# Algorithm performance

Big-O: nested loops

## By the end of this video you will be able to…

- Compute the big-O class of code with nested loops

```
public static int maxDifference (int[] vals) {
  int max = 0;
  for (int i=0; i < vals.length; i++) {
    for (int j=0; j < vals.length; j++) {
      if (vals[i] - vals[j] > max) {
        max = vals[i] - vals[j];
      }
    }
  }
  return max;
}
```

```
public static int maxDifference (int[] vals) {
  int max = 0;
  for (int i=0; i < vals.length; i++) {
    for (int j=0; j < vals.length; j++) {
      if (vals[i] - vals[j] > max) {
        max = vals[i] - vals[j];
      }
    }
  }
  return max;
}
```

## IVQ: sample run

```
public static int maxDifference (int[] vals) {
  int max = 0;
  for (int i=0; i < vals.length; i++) {
    for (int j=0; j < vals.length; j++) {
      if (vals[i] - vals[j] > max) {
        max = vals[i] - vals[j];
      }
    }
  }
  return max;
}
```

```
public static int maxDifference (int[] vals) {
  int max = 0;
  for (int i=0; i < vals.length; i++) {
    for (int j=0; j < vals.length; j++) {
      if (vals[i] - vals[j] > max) {
        max = vals[i] - vals[j];
      }
    }
  }
  return max;
}
```

```
public static int maxDifference (int[] vals) {
   int max = 0;
   for (int i=0; i < vals.length; i++) {
      for (int j=0; j < vals.len
         if (vals[i] -
            ma

      }
   }
   return
}
```

**Count from the inside out**

```
public static int maxDifference (int[] vals) {
  int max = 0;
  for (int i=0; i < vals.length; i++) {
    for (int j=0; j < vals.length; j++) {
      if (vals[i] - vals[j] > max) {
        max = vals[i] - vals[j];
      }
    }
  }
  return max;
}
```

```
public static int maxDifference (int[] vals) {
  int max = 0;
  for (int i=0; i < vals.length; i++) {
    for (int j=0; j < vals.length; j++) {
      if (vals[i] - vals[j] > max) {
        max = vals[i] - vals[j];
      }
    }
  }
  return max;
}
```

O(1)

```
public static int maxDifference (int[] vals) {
   int max = 0;
   for (int i=0; i < vals.length; i++) {
      for (int j=0; j < vals.length; j++) {

                                        O(1)
      }
   }
   return max;
}
```

```
public static int maxDifference (int[] vals) {
   int max = 0;
   for (int i=0; i < vals.length; i++) {
      for (int j=0; j < vals.length; j++) {



      }
   }
   return max;
}
```

O(n)

```
public static int maxDifference (int[] vals) {
   int max = 0;
   for (int i=0; i < vals.length; i++) {
```

O(n)

```
   }
   return max;
}
```

```
public static int maxDifference (int[] vals) {
  int max = 0;
  for (int i=0; i < vals.length; i++) {



  }
  return max;
}
```

O(n)

```
public static int maxDifference (int[] vals) {
   int max = 0;
   for (int i=0; i < vals.length; i++) {

   }
   return max;
}
```

$O(n)$

$O(n^2)$

```
public static int maxDifference (int[] vals) {
   int max = 0;   O(1)

                                                   O(n²)

   return max;   O(1)
}
```

```
public static int maxDifference (int[] vals) {
   int max = 0;
```

$O(1)$

$O(n^2)$

```
   return max;
```

$O(1)$

```
}
```

**Total: $O(n^2)$**