# What remains?

- Median in linear time,
- randomized algorithms (Monte Carlo, Las Vegas),
- other programming languages.

# Median in linear time

## in fact not just median

- If we wanted the algorithm finding only median, we fail...
- ... thus we generalize the problem:
- We are looking for $k$th smallest element (at the beginning $k = \frac{n}{2}$.
- This algorithm may be used for improving, e.g., quicksort (for pivot-choice).
- Note that if we pick median in linear time, we may implement quicksort with complexity $\Theta(n \log n)$.

# The algorithm itself

- Divide the input into 5-tuples,
- find median in each 5-tuple,
- find median of medians,
- use it as a pivot (divide the input into 2 piles),
- look for the element on the appropriate pile.

# Details of the algorithm

- Details: Let the pile of smaller elements have $l$ elements,
- if $l > k$ then look for $k$th smallest on the pile of smaller elements,
- if $l = k - 1$, pivot is the appropriate element,
- otherwise look for $k - l - 1$st smallest on the pile of bigger elements.
- How to find the medians of 5-tuples?
- by brute force. How about median of medians?
- Recursion (we don't have time for brute force).
- How about searching on a smaller pile?
- Recursion again!

# Analysis

- Let us denote complexity $T(n)$.
- We show that each pile has size at most $\frac{7n}{10}$ (by picture).
- Now we see that:
- $T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$.
- It suffices to show that there exists $k$ s.t. $T(n) \leq kn$.

# Randomized algorithms

- Not all algorithms are deterministic, some of them are using randomness.
- Randomness (pseudorandomness) gets generated in C# using instances of class `Random`. This generator is *not* random, it is pseudorandom.
- Algorithms that use randomness are called randomized.
- There are two principal families of these algorithms: Monte Carlo and Las Vegas.

# Monte Carlo algorithms

- These algorithms find quickly some candidate for solution (called *incumbent*) but they hardly finish.
- Example: Determine the area of a unit circle (i.e., estimate $\pi$).
- We are generating pairs of numbers from a unit square and we test whether the point lies inside the unit circle (quarter of circle). Are of the circle is $4 \cdot \frac{\text{number of successful points}}{\text{number of attempts}}$ because area of unit square is 1, we picked $1/4$ of the circle and 3 remaining quadrants are symmetric.
- We get a bound on $\pi$ after the first attempt (0 or 4). The precision should increase with number of attempts.
- This algorithm relies on Law of large numbers (average converges to expected value).

# Monte Carlo Algorithms

...for the second time

- Further algorithms from this family are: Simulated annealing or Genetic algorithms.
- Advantage: We may stop whenever we want (and we have an incumbent). Disadvantage: These algorithms do not finish.

# Las Vegas algorithms

- This family was originally defined for the algorithms solving problems to the optimum using randomness to defeat the worst case.
- Example: Randomized quicksort (pick random element as a pivot).
- Later approximation algorithms started being considered also members of this family – if they use randomness and after they finish, they may be just started over to get a better result.
- Example (randomized rounding): Considering a problem of linear integer programming we use (nonintegral) linear programming and non-integral values get rounded using "randomized rounding". Once the rounding is over, we have a solution.

# More on these algorithms

- Matoušek, Valtr,... Probabilistic method,
- Sgall: Randomized algorithms, Approximation and online-algorithms.

# Further programming languages

look in a very similar way

- C programming language,
- C++,
- Java,
- PHP, Javascript,...

# C programming language

... at the beginning there was language A, then B and after that C

- Designed as a modification of Pascal,
- it uses a syntax familiar to us.
- It does not implement objects, functions are identified by names (no function overloading),
- there is no type `string`, pointers have to be used instead.
- For dereference a unary (prefix) asterisk operator is used,
- memory is allocated calling function `malloc` and freed by `free`. Function `malloc` takes as a parameter number of allocated bytes. Data-types can be easily typecasted (pointer to long int and back...).

# C programming language

## for the second time

- Array is just disguised pointer, thus string can be represented using pointer at char.
- Everything is just disguised integer (long), there is no boolean, use integer instead, thus: if(a=b)...
- The entry point of the program is function `main`.
- Arguments are always passed by value (passing by reference may be worked around using pointers).

# Examples in C

```
#include <stdio.h>
#include <malloc.h>
int main(int argc, char*argv[])
{    int a=1;
     while(a<=argc) printf("%s\n ",argv[a++]);
}
-----------------------------
a=malloc(strlen(b)+1);
while(*a++=*b++); //strcpy
```

# Overall about C programming language

- Some people consider the language to be obsolete,
- however, everything important is written in C.
- Programmer has full control over all resources,
- thus it is necessary to be responsible and avoid incompetent type-casting, shooting into memory and also nobody checks whether the arguments are matching.
- There are no classes, there are structures and unions.
- When using structures, part of the data-type-name is also the word `structure` and `union`, respectively.

