**Opgave 1**

Bij het zoeken of de som bestaat hebben we te maken met meerder loops. Als eerste A[i], deze begint te tellen bij positie twee, A[2], van de array zodat we de eerste twee posities A[0] + A[1] kunnen optellen.

Als er geen som is moet als eerste de positie A[i] met één worden verhoogd, in mijn methode is dit *sumPos*. Nu kunnen moeten we weer vanaf 0 beginnen met het zoeken naar de som. In mijn methode is *start* het begin van de eerste som loop. Vervolgens zo is er een twee teller *loopCounter* die alle posities vanaf *start + 1* tot *sumPos -1* doorloopt om zo te bepalen of de som bestaat. Als dit geen succes oplevert kan de start positie met één worden verhoogd en kan de *loopCounter* opnieuw zijn werk doen. Bij het aanroepen moet: sumPos op twee, start op nul en loopCounter op éé’n staan

public static boolean sum(List<Integer> lijst, int index) {

if (lijst.size() == 0 || index < 2 || index > lijst.size() - 1) {

return false;

}

int verschil = lijst.get(index) - lijst.get(0);

lijst.remove(0);

index--;

if (verschil < 0 && lijst.size() > 0) {

return sum(lijst, index);

}

if (lijst.indexOf(verschil) != -1 && lijst.indexOf(verschil) < index) {

return true;

}

if (lijst.size() > 0) {

return sum(lijst, index);

}

return false;

}

**Opgave 3a**

Bij een volle binaire zijn alle lagen volledig gevuld met knooppunten. Het aantal externe knooppunten is dan gelijk aan 2h. Aangezien het aantal knooppunten per laag steeds verdubbeld hebben we te maken met machten van de afhankelijk zijn van de hoogte. Dus voor een hoogte van drie zijn er acht externe nodes

**Opgave 4**

protected void upheap(int j) {

if (j != 0) {

int p = parent(j);

if ((compare(heap.get(j), heap.get(p)) < 0)) {

swap(p, j);

}

j = p;

upheap(j);

}

}