Lösungsidee: Als erstes muss das dynamische Feld mit Hilfe von GetMem allokiert werden. Dann muss dieses Feld ausgefüllt werden mit den Werten der Knoten von einen Binären Baum. So jetzt muss ein neuer Binärer Baum erstellt werden der sortiert und balanciert ist. Meine Idee ist es den Baum mithilfe von Start und ende Grenzen (auf dem Array bezogen) zu balancieren. Wobei wenn man den linken Teil des aktuellen Knotens ausfüllen will man als Start Grenze den aktuellen Anfang im Array übergibt und als end Grenze der aktuelle Median des Arrays -1. Für die rechten Teilbäume ist die START Grenze der Median+1 des Arrays und als Endgrenze das aktuelle Ende des Arrays. Der Ausgangspfad der Rekursion wird durchlaufen, wenn die Start Grenze größer als die Endgrenze wird wobei dann an der aktuellen stelle im Baum ein NIL gesetzt wird

Quellcode:

```
PROGRAM balTree;

TYPE
  NodePtr = ^Node;
  Node = RECORD
           value : INTEGER;
           left, right : NodePtr;
         END;
  Tree = NodePtr;
  DynIntArray = ARRAY[1..1] OF INTEGER;
  DynIntArrayPtr = ^DynIntArray;

PROCEDURE CopyBinTreeToDynArr(t : Tree;
                              arr : DynIntArrayPtr;
                              VAR pos : LONGINT);
BEGIN
  IF t <> NIL THEN BEGIN
    (* in-order traversal *)
    CopyBinTreeToDynArr(t^.left, arr, pos);
    (*$R-*)
    arr^[pos] := t^.value;
    (*$R+*)
    Inc(pos);
    CopyBinTreeToDynArr(t^.right, arr, pos);
  END;
END;

FUNCTION NewNode(val : INTEGER) : NodePtr;
VAR n : NodePtr;
BEGIN
```

```
  New(n);
  n^.value := val;
  n^.left := NIL;
  n^.right := NIL;
  NewNode := n;
END;

PROCEDURE InsertSorted(v : INTEGER; VAR t : Tree);
BEGIN (* InsertSorted *)
  IF (t = NIL) THEN BEGIN
    t := NewNode(v);
  END ELSE IF (v > t^.value) THEN BEGIN
    InsertSorted(v,t^.right);
  END ELSE IF (v < t^.value) THEN BEGIN
    InsertSorted(v,t^.left);
  END;
END; (* InsertSorted *)

FUNCTION NumNodes(t : Tree) : LONGINT;
VAR
count : LONGINT;
BEGIN (* NumNodes *)  IF (t <> NIL) THEN BEGIN
    count := 1 + NumNodes(t^.left);
    count := 1 + NumNodes(t^.right);
  END ELSE BEGIN
    count := 0;
  END;
  NumNodes := count;
END; (* NumNodes *)

PROCEDURE Balance(arr : DynIntArrayPtr; start,ende : INTEGER; VAR t : Tree);
VAR
median : INTEGER;
BEGIN (* NewBalancedTree *)
  IF (start > ende) THEN BEGIN
    t := NIL;
  END ELSE BEGIN
    median := (start+ende) DIV 2;
    (*$R-*)
    t := NewNode(arr^[median]);
    WriteLn('NODE ',arr^[median]);
    (*$R+*)
    Balance(arr,start,median-1,t^.left);
    Balance(arr,median+1,ende,t^.right);
  END;
```

```
END; (* NewBalancedTree *)
//1234 5 678910
//67 8 9

FUNCTION GetLengthDyn(arr : DynIntArrayPtr): INTEGER;
VAR
count,i : INTEGER;
BEGIN (* GetLengthDyn *)
  count := 0;
  FOR i := LOW(arr^) TO HIGH(arr^) DO BEGIN
    WriteLn(arr^[i]);
    Inc(count);
  END; (* FOR *)
  GetLengthDyn := count;
END; (* GetLengthDyn *)

PROCEDURE WriteTreeInOrder(t : Tree);
BEGIN
  IF t <> NIL THEN BEGIN
    WriteTreeInOrder(t^.left);
    Write(t^.value, ' ');
    WriteTreeInOrder(t^.right);
  END;
END;

VAR t,bt : Tree;
    v, n : INTEGER;
    pos : LONGINT;
    arr : ^DynIntArray;

BEGIN (* balTree *)
  t := NIL;
  REPEAT
    Read(v);
    IF v <> 0 THEN
      InsertSorted(v, t);
  UNTIL v = 0;
  (* allocate dynamic array with correct length *)
  n := NumNodes(t);
  pos := 1;
  IF n > 0 THEN BEGIN
    GetMem(arr, SIZEOF(INTEGER) * n);
    CopyBinTreeToDynArr(t, arr, pos);
  END;
  Balance(arr,1,n,bt);
```

```
  WriteTreeInOrder(bt);
  FreeMem(arr, SIZEOF(INTEGER) * n)
END. (* balTree *)
```

Testfälle:

