

Algorithms and Data Structures

DBB213

Erik van der Spek

TU **e**

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Functions and datastructures lecture



Datastructures

- Arrays

- e.g. `int[] billy;`



billy[0] = ...

- Nested arrays

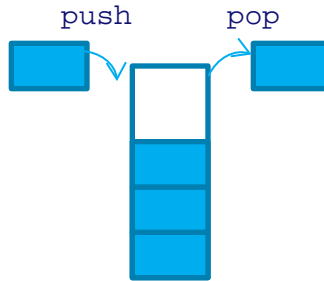
- e.g. `int[][] billy;`



billy[0][2] = ...

(other datastructures from Java)

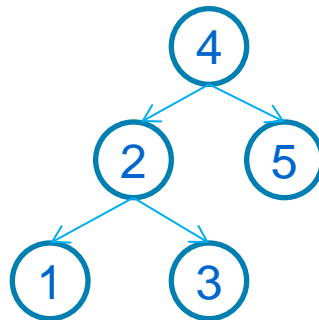
- Stack



- LinkedList



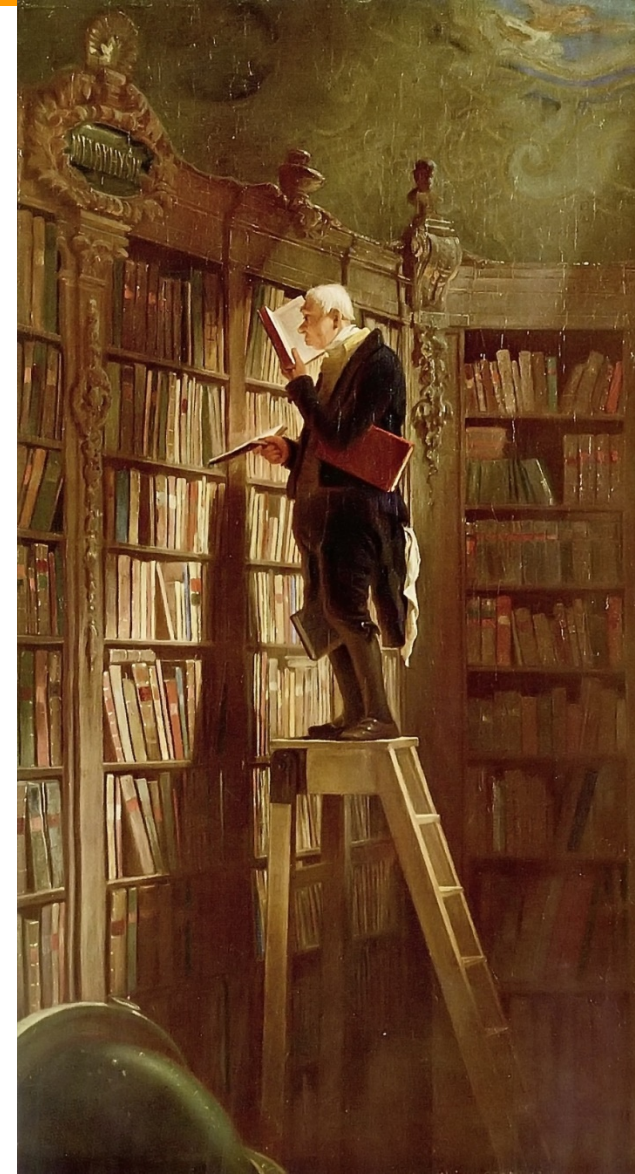
- TreeMap



..and many more

Operating on the datastructure

- Placing, retrieving and sorting
- `billy[3] = "pikachu";`
- Loops:
 - `for(int i =0; i<billy.length; i++)`
 - `while (i<billy.length) {`
 `...`
 `i++;`
 `}`



Content

- General principles
- Calculating
- Searching
- Optimising
- Sorting

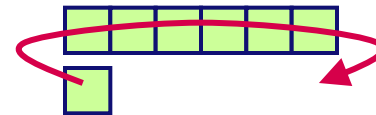
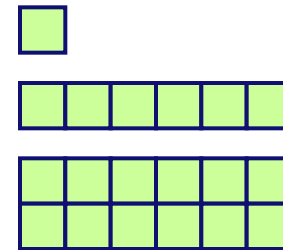
- Generating & testing

