Algorithm Analysis?

EMBRY-RIDDLE Aeronautical University

- What is algorithm analysis?
  - Algorithm analysis is a means of comparing algorithms to one another
  - Given any problem, many sequences of steps may result in identical results
  - How do you select which is the "best"?

CS 315 College of Engineering, Daytona Beach, FL

1

Algorithm Analysis?

EMBRY-RIDDLE Aeronautical University

- What is algorithm analysis?
  - Algorithm analysis is a means of comparing algorithms to one another
  - Given any problem, many sequences of steps may result in identical results
  - How do you select which is the "best"?

Analysis Criteria include:

- Correctness
- Simplicity
- Time consumed
- Space used

CS 315 College of Engineering, Daytona Beach, FL

2

Algorithm Analysis?

EMBRY-RIDDLE Aeronautical University

- What is algorithm analysis?
  - Algorithm analysis is a means of comparing algorithms to one another
  - Given any problem, many sequences of steps may result in identical results
  - How do you select which is the "best"?

Analysis Criteria/Metrics include:
- Correctness
- Simplicity
- Time consumed
- Space used

CS 315 College of Engineering, Daytona Beach, FL

3

Scientific Method and Analysis

EMBRY-RIDDLE Aeronautical University

- Often difficult to formulate a precise measure of an algorithm's performance
  - Varies based upon system inputs
  - Algorithms can be too complex to determine exact time or space performance
- Scientific method can be employed:
  - **Observe** – observe the system
  - **Hypothesize** – develop a model to approximate the system
  - **Predict** – Predict output given inputs
  - **Verify** – evaluate performance versus predictions
  - **Validate** – re-run until hypothesis and observations agree
- Hypothesis must be **falsifiable**
- Experimental design should permit repeated steps

CS 315 College of Engineering, Daytona Beach, FL

4

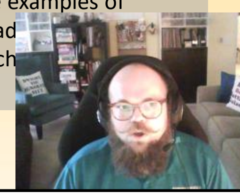
Scientific Method and Analysis

EMBRY-RIDDLE Aeronautical University

- Often difficult to formulate a precise measure of an algorithm's performance
  - Varies based upon system inputs
  - Algorithms can be too complex to determine exact time or space performance
- Scientific method can be employed:
  - **Observe** – observe the system
  - **Hypothesize** – develop a model to approximate the system
  - **Predict** – Predict output given inputs
  - **Verify** – evaluate performance versus predictions
  - **Validate** – re-run until hypothesis and observations agree
- Hypothesis must be **falsifiable**
- Experimental design should permit repeated steps

In this chapter, the authors are demonstrating this process with the ***ThreeSum*** example:

- Observations are made
- Models and experiments are used to validate model
- Propositions are made as the authors provide examples of propositions mad
analysis and each
the text.



CS 315 College of Engineering, Daytona Beach, FL

5

Observation

EMBRY-RIDDLE Aeronautical University

- Observation is not enough to, but observing run-time can help us develop our model
- e.g. Stopwatch / Wall clock time
  - Measurement of the runtime is one approach to observe an algorithms timing performance
  - What are some of the reasons that wall clock time can be problematic?
    - Cannot use results to produce generalized predictions of runtime
      - Conditions such as CPU, system load, compilation methods, available RAM, etc. can influence the runtime of software
    - More rigorous experimentation with statistical analysis needed to

CS 315 College of Engineering, Daytona Beach, FL

6

Analysis of Experimental Data

EMBRY-RIDDLE
Aeronautical University

- Given our metric, e.g. run-time, our experiment would need to vary the size of the problem being solved
  - e.g. size of the data set being processed, or some other run-time parameter
- Plotting timing results
  - Standard Plot
  - Logarithmic Plot
- Textbook provides an example of this technique called the *D*
Rule.

CS 315 College of Engineering, Daytona Beach, FL

7

Modeling Complexity

EMBRY-RIDDLE
Aeronautical University

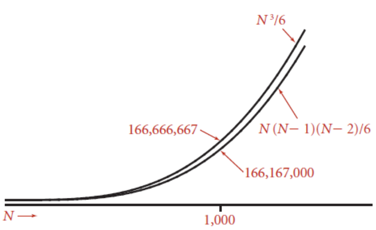
- Donald Knuth observed:
  - Cost of executing each statement
    - How long does the op take?
  - Frequency of execution of each statement
    - How often is a particular operation made?
- Moving away from real-time we focus on approximation:
  - Assume that each read/write/access operation costs roughly one (1) time unit.
  - Each problem is of some size N, which is defined at runtime
    - E.g. an array's length would be of size N for an array processing problem.
- We seek to create as accurate model of the runtime prediction f(r
is the number of operations performed as a function of problem s

CS 315 College of Engineering, Daytona Beach, FL

8

Approximation of our model

EMBRY-RIDDLE Aeronautical University



Leading-term approximation

function	tilde approximation	order of growth
$N^3/6 - N^2/2 + N/3$	$\sim N^3/6$	N^3
$N^2/2 - N/2$	$\sim N^2/2$	N^2
$\lg N + 1$	$\sim \lg N$	$\lg N$
3	~ 3	1

