Aufgabe 1

akku = -1

iopt = -1

jopt = -1

**for** i from1 to n

**for** j from i to n

**if**(InvW(j) - InvW(i) > akku)

iopt = i

jopt = j

akku = InvW(j) - InvW(i)

endif

endfor

endfor

Aufgabe 2

**public** **static** List<Integer> investment(List<Integer> investments){

**int** accu = -1;

**int** iopt = 0;

**int** jopt = 0;

**int** diff = 0;

List<Integer> result = **new** ArrayList<Integer>();

**for**(**int** i = 0 ; i < investments.size()-1 ; i++){

**for**(**int** j = i ; j < investments.size()-1 ; j++){

diff = investments.get(j) - investments.get(i);

**if**( diff > accu){

iopt = i;

jopt = j;

accu = diff;

}

}

}

result.addAll(Arrays.*asList*(iopt,jopt,accu));

**return** result;

}

Aufgabe 3  
(Entwurf für Listen der Länge >= 4)

1. Teile die Ursprungsliste in Listen der Länge 4  
2. Speichere diese in der ursprünglichen Reihenfolge in lists  
3. Ersetze jede Liste in lists durch eine Liste der Form   
 [min,max,bestmin,bestmax,best]  
wobei min -> das Minimum der Liste  
 max -> das Maximum der Liste  
 bestMin -> Minimum des besten Intervalls der Liste  
 bestMax -> Maximum des besten Intervalls der Liste  
 best -> Größe des besten Intervalls der Liste  
4. Rufe lists rekursiv auf und zerteile sie bei jedem Aufruf in zwei weitere Teillisten, bis die Teilliste genau zwei Listenelemente hat  
   
5. Füge diese Zwei Listen zusammen, sodass sie wieder der obigen Form entsprechen. Tue dies solange, bis nur noch ein Listenelement übrig ist.  
6. Wenn lists[4] kleiner als 0 ist, dann gib einen Fehler aus  
7. Gib ein Array zurück mit den Elementen [indexOf(lists[2]),indexOf(lists[3]),list[4]]

Aufgabe 4

Die Idee ist, die Eingabe in Listen aus vier Elementen zu teilen und über diese vier Elemente Aussagen zu treffen. Diese Aussagen sind die maximale Größe, die minimale Größe und Informationen über den größt möglichen Intervall innerhalb jeder diese Teillisten.  
Nun können wir zwei benachbarte Teillisten ohne Informationsverlust zusammenführen, indem wir  
a) das Maximum beider Listen bestimmen und als das neue Maximum annehmen  
b) das Minimum beider Listen bestimmen und als das neue Minimum annehmen  
c) den größt möglichen Intervall ersetzen durch den größten von  
 - der linken Liste  
 - der rechten Liste  
 - dem entstehenden Intervall des Minimums der linken und Maximums der rechten Liste

Dadurch haben wir linear viele Vergleiche der Elemente untereinander um die Eigenschaften der jeweiligen 4-Elementigen Listen zu bestimmen mal logarithmisch viele Schritte um diese Teillisten zusammenzufügen

Aufgabe 5

1. Wenn es genau ein Optimum gibt, dann wird an irgendeiner Stelle entweder bei 3.3 oder 3.4 ein positiver Wert als Optimum abgespeichert. Hierdurch wird bei 3.6 kein Fehler geworfen und daraufhin bei 3.7 ein Ergebnis ausgegeben.
2. Wenn es kein Optimum gibt dann werden beim Zusammenfügen der Listen nur negative beste Intervalle gefunden. Dadurch wird dann bei Punkt 3.6 ein Fehler geworfen.
3. - wenn das Optimum innerhalb einer der bei 3.3 erzeugten Teillisten liegt, dann wird innerhalb dieser Teilliste in der [4] die Größe dieses Optimums und in [2] und [3] die Grenzen gespeichert. Beim Zusammenfügen der Teillisten wird nun immer dieses Optimum übernommen, weil es größer ist als alle anderen (zwangsweise) negativen besten Optima. Der Intervall wird in 3.7 zurückgegeben.  
   - wenn das Optimum in zwei verschiedenen Arrays liegt, dann wird bei 3.5 dieses Optimum irgendwann durch das Zusammenfügen gefunden, weil das Minimum der linken und das Maximum der rechten Liste dann dem besten Intervall entsprechen. Dieser wird in 3.7 zurückgegeben.
4. Die einzige Möglichkeit einen Fehler zu melden hat der Algorithmus, wenn nach dem bestimmen der Daten für die Intervalle in Punkt 3.3 bei jedem bestimmen eine negative Zahl gefunden wird und beim Zusammenfügen dieser Listen bei 3.4 das Minimum der linken Liste immer größer ist als das Maximum der rechten Liste. Dies kann ausschließlich der Fall sein, wenn die Liste monoton fallend ist, weil ansonsten entweder bei 3.3 oder 3.4 ein nicht negativer Wert für das beste Intervall gespeichert werden würde.