General principles

- Data structures
 - built from variables and arrays
- Algorithms
 - built from if, if-else, for, while, functions calling each other
- To be designed together
 - like the railway topology and the train schedule
 - you cannot design an efficient train schedule
 - unless you know the railway topology

General principles (continued)

- Calculating
 - simple variables
 - or one-dimensional memory
 - or two-dimensional memory
- Searching
 - one-dimensional problems
 - with single-loop solutions
- Optimising
 - one-dimensional problems
 - with single-loop solution and memory
- Sorting
 - one-dimensional problems
 - with nested-loop solutions



General principles (continued)

- Issues
 - correctness
 - efficiency
- Correctness
 - respecting array bounds
 - initialising all variables
 - careful reasoning
- Efficiency
 - data structures for memorization
 - $\mathcal{O}(n)$ is better than $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$
 - use existing classics

Calculating

- Basic idea
 - intermediate results in variable
 - while-loop or for-loop for repetition

```
int sum=0;  
int n=11;  
int i=0;
```

```
while (i < n) {  
    sum = sum + i;  
    i++;  
}  
println(sum);
```



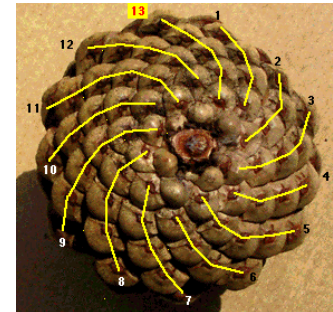
1+2+3+4+5+6+7+8+9+10 = ???

Calculating (continued)

- Basic idea
 - intermediate results in array
 - while-loop or for-loop for repetition

```
int[] fibo = new int[10];
fibo[0]=0;
fibo[1]=1;

int i = 2;
while (i < 10){
    fibo[i] = fibo[i-1] + fibo[i-2];
    println(fibo[i]);
    i++;
}
```



jwilson.coe.uga.edu



Calculating (continued)

- Basic idea
 - intermediate result in array of arrays
 - while-loop or for-loop for repetition

```
int[][] pas = new int[10][10];  
int n=0;  
int k=0;
```



[commons.wikimedia.org/wiki/
File:Blaise_pascal.jpg](https://commons.wikimedia.org/wiki/File:Blaise_pascal.jpg)

Calculating (continued)

```
n=0;
k=0;

pas[n][k]=1;
n=1;
while (n < 10) {
    k=0;
    pas[n][k]=1;
    k=1;
    while (k < n){
        pas[n][k]=pas[n-1][k-1] + pas[n-1][k];
        k++;
    }
    pas[n][k]=1;
    n++;
}
```

Calculating (continued)

```
for (int i=0; i<10; i++) {  
    for (int j=0; j<i+1; j++) {  
        print(pas[i][j]);  
        print("  ");  
    }  
    println();  
}
```



```
1  
1 1  
1 2 1  
1 3 3 1  
1 4 6 4 1  
1 5 10 10 5 1  
1 6 15 20 15 6 1  
1 7 21 35 35 21 7 1  
1 8 28 56 70 56 28 8 1  
1 9 36 84 126 126 84 36 9 1
```

Calculating (continued)

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```


(recap)

- General principles
- Calculating
- Searching
- Optimising
- Sorting
- Generating & testing


Searching

- Basic idea
 - data in one array
 - scan the entire array with one for-loop
- Basic code

```
int[] a=new int[N];  
// a is filled here  
for (int i=0; i<N; i++) {  
    if (a[i] == x) {  
        // aha, x found  
    }  
}
```

Searching (continued)

```
int[] a=new int[5];  
a[0]=3;  
a[1]=7;  
a[2]=8;  
a[3]=4;  
a[4]=7;
```

```
int k=-1;  
for (int i=0; i<5; i++) {  
    if (a[i] == 4) {  
        println("aha ");  
        k=i;  
    }  
}  
print(k); 
```

Searching (continued)

- Tricks of the trade
 - for an array of N elements
 - the indexes run from 0 to $N-1$
 - use a local loop counter named i
 - store where it is found in another variable (here: k)
 - idea: use a boolean found to stop once your x is found

Searching (continued)

```
//array as before
```

```
int k=-1;
boolean found = false;
int i=0;
while (i<5 && !found){
    if (a[i] == 4) {
        found=true;
        k=i;
    }
    i++;
}
print(k);
```



