CSC 212: Data Structures and Abstractions

03: Introduction to Analysis of Algorithms (part 1)

Prof. Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Spring 2025

THINK BIG WE DO℠

Jan 7, 2025 - Technology

OpenAI's new o3 model freaks out computer science majors

Angrej Singh

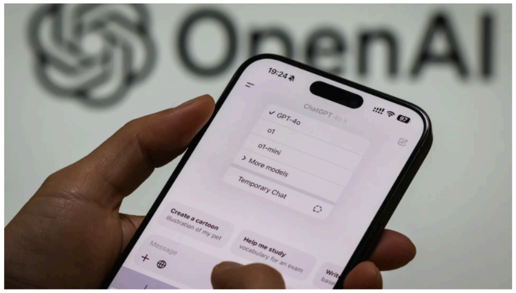


Photo illustration: Cheng Xin/Getty Images

OpenAI's announcement of its new o3 reasoning model has triggered another wave of anxiety among some computer science majors who fear AI will edge them out of the job market.

For people who want to go into computer science, "I would tell them there are so many new things that need to be built and would not worry at all," Pascal Van Hentenryck, director of Georgia Tech's AI Hub.

"The job opportunities are going to increase," he said. The easy and tedious tasks will become automated and people will be able to **work at a higher, sophisticated level**."

From lab session

Dynamic memory allocation using new/delete

```
int *ptr1 = new int[100];
int *ptr2 = ptr1;  // both pointers reference the same address

delete [] ptr1; // array is freed
delete [] ptr2; // ERROR: attempting to delete already freed memory
```

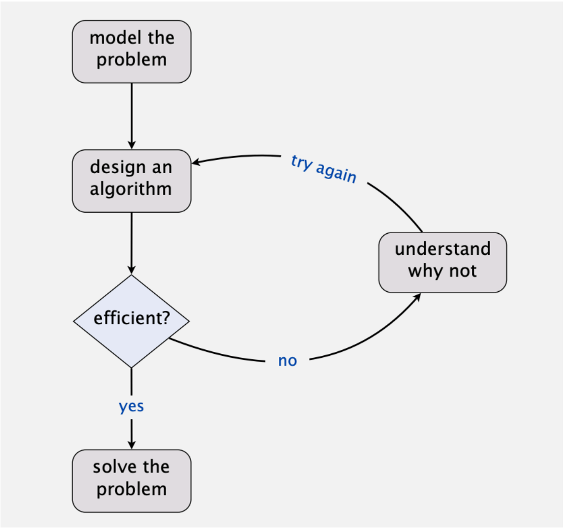
Review sessions every Friday 2-4p at Tyler 55

Analysis of algorithms

Problems, algorithms, programs

- Problem
  - a computational problem represents a formalized task requiring a solution
  - well-defined input specifications, expected output requirements, and formal constraints and conditions
- Algorithm
  - precise, unambiguous sequence of computational steps
  - essential properties:
    - correctness (produces valid output for all valid inputs), finiteness (terminates after a finite number of steps), determinism (produces consistent results), feasibility (each step must be executable)
- Program implementation
  - concrete realization of an algorithm using a programming language

Developing a usable algorithm



[COS 226 lectures, Princeton University]

Importance of analysis of algorithms

- Scientific classification of algorithmic solutions
- Performance prediction and resource utilization
  - time complexity (running time) and space complexity (memory)
- Establishment of theoretical guarantees
- Understanding problem complexity
- Optimization opportunities identification

Approaches

- Empirical analysis
  - implement the program using a programming language
  - systematic testing (execution) with varied input sizes
  - statistical analysis of collected data and hypothesis formation
  - validation through prediction models
- Theoretical analysis
  - mathematical modeling of time complexity and space complexity

Empirical analysis

Example: the e number

Mathematical constant that is the base of the natural logarithm. It is approximately equal to 2.71828

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n \qquad e = \frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\ldots$$

Algorithm 1

Solution 1

```
double euler1(int n) {
    double e = 1.0;
    for (int i = 1; i <= n; i++) {
        double fact = 1.0;
        for (int j = 1; j <= i; j++) {
            fact *= j;
        }
        e += (1.0 / fact);
    }
    return e;
}
```


Example: the e number

Mathematical constant that is the base of the natural logarithm. It is approximately equal to 2.71828.

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n \qquad e = \frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\ldots$$

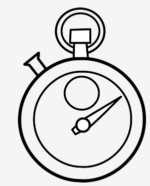
Algorithm 2

Solution 2

```
double euler2(int n) {
    double e = 1.0;
    double fact = 1.0;
    for (int i = 1; i <= n; i++) {
        fact *= i;
        e += (1.0 / fact);
    }
    return e;
}
```

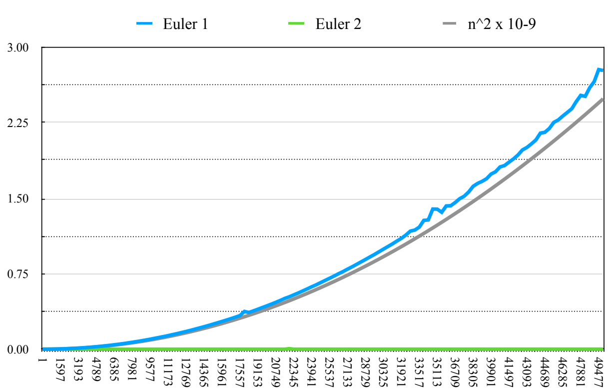

Timing

```
$ g++ -std=c++11 -O0 euler.cpp -o prog
```



```
#include <iostream>
#include <iomanip>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <n> <alg>\n";
        return 1;
    }

    double e;
    int n = std::stoi(argv[1]);
    int alg = std::stoi(argv[2]);

    auto start = std::chrono::high_resolution_clock::now();
    if (alg == 1) e = euler1(n);
    if (alg == 2) e = euler2(n);
    auto end = std::chrono::high_resolution_clock::now();

    std::chrono::duration<double> elapsed = end - start;
    std::cout << std::fixed << std::setprecision(10);
    std::cout << e << " " << (double) elapsed.count() << std::endl;

    return 0;
}
```


Hypothesis, Prediction, Validation



Hypothesis: formulate specific hypothesis about performance characteristics (usually time complexity), for example, we observe $T(n) = cn^2$.

Prediction: make concrete predictions that can be tested empirically, for example, predict running times for different values of n.

Validation: designing and implementing controlled experiments and comparing results against predictions.

```
$ seq 1 399 50000 | while read n; do echo -n "$n\t"; ./prog $n 1; done > e1.txt
$ seq 1 399 50000 | while read n; do echo -n "$n\t"; ./prog $n 2; done > e2.txt
```


Limitations of empirical analysis

- Implementation overhead
  - multiple algorithm implementations required
  - development time constraints
  - code quality variations
- Experimentation challenges
  - comprehensive test case design
  - time-intensive execution cycles
  - input distribution considerations
- Environmental dependencies
  - compiler optimizations (same code performs differently with different flags -O1, -O2, -O3)
  - hardware architecture variations (AVX, branch prediction, cache sizes)
  - operating system scheduling and runtime environment fluctuations