Uebung 1

1.
O(1)

```
1.  public char charAt(int index) {
2.   if ((index < 0) || (index >= value.length)) {
3.   throw new StringIndexOutOfBoundsException(index);
4.   }
5.   return value[index];
6.  }
7.
8.  public static int getZahl(int i){
9.  int[] elemente =
10. {4,6,8,2,5,10,11,14,13,19,15,3,34,23,9,7,47,39,21,99};
11. return elemente[i];
12. }
```

Genau ein Zugriff egal wie groß das Array ist
Meisten Anweisungen in einem Programm werden nur einmal oder eine kontante Anzahl von Malen
wiederholt. Haben alle Anweisungen eines Programmes diese Eigenschaft, spricht man von
konstanter Laufzeit

O(n)
Lineare Suche im Array

```
1.  public static void printAllItems(int[] arrayOfItems) {
2.  for (int item : arrayOfItems) {
3.  System.out.println(item);
4.  }
5.  }
6.
7.  public int findSmallest(int[] ar){
8.   smallest = ar[i];
9.   for(int i = 0; i < ar.length; i++){
10.  if(ar[i] < smallest){
11.  smallest = ar[i];
12.  }
13.  }
14. }
```

Muss das ganze Array einmal durchlaufen ◊ je größer das Array ist desto länger geht es was
zur Notation von O(n) führt.
Best case, worst case und average case sind identisch.

O(log(n))

```
1.  public int binarySearch(int[] inputArr, int key) {
2.   int start = 0;
3.   int end = inputArr.length - 1;
4.   while (start <= end) {
5.   int mid = (start + end) / 2;
6.   if (key == inputArr[mid]) {
7.   return mid;
8.   }
9.   if (key < inputArr[mid]) {
10.  end = mid - 1;
11.  } else {
12.  start = mid + 1;
13.  }
14.  }
15.  return -1;
16. }
17. int power(int x, unsigned int y){
18.  int temp;
19.  if( y == 0)
20.  return 1;
21.  temp = power(x, y/2);
22.  if (y%2 == 0)
23.  return temp*temp;
24.  else
25.  return x*temp*temp;
26. }
```

O(n2)

```
1.  public void AllPairSums(int[] arrayOfNumbers) {
2.   for (int firstNumber : arrayOfNumbers) {
3.   for (int secondNumber : arrayOfNumbers) {
4.   System.out.println(firstNumber + secondNumber);
5.   }
6.   }
7.   }
8.  }
9.  bool ContainsDuplicates(IList<string> elements){
10.  for (var outer = 0; outer < elements.Count; outer++){
11.  for (var inner = 0; inner < elements.Count; inner++){
12.  // Don't compare with self
13.  if (outer == inner) continue;
14.  if (elements[outer] == elements[inner]) return true;
15.  }
16.  }
17.  return false;
18. }
```

Verdoppelung der Daten führt zur Vervierfachung der Laufzeit, bei Verdreifachung zur Vernaunfachung etc. ◊ k2
Schritte für k Daten