(recap)

- General principles
- Calculating
- Searching
- Optimising
- Sorting
- Generating & testing

Optimising

- Basic idea
 - data in one array
 - keep track of *optimal value so far*
 - scan the entire array with one for-loop
- Tricks of the trade
 - initialise with a dummy value like minus “infinity”

```
int[] a=new int[5];  
a[0]=3;  
a[1]=7;  
a[2]=8;  
a[3]=4;  
a[4]=7;
```


Optimising (continued)

```
int[] a=new int[5];
```

```
//data as before
```

```
int mymax = -1000;
```

```
int at = -1;
```

```
for (int i=0; i < 5; i++){
```

```
    if (a[i] > mymax) {
```

```
        mymax = a[i];
```

```
        at = i;
```

```
    }
```

```
}
```

```
print(mymax);
```

```
print(" at ");
```

```
print(at);
```

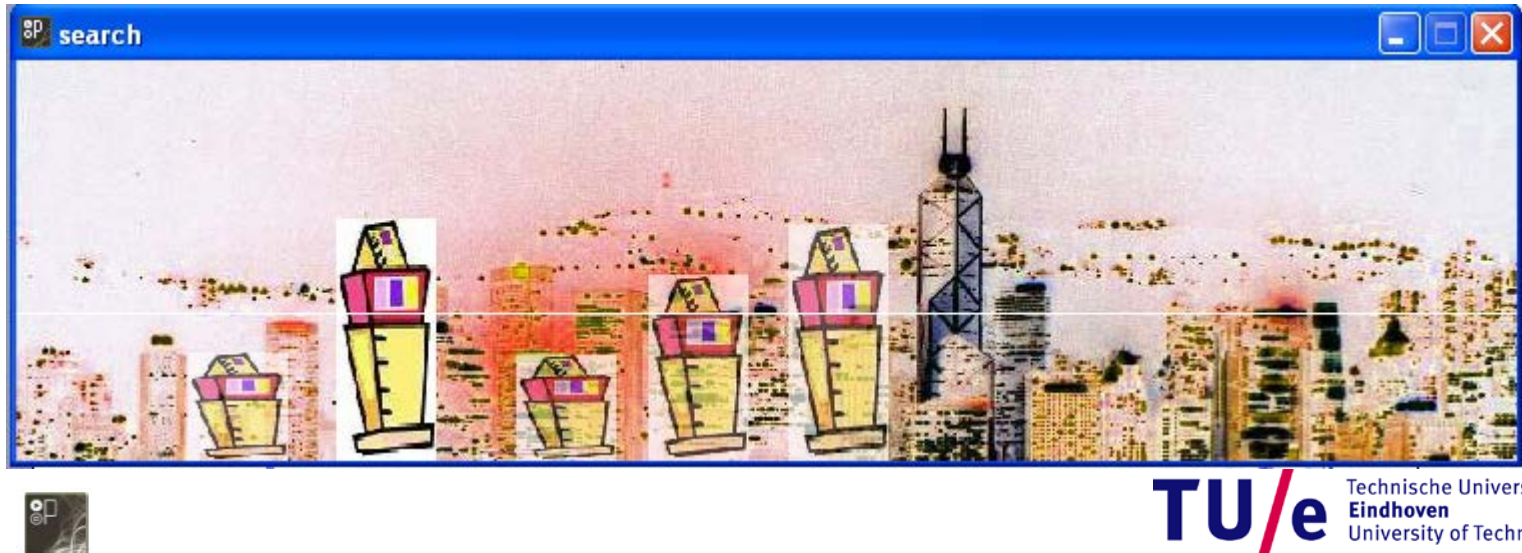


(recap)

- General principles
- Calculating
- Searching
- Optimising
- Sorting
- Generating & testing

Calculating and optimising (demo)

- Demo application
 - edit an array of towers
 - random height levels
 - calculate average
 - find the tallest



Calculating and optimising (demo)

- Interface of the demo

The construction truck is at horizontal position x

```
void keyPressed() {  
    if(keyCode == LEFT) { x=x-2; }  
    if(keyCode == RIGHT) { x=x+2; }  
    if(key == ' ') { // add tower  
        if (towers<maxtowers){  
            loc[towers]=x;  
            lvl[towers]=floor(random(0,200));  
            towers++;  
        }  
    }  
    if(keyCode == ENTER) { done=true; }  
}
```

