69 53 83 62
48 95 82 83

--------------------------------
Process exited after 10.16 seconds with return value 0
Press any key to continue . . .
```