# 1 Exercise 1

1. `static int N = 16;`
2. 0
3. 
```
static boolean equal(List X, List Y) {
    List l1 = X;
    List l2 = Y;
    while (l1 != null && l2 != null && l1.hd == l2.hd) {
        l1 = l1.tl;
        l2 = l2.tl;
    }
    return l1 == l2;
}
```
4. 
```
static int base10(List X) {
    int res = 0;
    List l = X;
    while(l != null) {
        res = res*N+l.hd;
        l = l.tl;
    }
    return res;
}
```

# 2 Exercise 2

1. `static boolean[] T = new boolean[n];`
2. 
```
static void imprimeEnumeration(int n, boolean[] T) {
    if (n<1)
        imprime(T);
    else {
        T[n-1] = false;
        imprimeEnumeration(n-1,T);
        T[n-1] = true;
        imprimeEnumeration(n-1,T);
    }
}
```
3. $\Theta(n2^n)$

# 3 Exercise 3

1. (a) $\neg x_1$

   (b) 
```
new BooleanExpr(2,0,false,
  new BooleanExpr(3,0,false,
     new BooleanExpr(0,0,false,null,null),
     new BooleanExpr(2,0,false,new BooleanExpr(0,1,false,null,null),null)),null);
```

   (c) 
```
BooleanExpr(int v) {
    select  = 0; Var = v; Cste = false; argg = null; argd = null;
}
```

```
BooleanExpr(boolean c) {
    select  = 1; Var = 0; Cste = c; argg = null; argd = null;
}

BooleanExpr(BooleanExpr x, BooleanExpr y) {
    select  = 3; Var = 0; Cste = false; argg = x; argd = y;
}
```

(d)
```
static BooleanExpr Non(BooleanExpr x) {
    return new BooleanExpr(2,0,false,x,null);
}

static BooleanExpr Ou(BooleanExpr x, BooleanExpr y) {
    return Non(new BooleanExpr(Non(x),Non(y)));

}

static BooleanExpr Implique(BooleanExpr x, BooleanExpr y) {
    return Ou(Non(x),y);
}
```

2. (a)
```
static boolean eval(BooleanExpr E, boolean[] T) {
    switch(E.select) {
    case 0: return T[E.Var];
    case 1: return E.Cste;
    case 2: return !(eval(E.argg,T));
    case 3: return eval(E.argg,T) ? eval(E.argd,T) : false;
    case 4: return eval(E.argg,T) ? true : eval(E.argd,T);
    default: return false;
    }

}
```

(b)
```
static boolean enumereSAT(BooleanExpr E, int n, boolean[] T) {
    if (n<1)
        return eval(E,T);
    else {
        T[n-1] = false;
        boolean res = enumereSAT(E,n-1,T);
        T[n-1] = true;
        return res && enumereSAT(E,n-1,T);
    }
}

static boolean SAT(BooleanExpr E) {
    boolean[] T = new boolean[N];
    return enumereSAT(E,N,T);

}
```

(c) $\Theta(k)$, $\Theta(k2^n)$

3. (a)
```
static BooleanExpr Neg(BooleanExpr E) {
    switch(E.select) {
    case 0: case 1: return Non(E);
    case 2: return E.argg;
    case 3: return new BooleanExpr(4,0,false,Neg(E.argg),Neg(E.argd));
```

```
        case 4: return new BooleanExpr(Neg(E.argg),Neg(E.argd));
        default: return E;
        }
    }

    static BooleanExpr NNF(BooleanExpr E) {
        switch (E.select) {
        case 0: case 1: return E;
        case 2: return Neg(NNF(E.argg));
        case 3: return new BooleanExpr(NNF(E.argg),NNF(E.argd));
        case 4: return new BooleanExpr(4,0,false,NNF(E.argg),NNF(E.argd));
        default: return E;
        }
    }
```

(b)

```
    static BooleanExpr simplifier(BooleanExpr E) {
        if (E==null)
            return E;
        else {
            BooleanExpr e1 = simplifier(E.argg);
            BooleanExpr e2 = simplifier(E.argd);
            switch (E.select) {
            case 2: return (e1.select==2) ? e1.argg :  Non(e1);
            case 3:
                switch (e1.select) {
                case 0:
                    if (e2.select == 0 && e1.Var==e2.Var)
                        return e1;
                    else return new BooleanExpr(e1,e2);
                case 1:
                    return e1.Cste ? e2 : new BooleanExpr(false);
                }
            case 4:
                switch (e1.select) {
                case 0:
                    if (e2.select == 0 && e1.Var==e2.Var)
                        return e1;
                    else return new BooleanExpr(e1,e2);
                case 1:
                    return e1.Cste ? new BooleanExpr(true) : e2;

                }
            default: return E;
            }
        }
    }
```