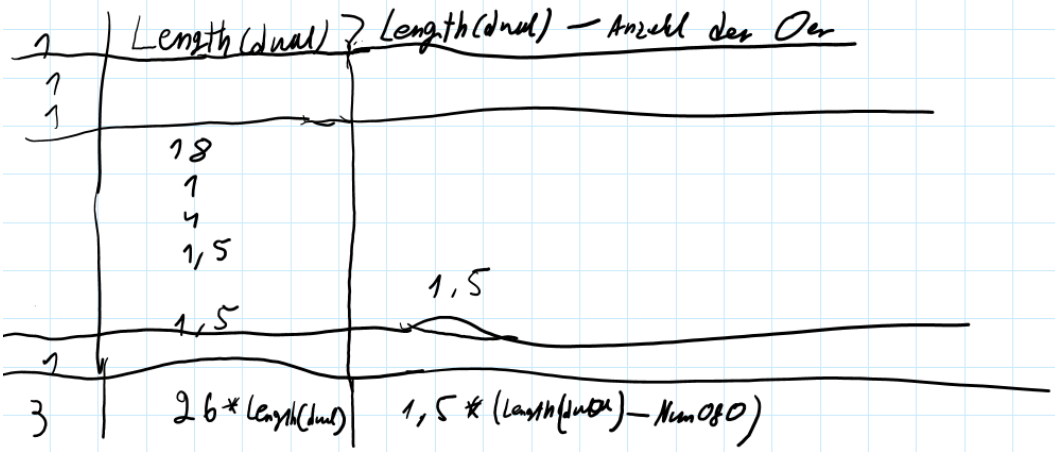
```
Eingabe:
1 2 3 4 5 6 7 8 9 10 0
Ordnung wie die Werte im Baum gespeichert werden:
NODE 5
NODE 2
NODE 1
NODE 3
NODE 4
NODE 8
NODE 6
NODE 7
NODE 9
NODE 10
In-Order WriteTree:
1 2 3 4 5 6 7 8 9 10
```

```
Eingabe:
11 22 33 44 55 66 77 88 99 0
Ordnung wie die Werte im Baum gespeichert werden:
NODE 55
NODE 22
NODE 11
NODE 33
NODE 44
NODE 77
NODE 66
NODE 88
NODE 99
In-Order WriteTree:
11 22 33 44 55 66 77 88 99
```

```
Eingabe:
1 0
Ordnung wie die Werte im Baum gespeichert werden:
NODE 1
In-Order WriteTree:
1
```

```
FUNCTION IntOf(dual: STRING): INTEGER;
  VAR
    result, i: INTEGER;
BEGIN
   result := 0;
   i := 1;
   WHILE i <= Length(dual) DO BEGIN
      result := result * 2;
      IF dual[i] = '1' THEN
         result := result + 1;
      i := i + 1;
   END; (* WHILE *)
   IntOf := result;
END; (* IntOf *)
```

Operation	Ausführungszeit
Wertzuweisung	1
Vergleich	1
Indizierung	0,5
Addition, Subtraktion	0,5
Multiplikation	3
Prozeduraufruf	16 + 2 * Anzahl der Parameter



$$3 + 26 * m + 1{,}5 * (m - n)$$

m=Länge
n= Anzahl 0er

$3+26*6+1{,}5\times(6-4) = 162 \quad (100100)$

$3+26*6+1{,}5\times(6-4) = 162 \quad (100001)$

$3+26*6+1{,}5\times(6-3) = 163{,}5 \quad (110100)$

$3+26*4+1{,}5\times(4-0) = 113 \quad (1111)$

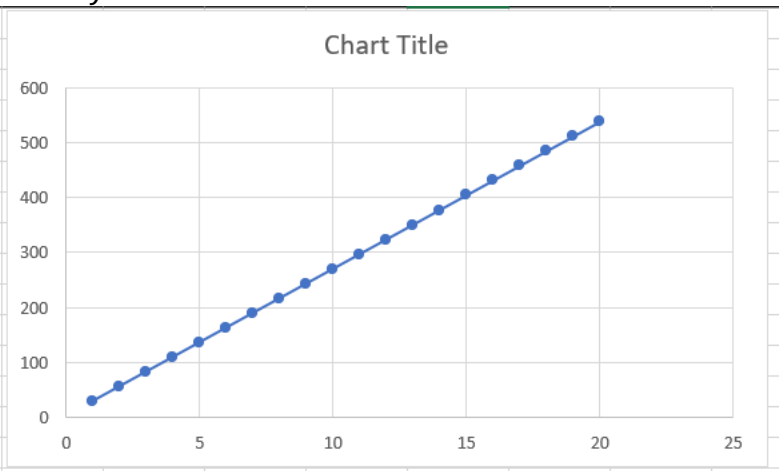
$3+26*4+1{,}5\times(4-4) = 107 \quad (0000)$

$3+26*1+1{,}5\times(1-0) = 30{,}5 \quad (1)$

$3+26*1+1{,}5\times(1-1) = 29 \quad (0)$

b) Ich wusste nicht wie ich das mit der Grobanalyse machen soll weshalb ich die Formel der Feinanalyse benutzt hab.

