Algorithms and Data Structures

DG200 team (contribution Loe Feijs)

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Content

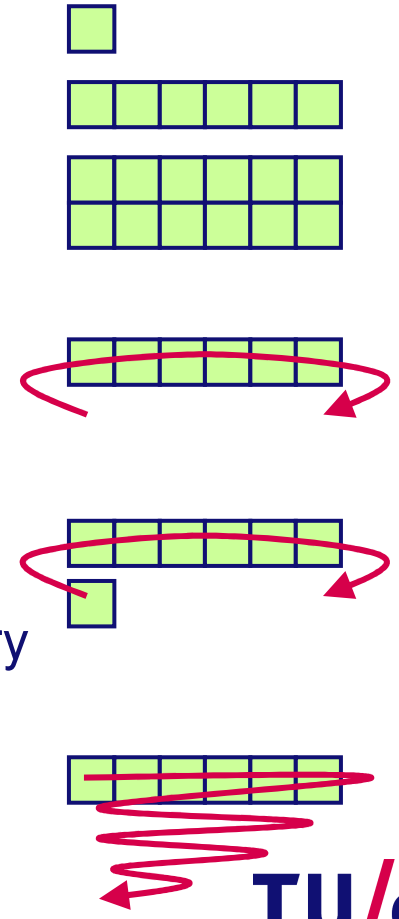
- General principles
- Calculating
- Searching
- Optimising
- Sorting

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 memoization
  - $\mathcal{O}(n)$ 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);
```

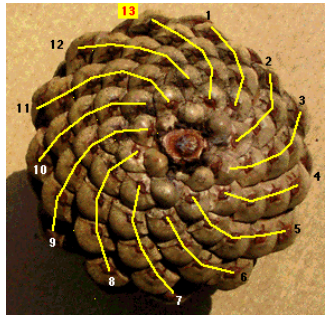
1+2+3+4+5+6+7+8+9+10 = ???

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

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
```

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);
```

Exercise on searching

- Do it yourself
  - write a program that takes an array with five numbers between zero and 100 and finds a given value x (if x is in the array). This is like the given program, but now we add the demand that the program should start searching at the end, working backwards.

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);
```

Exercise on optimising

- Do it yourself
  - write a program that takes an array with five numbers between zero and 100 and find the smallest value. This is like the given program, but now we want the minimum, not the maximum.

(recap)

- General principles
- Calculating
- Searching
- Optimising
- Sorting

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

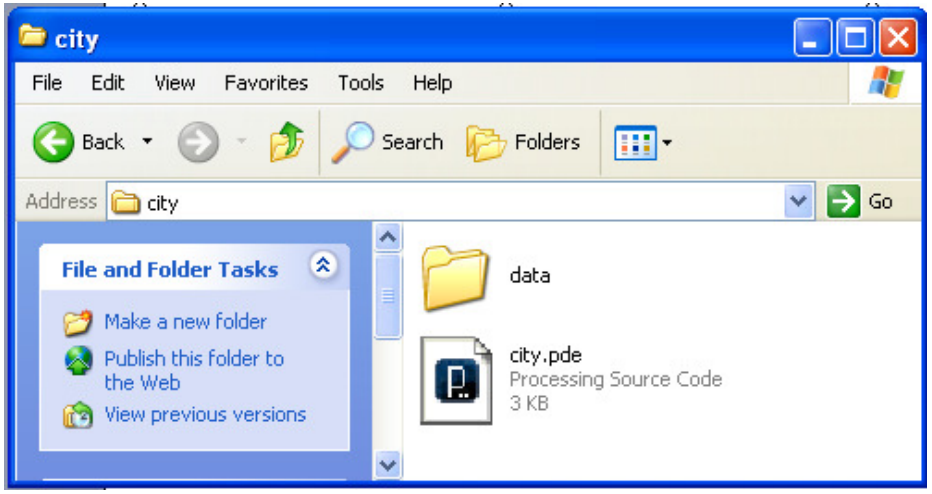
```
int tallest() {
    int m = -1; // max level found so far
    int i = -1; // index where it's found
    for (int j=0; j<towers; j++)
        if (lvl[j] > m) {
            m = lvl[j];
            i = j;
        }
    return i;
}
```