For each tower index there is a location loc and a height level lvl

Calculating and optimising (demo)

- Action of the demo

```
// in draw
if (done) {
    int i=tallest();
    tint(255,255);
    image(tower,loc[i],200-lvl[i],50,lvl[i]);
    int a=average();
    stroke(255);
    line(0, 200-a, 750, 200-a);
}
```

Finding a maximum

Calculating an average

Calculating and optimising (demo)

- Finding a maximum

```
int tallest() {  
    int m = -1; // max level found so far  
    int i = -1; // index where it's found  
    for (int j=0; j<towers; j++)  
        if (lv1[j] > m) {  
            m = lv1[j];  
            i = j;  
        }  
    return i;  
}
```

Calculating and optimising (demo)

- Finding a maximum

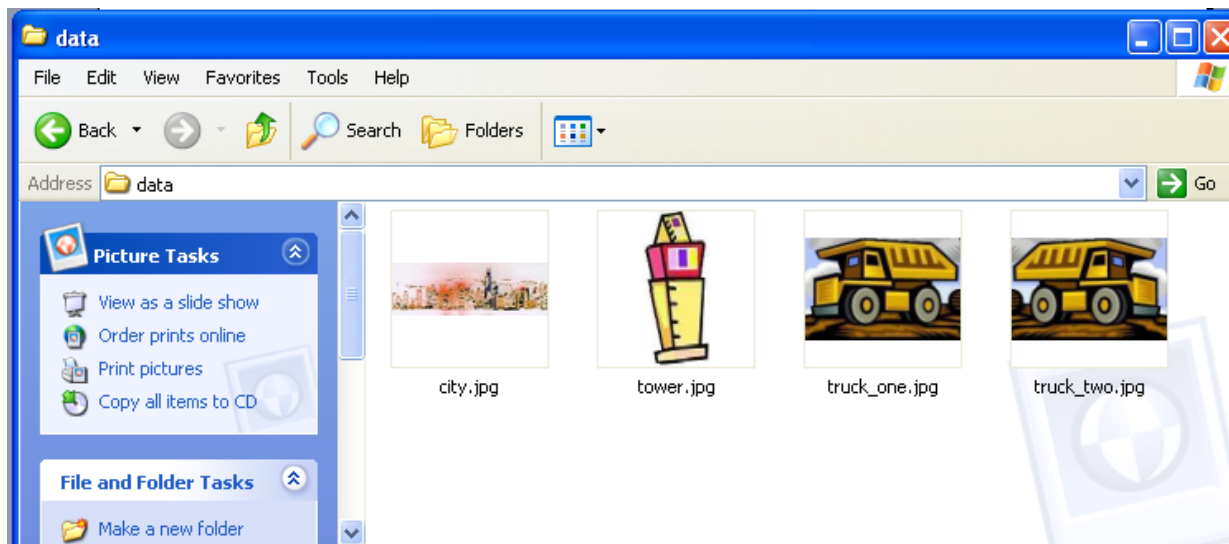
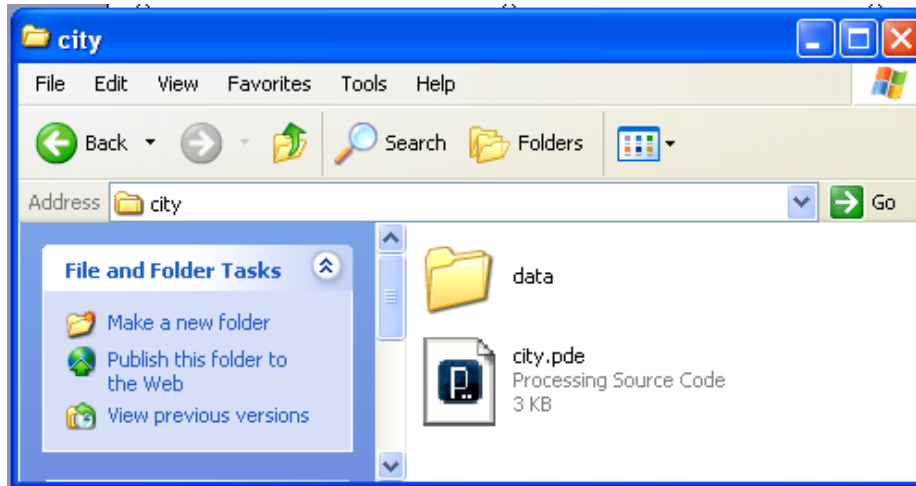
```
int tallest() {  
    int m = -1; // max level found so far  
    int i = -1; // index where it's found  
    for (int j=0; j<towers; j++)  
        if (lvl[j] > m) {  
            m = lvl[j];  
            i = j;  
        }  
    return i;  
}
```


