Reasons to study Data Structures & Algorithms

- Provide abstraction
- Handling of real-world data storage
- Programmer's tools
- Modeling of real-world objects and concepts
- Programs more readable, understandable, maintainable
- If you have a good feeling for them, you are a better programmer!
  - You can solve the problem faster
  - Your program works better, with fewer bugs
  - You will know when you have been given an impossible task

Considerations

- Speed
- Memory usage
- Complexity of structure

Examples of tradeoffs:

- Insertion speed vs. search speed vs. deletion speed
- Structure vs. speed

Your task as a programmer: think of the tradeoffs and select the best option

Complexity

- Expresses the efficiency of an algorithm
- Running time expressed as function of problem size
- Problem size: number of inputs
- Some complexity classes:
  - Logarithmic
  - Linear
  - Quadratic
  - Exponential

“Big O” notation

- Function g(n) is O(f(n)) if there are positive constants c and n_0 such that $g(n) \leq c*f(n)$ for all $n \geq n_0$
- Upper bound on rate of growth of an algorithm
- Determines its “cost” in terms of time to run

Complexity Examples 1 and 2

```
for(i = 0; i < n; i++) {
  b[i] = a[i]+1;
  c[i] = a[i]+b[i];
}
```

```
for(i = 0; i < n; i++) {
  for(j = 7; j < n; j=j+3) {
    b[i] = a[i]+1;
  }
  c[i] = a[i]+b[i];
}
```

Complexity Example 3

```
for(i = 0; i < n; i++) {
  if(a[i] > 0) {
    for(j = 0; j < n; j++)
      c[i] = c[i] + b[j];
  }
  else
    c[i] = b[i];
}
```

Question: what if the outer loop were changed to:

`for(i = 1; i < n; i=i*2)`?

Complexity Example 4

```
int xyz(int n) {
  if(n <= 1)
    return 1;
  else
    return xyz(n/2) + n;
}
```

Question: what if the last statement were changed to

```
 return xyz(n-2) + n; ?
```

Complexity Example 5

```
int bc(int n)
{ int c;

  c = 0;
  while(n > 0)
  {
    c = c + (n % 2);
    n = n / 2;
  }
  return c;
}
```

Complexity Example 6

```
int pc(int n)
{ int i, c;

  if(n == 1) {
    return 1;
  }
  c = 0;
  for(i = 0; i < n; i++) {
    c = c + pc(n-1);
  }
  return c;
}
```

Table of Computational Complexity

If a step takes 0.01 ms and the problem size is N:

	N=10		**N=100**		**N=1000**	
	Steps	**Time**	**Steps**	**Time**	**Steps**	**Time**
f(N) = N	10	0.1ms	100	1ms	1000	10ms
f(N) = N^2	100	1.0ms	10^4	100ms	10^6	10s
f(N) = N^3	1000	10.0	10^6	10s	10^9	2hrs 47min
f(N) = 2^N	1024	10.2	1.27×10^{30}	4.0×10^{17} yrs	1.07×10^{301}	3.4×10^{288} yrs

Comparison: $O(n^2)$ vs $O(n \log n)$

Assume a machine is running at 50 000 MIPs
Assume 1 instruction per iteration.
Time to sort 1 billion numbers:

Bubble sort $O(n^2)$

$(10^9)^2$ steps / (5×10^{10} steps/sec) $\approx$ 8 months.

Quick sort O(n log n)

$10^9 * \log_2(10^9)$ steps / (5×10^{10} steps/sec)
$= 10^9 * 30$ steps / (5×10^{10} steps/sec) $\approx$ 0.6 seconds.