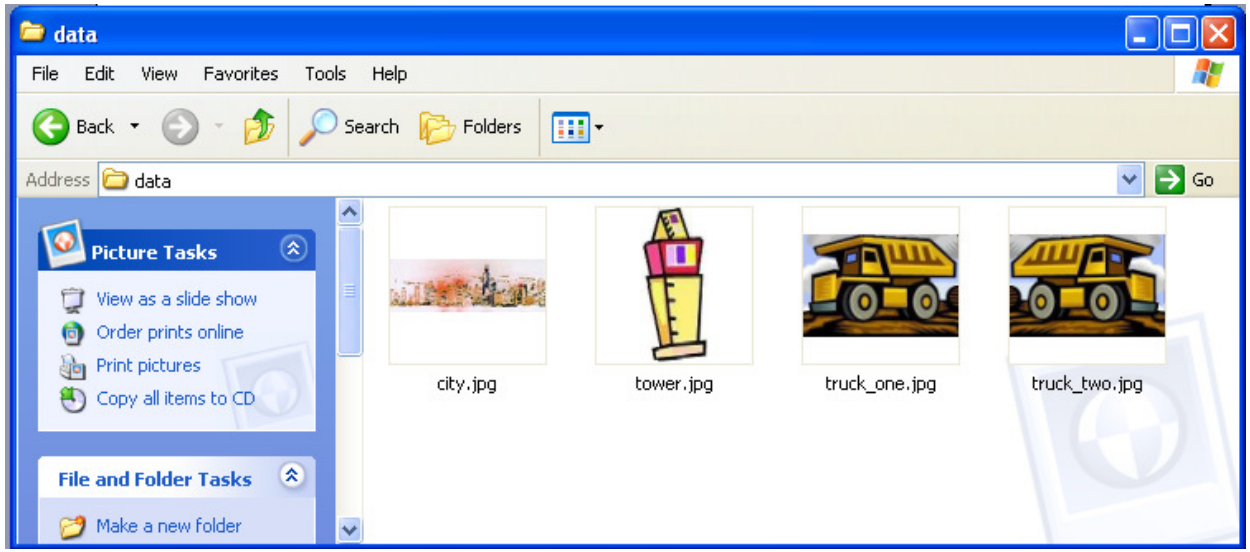
Calculating and optimising (demo)

- 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)

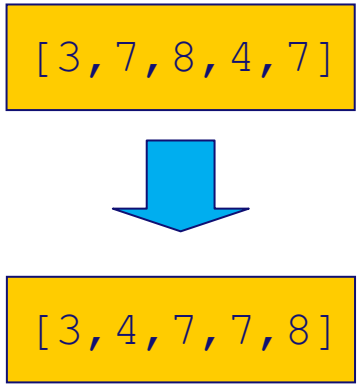




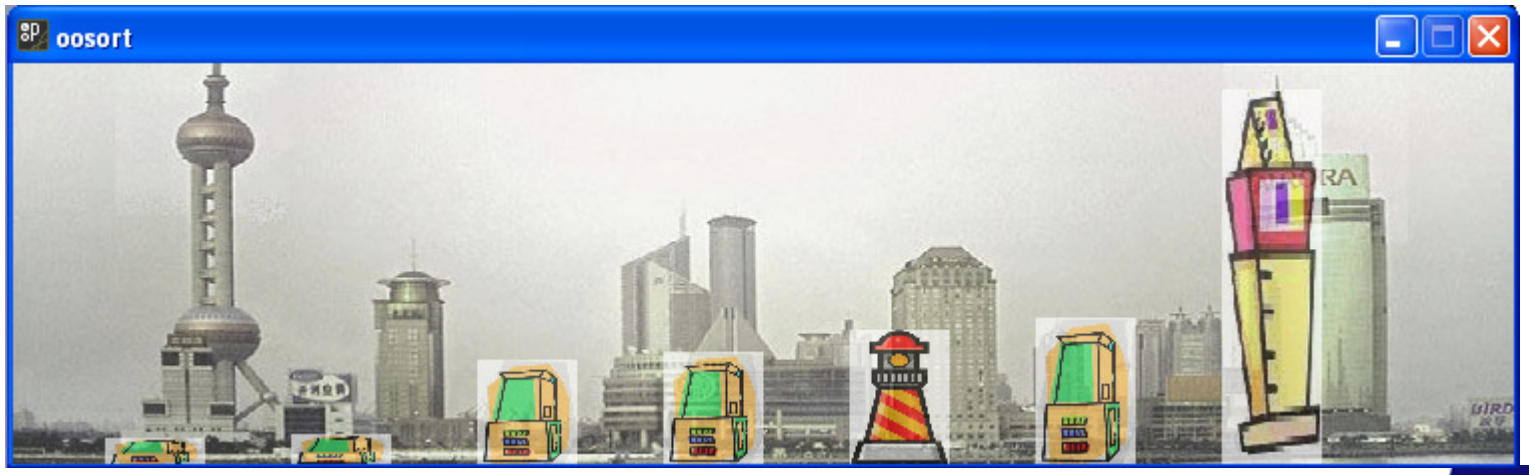
Sorting

- 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;
```



Sorting (continued)



sort

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

Exercise on sorting

- Do it yourself
  - write a program for bubble sort. Begin by reading, for example wikipedia: bubble sort