Calculating and optimising (demo)

- Calculating an average

```
int average() {  
    int s=0; // sum calculated so far  
    for (int j=0; j<towers; j++)  
        s = s + lvl[j];  
    if (towers > 0)  
        return s / towers;  
    else return 1;  
}
```

Calculating and optimising (demo)



(recap)

- General principles
- Calculating
- Searching
- Optimising
- Sorting
- Generating & testing

Sorting

- Basic idea
 - keep the data in the array
 - move them into increasing order
 - at least two nested loops are necessary

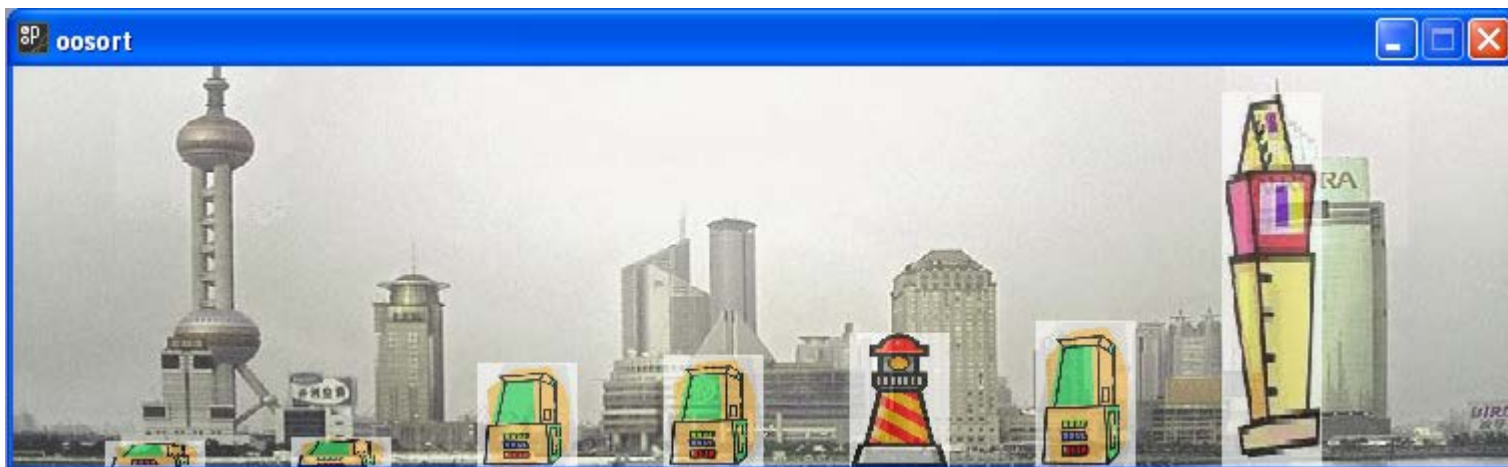
```
int[] a=new int[5];  
a[0]=3;  
a[1]=7;  
a[2]=8;  
a[3]=4;  
a[4]=7;
```

[3 , 7 , 8 , 4 , 7]



[3 , 4 , 7 , 7 , 8]

Sorting (continued)



Sorting (continued)

- Selection sort
 - noted for its simplicity
 - see e.g. wikipedia, selection sort
 - more: heapsort, insertion sort, bubble sort, quicksort, Shellsort, ...

```
int towers=5;  
int[] lvl = new int[towers];
```

```
lvl[0]=3;  
lvl[1]=7;  
lvl[2]=8;  
lvl[3]=4;  
lvl[4]=7;
```

Sorting (continued)

```
//selectionSort
for (int i = 0; i < towers; i++) {
    // now i smallest values are sorted in 0..i-1
    int mi = i;
    for (int j = i; j < towers; j++) {
        // now mi is index of smallest in i..j-1
        if (lvl[j] < lvl[mi])
            mi = j;
    }
    // swap for indexes i and mi
    int tmp = lvl[i];
    lvl[i]   = lvl[mi];
    lvl[mi]  = tmp;
}
```


