Algorithms NMCGJ

Comparison between iterative and recursive algorithms.

Greatest common divisor

```
// precondition: x > 0 && y > 0

public int gcd(int x, int y) {
   while (x != y) {
      if (x > y)  // if x > y > 0 then gcd(x, y) == gcd(x - y, y)
         x = x - y;
      else        // if y > x > 0 then gcd(x, y) == gcd(x, y - x)
         y = y - x;
   }
   return x;
}
```

```
// precondition: x > 0 && y > 0

public int gcd(int x, int y) {
   if (x == y)
      return x;
   else if (x > y)
      return gcd(x - y, y);
   else
      return gcd(x, y - x);
}
```

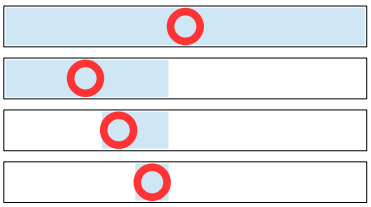
Power

```
// precondition: n >= 0

public int pow(int x, int n) {
   int p = 1;
   while (n > 0) {
      if (n % 2 == 0) {  // if n even then x^n == (x*x)^(n/2)
         n /= 2;
         x *= x;
      } else {           // if n odd then x^n == x * x^(n-1)
         n -= 1;
         p *= x;
      }
   }
   return p;
}
```

```
// precondition: n >= 0

public int pow(int x, int n) {
   if (n == 0)
      return 1;
   else if (n % 2 == 0)
      return pow(x * x, n / 2);
   else
      return x * pow(x, n - 1);
}
```

Binary search

- Assuming a sorted array (`list`), from small to large values
- Comparing any element with given value (`v`)
  - If larger, then not after
  - If smaller, then not before



```
// precondition: sorted list

public boolean binarySearch() {
   int a = 0, b = list.length - 1;
   while (a < b) {
      int m = (a + b) / 2;
      if (v <= list[m])  // v smaller or equal
         b = m;
      else               // v larger
         a = m + 1;
   }
   return (list[a] == v);
}
```

```
// precondition: sorted list

public boolean binarySearch() {
   return binarySearchRec(0, list.length-1);
}

private boolean binarySearchRec(int a, int b) {
   if (a == b)
      return (list[a] == v);
   else {
      int m = (a + b) / 2;
      if (v <= list[m])
         return binarySearchRec(a, m);
      else
         return binarySearchRec(m+1, b);
   }
}
```