# Overall about C II

- There are no generic data-types (and even no templates), there are macros instead.
- Example: `#define MAX(a,b) (a>b?a:b)` – macro is preprocessed before compilation with all unpleasant consequences:
- `MAX(f(x),g(x))` calls function returning larger result twice!

# C++

- In general, C++ is an extention of C thus most stuff written in C works also in C++, just...
- C++ has stronger control over type-casting, it tests parameters of called functions and it implements objects.
- Almost anything known in other languages can also be found in C++ (including multiple inheritance $\Rightarrow$ no need for interfaces).
- It looks similarly to C# up to particular exceptions:
- `abstract int f();` $\Rightarrow$ `virtual int f()=NULL;`
- `override void f()...` $\Rightarrow$ `virtual void f()...`
- `abstract class c...` $\Rightarrow$ `class c` (compiler is not as stupid to miss the fact that the class contains purely virtual function).

# Overall on C++

- C++ does not force us to use the objects, thus there are member- and friend-functions. After defining a function in the class-definition we may declare this function to be a "friend" (and thus it may access private items).
- Member-functions are defined using four-dot-operator: `int classname::functionname(int parameters)...`
- Instead of interfaces we use abstract classes and multiple inheritance (thus we do not use `implements` keyword).
- Templates (ancestor of generic data-types) do not have to be parametrized only by data-types (but also, e.g., by a number).
- There is no garbage-collector, what gets allocated using `new` has to be deallocated using `delete` keyword (thus it makes sense to define destructors).

# Java

- The language so far the most similar to C# (including the fact that it is interpreted language).
- There are several milliards of devices (usually programmed in C) around the world that are able to interpret Java.
- Syntactially very similar just without namespaces.
- Public class can be defined only in the file with the same name as the class has.
- Programs are not always accessed through static method `main`,
- library-functions are named differently (`System.Console.WriteLine` -> `System.out.println`).
- Inheritance is denoted using keyword (`extends`) instead of semicolon,
- `using` $\Rightarrow$ `import`.

# Java application example

## doing nothing

```
class apple{
      public static void main(String args[])
      {    System.out.println("Nothing!");
      }
}
```

# Further use of Java

not only applications can be written in Java

- Applet – code written in Java used on web-pages,
- MIDlet – code interpretted using mobile-phones (now obsolete),
- Android applications – there are several possibilities.
- Common behavior: Always it is necessary to define the appropriate son of the appropriate class,
- enigmatic interface equipped by particular traps (e.g., code of MIDlet must not exceed 32 kB, there is approximately 2MB heap, display is redrawn under cryptic circumstances, in particular implementations of Java virtual machine there used to be errors).
- MIDlets and Android applications usually require a different compiler (than the applications and applets).

# Applet

```
import java.applet.*; //Everyone must include
Applet-classdef
import java.awt.*; //Who wants to draw on display?
public class apple extends Applet{
     public void init(){}
     public void stop(){}
     public void paint(Graphics g)
     {    g.drawString("Ha!",20,20);
          g.drawString("What means your
'Ha!'?",20,40);
     }
}  //Must be in apple.java (as it is a public class)!
//It is necessary to define init method.
//The paint-method is called when redrawing the window.
//When the program runs, the window is *not* redrawn
```

# MIDlet I/II

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;
public class strange_game extends MIDlet {
     protected void startApp() throws
                    MIDletStateChangeException {
          Display display = Display.getDisplay(this);
          display.setCurrent((
                    new GameMidletScreen(this)) );
     }
...
```

# MIDlet II/II

```
public class strange_game extends MIDlet {
     protected void startApp() throws
                    MIDletStateChangeException {
          Display display = Display.getDisplay(this);
          display.setCurrent((
                    new GameMidletScreen(this)) );
     }
     protected void destroyApp(boolean unconditional)
               throws MIDletStateChangeException {}
     public void exit()
     {    try{destroyApp(false);}catch(Exception e){}
          notifyDestroyed();
     }
     protected void pauseApp(){}
}
```

# Android application

```
package pytel.com;

import android.app.Activity;
import android.os.Bundle;

public class AnthropoidActivity extends Activity {
     /** Called when the activity first created. */
     @Override
     public void onCreate(Bundle savedInstanceState){
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);
     }
}
```

# Chalk the following up to experience

- When we know general syntax of a family of programming languages, programming represents:
- Algorithm-design (teoretical topic – in fact mathematics).
- Interface-design and implementation (rituals with uncertain restrictions).
- Application-kernel implementation (praktical work almost the same in all languages).
- Unpleasant part is represented by input/output programming, after designing input and output whoever (who is generally able of programming) must succeed.

# Conclusion

- Computer-science is not just programming,
- however, programming-knowledge is necessary for understanding particular topics in computer-science
- except of this, programming is relatively well (financially) honoured,
- however, it does not need to be the case for too long time. Thus it is good idea to understand even theory!

That's all from me...
...except of exercises from last lecture!

Thank you for your attention.