Sorting (continued)

- Demo application
 - edit an array of towers
 - three tower types
 - towers as objects
 - sort by level

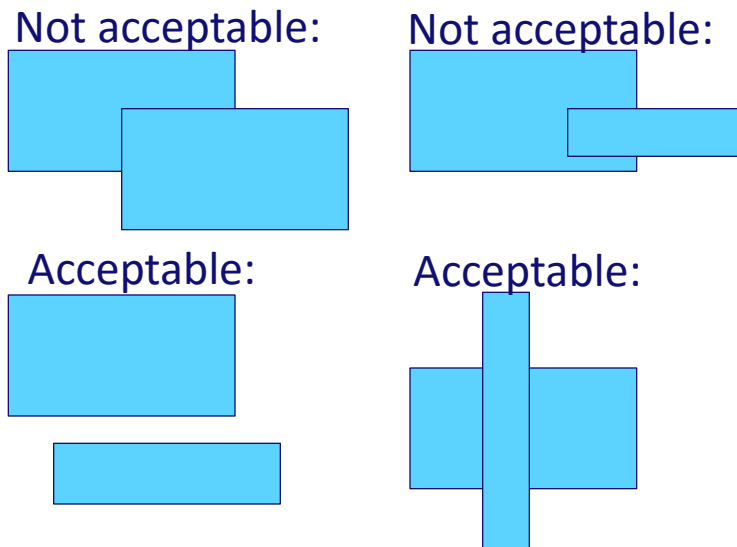


(recap)

- General principles
- Calculating
- Searching
- Optimising
- Sorting
- Generating & testing

Generate and test

- Situation: when it is difficult to produce perfect solutions in one step.
- Solution: generate many candidate solutions and test them afterwards.
- Case study: generate a collection of rectangles whose corners do not intersect.



- Inspiration: clusters of rectangles in Malevich's abstract art.
- Situation: one cluster of same-color rectangles should not corner-intersect

Generate and test

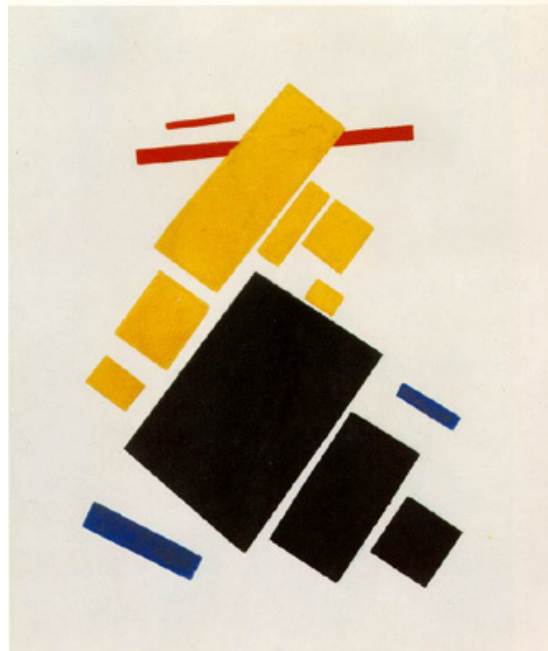
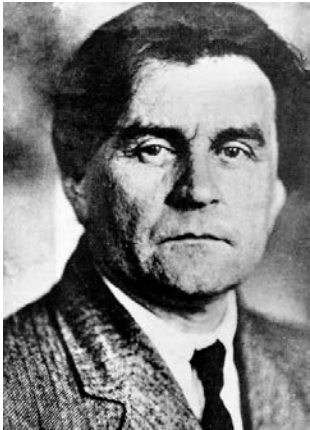
- Situation: when it is difficult to produce perfect solutions in one step.
- Solution: generate many candidate solutions and test them afterwards.
- Basic code:

```
while (not acceptable){  
    Generate a candidate solution  
    Test the candidate solution  
}  
//hurray, acceptable solution found
```



Generate and test: Inspiration Malevich

- Malevich (Малевич) 1879–1935, Russian painter and art theoretician. Pioneer of geometric abstract art. Started Suprematist movement



Malevich' work 1915



Mondrian's work 1918

- Algorithm: Mirjam E. Haring. Malevich, computational simulation of an abstract artistic style, kunstmatige intelligentie, university of amsterdam, (2011)