1	29,75
2	56,5
3	83,25
4	110
5	136,75
6	163,5
7	190,25
8	217
9	243,75
10	270,5
11	297,25
12	324
13	350,75
14	377,5
15	404,25
16	431
17	457,75
18	484,5
19	511,25
20	538
50	1340,5
100	2678
200	5353



c) Anhand des Graphs kann man erkennen dass es O(n) ist.

3)

```
1  FUNCTION IntOfDual(dual: STRING): INTEGER;

2    FUNCTION IoDRec(pos: INTEGER): INTEGER;
3    BEGIN
4      IF pos = 0 THEN
5        IoDRec := 0
6      ELSE IF dual[pos] = '1' THEN
7        IoDRec:= IoDRec(pos - 1) * 2 + 1
8      ELSE
9        IoDRec:= IoDRec(pos - 1) * 2;
10   END; (* IoDRec *)

11 BEGIN (* IntOfDual *)
12   IntOfDual := IoDRec(Length(dual));
13 END; (* IntOfDual *)
```

Operation	Ausführungszeit
Wertzuweisung	1
Vergleich	1
Indizierung	0,5
Addition, Subtraktion	0,5
Multiplikation	3
Prozeduraufruf	16 + 2 * Anzahl der Parameter

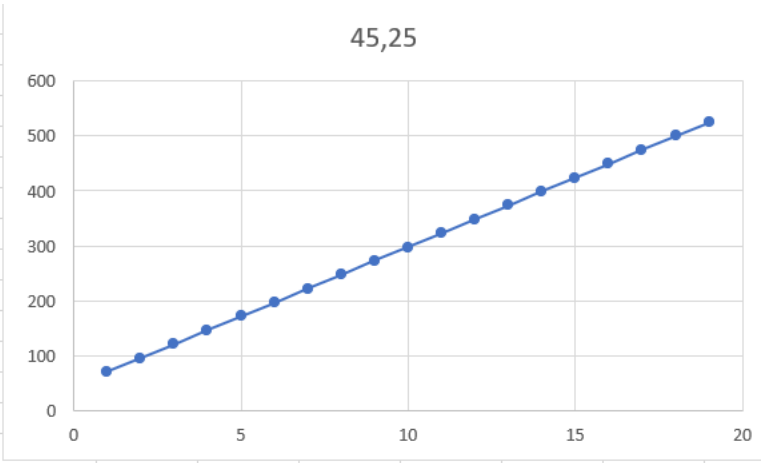
1er Zweig:
1
1
2er Zweig:
1,5
1+16+2+0,5+3+0,5=23
3er Zweig:
22,5 1

Formel:1*n+24,5*m+22,5*p+1+19;

Formel:1*n+1,5*n+23*m+22,5*p+1+19;

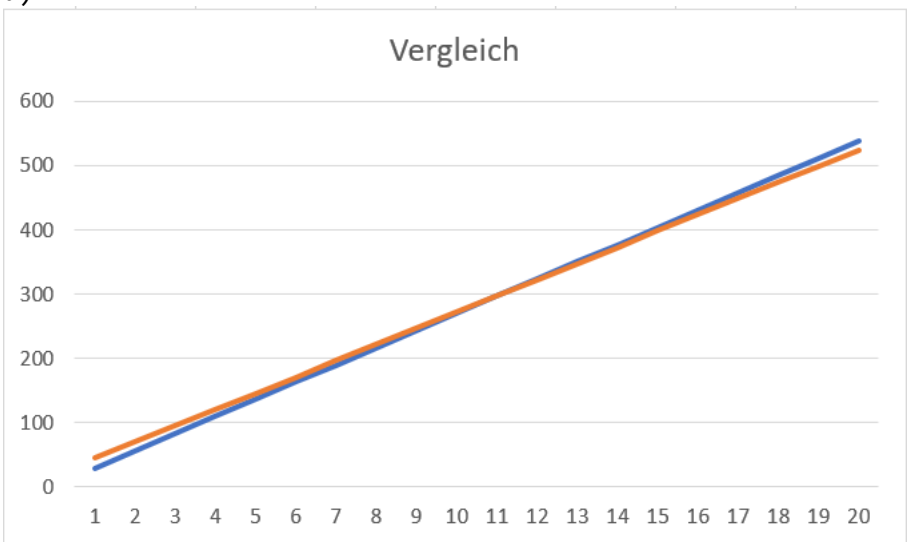
n = Länge; m = Anzahl der 1; p= Anzahl der 0er

1	45,25
2	70,5
3	95,75
4	121
5	146,25
6	171,5
7	196,75
8	222
9	247,25
10	272,5
11	297,75
12	323
13	348,25
14	373,5
15	398,75
16	424
17	449,25
18	474,5
19	499,75
20	525
50	1282,5
100	2545
200	5070



c) Anhand des Graphens kann man erkennen dass es eine O(n) ist.

d)



Orange = Rekursiv; Blau = Iterativ

Für kleinere Zahlen würde sich die iterative Lösung eher Lohnen als die Rekursive. Für Längere Zahlen lohnt sich die rekursive Lösung eher.