Typical tilde approximations

Tilde Approximation:

from textbook:

Definition. We write $\sim f(N)$ to represent any function that, when divided by $f(N)$, approaches 1 as N grows, and we write $g(N) \sim f(N)$ to indicate that $g(N)/f(N)$ approaches 1 as N grows.

order of growth

description	function
constant	1
logarithmic	$\log N$
linear	N
linearithmic	$N \log N$
quadratic	N^2
cubic	N^3
exponential	2^N

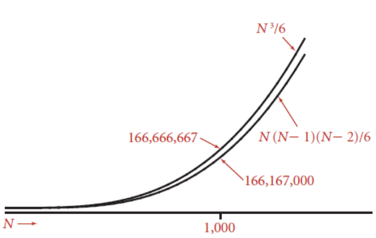
Com
orde

CS 315 College of Engineering, Daytona Beach, FL

9

Approximation of our model

EMBRY-RIDDLE Aeronautical University



Leading-term approximation

function	tilde approximation	order of growth
$N^3/6 - N^2/2 + N/3$	$\sim N^3/6$	N^3
$N^2/2 - N/2$	$\sim N^2/2$	N^2
$\lg N + 1$	$\sim \lg N$	$\lg N$
3	~ 3	1

Typical tilde approximations

Tilde Approximation:

from textbook:

Definition. We write $\sim f(N)$ to represent any function that, when divided by $f(N)$, approaches 1 as N grows, and we write $g(N) \sim f(N)$ to indicate that $g(N)/f(N)$ approaches 1 as N grows.

Alternatively, we can represent our mathematical models with arithmetic approximation methods such as:

- Big-O
- Big-Omega
- Big-Theta

This approximations are similar, but different to tilde, and will be discussed later in this module.

order of growth

description	function
constant	1
logarithmic	$\log N$
linear	N
linearithmic	$N \log N$
quadratic	N^2
cubic	N^3
exponential	2^N

Com
orde

CS 315 College of Engineering, Daytona Beach, FL

10

Example

EMBRY-RIDDLE Aeronautical University

```
public class ThreeSum
{
   public static int count(int[] a)
   {
      int N = a.length;
      int cnt = 0;
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
               if (a[i] + a[j] + a[k] == 0)
                  cnt++;
      return cnt;
   }

   public static void main(String[] args)
   {
      int[] a = In.readInts(args[0]);
      StdOut.println(count(a));
   }
}
```

blocks of statements: A, B, C, D, E

frequencies of execution: 1, N, $\sim N^2/2$, $\sim N^3/6$, x

inner loop

Anatomy of a program's statement execution frequencies

statement block	time in seconds	frequency	total time
E	t_0	x (*depends on input*)	$t_0 x$
D	t_1	$N^3/6 - N^2/2 + N/3$	$t_1(N^3/6 - N^2/2 + N/3)$
C	t_2	$N^2/2 - N/2$	$t_2(N^2/2 - N/2)$
B	t_3	N	$t_3 N$
A	t_4	1	t_4

grand total
$$(t_1/6)N^3 + (t_2/2 - t_1/2)N^2 + (t_1/3 - t_2/2 + t_3)N + t_4 + t_0 x$$

tilde approximation $\sim (t_1/6)N^3$ (*assuming x is small*)

order of growth N^3

Analyzing the running time of a program

CS 315 College of Engineering, Daytona Beach, FL

11

Example

EMBRY-RIDDLE Aeronautical University

```
public class ThreeSum
{
   public static int count(int[] a)
   {
      int N = a.length;
      int cnt = 0;
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
               if (a[i] + a[j] + a[k] == 0)
                  cnt++;
      return cnt;
   }

   public static void main(String[] args)
   {
      int[] a = In.readInts(args[0]);
      StdOut.println(count(a));
   }
}
```

blocks of statements: A, B, C, D, E

frequencies of execution: 1, N, $\sim N^2/2$, $\sim N^3/6$, x

inner loop

Anatomy of a program's statement execution frequencies

statement block	time in seconds	frequency	total time
E	t_0	x (*depends on input*)	$t_0 x$
D	t_1	$N^3/6 - N^2/2 + N/3$	$t_1(N^3/6 - N^2/2 + N/3)$
C	t_2	$N^2/2 - N/2$	$t_2(N^2/2 - N/2)$
B	t_3	N	$t_3 N$
A	t_4	1	t_4

grand total
$$(t_1/6)N^3 + (t_2/2 - t_1/2)N^2 + (t_1/3 - t_2/2 + t_3)N + t_4 + t_0 x$$

tilde approximation $\sim (t_1/6)N^3$ (*assuming x is small*)

order of growth N^3

Analyzing the running time of a program (example)

We will need to evaluate our analysis experimentally to confirm, but this seems like a reasonable estimate.

CS 315 College of Engineering, Daytona Beach, FL 12

12