Generate and test: how to make a cluster of rectangles which do not corner-intersect?

```
void myCluster(int cx, int cy){
    mem = 0;
    int successes = 0;
    int howmany = int(random(15));
    while (successes < howmany) {
        //generate a rectangle
        int y = cy + int(random(-100,100));
        int x = cx + int(random(-150,150));
        int w = int(random(200));
        int h = int(random(100));
        //and test it
        if (acceptable(x,y,w,h)) {
            rect(x,y,w,h);           //draw it
            memo(x,y,w,h);           //memorize it
            successes++;
        }
    }
}
```

Keep track of all rectangles already accepted in this cluster so new candidates can be compared for acceptability

Generate and test: how to make a cluster of rectangles which do not corner-intersect?

```
void myCluster(int cx, int cy){
    mem = 0;
    int successes = 0;
    int howmany = int(random(15));
    while (successes < howmany) {
        //generate a rectangle
        int y = cy + int(random(-100,100));
        int x = cx + int(random(-150,150));
        int w = int(random(200));
        int h = int(random(100));
        //and test it
        if (acceptable(x,y,w,h)) {
            rect(x,y,w,h);           //draw it
            memo(x,y,w,h);           //memorize it
            successes++;
        }
    }
}
```

Generate and test: how to keep track of a collection of rectangles?

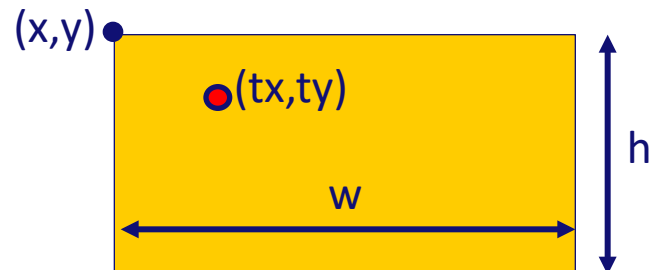
```
//memory bank to store existing rectangles
int memx[] = new int[100];
int memy[] = new int[100];
int memw[] = new int[100];
int memh[] = new int[100];
int mem = 0;

void memo(int x, int y, int w, int h){
    //memorize x,y,w,h as the mem-th rectangle
    memx[mem] = x;
    memy[mem] = y;
    memw[mem] = w;
    memh[mem] = h;
    mem++;
}
```

mem is the number of rectangles in memory so far; the memory consists of four arrays, one for x values, one for y values, one for widths, one for heights.

Generate and test, how to test whether a (corner-)point is inside another rectangle?

```
boolean inside(int tx, int ty,  
               int x, int y, int w, int h){  
  
    //test whether a point tx,ty  
    //is inside a rectangle (x,y,w,h)  
  
    if ((tx >= x)&&(tx <= x+w)&&(ty >= y)&&(ty <= y+h))  
        return true;  
    else return false;  
}
```



Generate and test, how to test whether two rectangles do not corner-intersect?

```
boolean accept(int x1, int y1, int w1, int h1,  
               int x2, int y2, int w2, int h2) {  
  
    //test for two rectangles that no corner of one is inside the other;  
    //they may still overlap as one can cross entirely through the other  
  
    return !inside(x1,y1,x2,y2,w2,h2)  
           && !inside(x1+w1 ,y1      ,x2,y2,w2,h2)  
           && !inside(x1      ,y1+h1 ,x2,y2,w2,h2)  
           && !inside(x1+w1 ,y1+h1 ,x2,y2,w2,h2)  
           && !inside(x2      ,y2      ,x1,y1,w1,h1)  
           && !inside(x2+w2 ,y2      ,x1,y1,w1,h1)  
           && !inside(x2      ,y2+h2 ,x1,y1,w1,h1)  
           && !inside(x2+w2 ,y2+h2 ,x1,y1,w1,h1);  
  
}
```

One corner point of
(x1,y1,w1,h1) could be inside
(x2,y2,w2,h2), perhaps

One corner point of
(x2,y2,w2,h2) could be inside
(x1,y1,w1,h1), perhaps

Generate and test, how to test a candidate rectangle against the entire memory?