Aufgabe 6

Elemente Aufteilen und Werte bestimmen = O(n/4)  
Weil wir genau ein mal über die Liste iterieren und bei jedem Interationsschritt nur Konstante Operationen ausführen.  
  
Der rekursive Aufruf:  
n beschreibt die Länge der Eingabeliste

c = 25, weil dies ungefähr der Anzahl der Rechenschritt entspricht die für das Zusammenfügen der Listen benötigt wird.  
a = 2, weil zwei Aufrufe gemacht werden müssen um die Teillösungen zu bestimmen  
b = 2, weil die Größe der Eingabe in zwei Teillösungen geteilt wird  
f(n) = n, weil genau ein mal über die Eingabe iteriert werden muss, um die Teileingaben zu erzeugen

Damit ist die Form auf den zweiten Fall des Mastertheorems zurückzuführen, weil

Somit ist die Komplexität unseres Algorithmus

Bzw

Aufgabe 7

**public** **static** List<Integer> investm(List<Integer> investments) {

List<Integer> akku = **new** ArrayList<Integer>();

List<List<Integer>> lists = **new** ArrayList<List<Integer>>();

List<List<Integer>> lists2 = **new** ArrayList<List<Integer>>();

**if**(investments.size() == 1) {

**throw** **new** IllegalArgumentException("Das Array hat nur ein Element");

} **else** **if**(investments.size() == 2){

**if**(investments.get(0) <= investments.get(1)) {

akku.addAll(Arrays.*asList*(0,1,investments.get(1) - investments.get(0)));

**return** akku;

} **else** {

**throw** **new** IllegalArgumentException("Es gibt keinen gewinnbringenden Zeitraum");

}

}

//Aufteilen der Liste in 4er Arrays

**for**(**int** i = 0; i < investments.size(); i += 4) {

lists.add(Arrays.*asList*(investments.get(i), investments.get(i+1), investments.get(i+2), investments.get(i+3)));

}

// Speichere in jedem 4er Tupel in der Form [min,max,bestmin,bestmax,best]

**for**(List<Integer> list : lists) {

**int** min = Integer.*MAX\_VALUE*;

**int** max = Integer.*MIN\_VALUE*;

**for**(**int** i = 0; i < 4; i++) {

**if**(list.get(i) > max) {

max = list.get(i);

} **else** **if**(list.get(i) < min) {

min = list.get(i);

}

}

List<Integer> best = *investment*(list);

lists2.add(Arrays.*asList*(min,max,list.get(best.get(0)),list.get(best.get(1)),best.get(2)));

}

akku = *investHelper*(lists2);

akku.set(2, investments.indexOf(akku.get(2)));

**for**(**int** i = akku.get(2); i < investments.size();i++) {

**if**(akku.get(3) == investments.get(i)) {

akku.set(3, i);

**break**;

}

}

**return** Arrays.*asList*(akku.get(2), akku.get(3), akku.get(4));

}

**private** **static** List<Integer> investHelper(List<List<Integer>> lists) {

**if**(lists.size() == 1) {

**return** lists.get(0);

} **else** **if** (lists.size() == 2) {

**int** num1 = lists.get(0).get(4);

**int** num2 = lists.get(1).get(4);

**int** num3 = lists.get(1).get(1) - lists.get(0).get(0);

**int** max = lists.get(0).get(1) > lists.get(1).get(1) ? lists.get(0).get(1) : lists.get(1).get(1);

**int** min = lists.get(0).get(0) < lists.get(1).get(0) ? lists.get(0).get(0) : lists.get(1).get(0);

**if**(num1 > num2 && num1 > num3) {

lists.get(0).set(0, min);

lists.get(0).set(1, max);

**return** lists.get(0);

} **else** **if** (num2 > num1 && num2 > num3) {

lists.get(1).set(0, min);

lists.get(1).set(1, max);

**return** lists.get(1);

} **else** {

**return** Arrays.*asList*(lists.get(0).get(0),lists.get(1).get(1),lists.get(0).get(0),lists.get(1).get(1),lists.get(1).get(1) - lists.get(0).get(0));

}

} **else** {

List<List<Integer>> list1 = **new** ArrayList<List<Integer>>();

List<List<Integer>> list2 = **new** ArrayList<List<Integer>>();

**for**(**int** i = 0; i < lists.size(); i++) {

**if**(i <= lists.size()/2) {

list1.add(lists.get(i));

} **else** {

list2.add(lists.get(i));

}

}

**return** *investHelper*(Arrays.*asList*(*investHelper*(list1),*investHelper*(list1)));

}

}