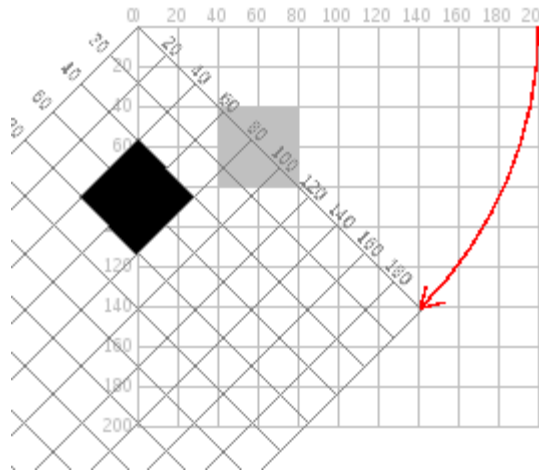
```
boolean acceptable(int x, int y, int w, int h){  
  
    //test for a new rectangle whether it is  
    //acceptable against all others in memory  
  
    boolean okay = true;  
    for (int i=0; i<mem; i++) {  
        if (!accept(x,y,w,h,memx[i],memy[i],memw[i],memh[i]))  
            okay = false;  
    }  
    return okay;  
}
```

retrieve (x,y,w,h) quadruple from memory
location number mem

Generate and test: how to make several such clusters?

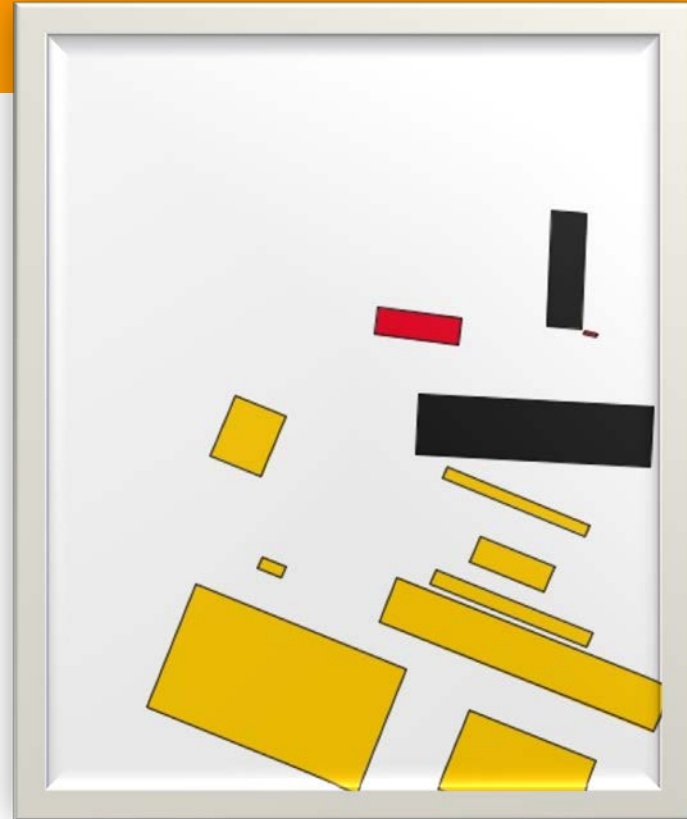
```
void myArt(){
  int howmany = 1 + int(random(4));
  for (int j=0; j<howmany; j++){
    int cx = width/2 + int(random(-width/4,width/4));
    int cy = height/2 + int(random(-height/4,height/4));
    myColor(j);
    pushMatrix();
    rotate(radians(int(random(40))));
    myCluster(cx, cy);
    popMatrix();
  }
}
```

so later in the abstract art work, each cluster has its own characteristic angle of rotation





Computer Generated 2012



Computer Generated 2012



Kasimir Malevich, Suprematist
Painting: Airplane Flying, 1915.



Kasimir Malevich, Suprematist
Painting: eight red rectangles,
1915.

Generate and test: putting things together

```
void setup(){
    size(400,500);
    background(220);
    myArt();
}

void myColor(int i){
    color c = color(0,0,0);
    switch (i % 5) {
        case 0: c = color(20,20,20); break; //black
        case 1: c = color(200,0,25); break; //red-ish
        case 2: c = color(220,175,0); break; //yellow-ish
        case 3: c = color(0,0,150); break; //blue-ish
        case 4: c = color(0,125,0); break; //green-ish
    }
    stroke(0);
    fill(c);
}
```

Be an artist!

