SCC 120
Fundamentals of Computer Science

Introduction to Algorithms

Overview (this unit)

- What is an algorithm, Examples of algorithms, Why study them
- Running time, Input size
- Cost of operations
- Operation counting and T(N): Linear search

What is an Algorithm?

- “Algorithm”: named after Persian mathematician al-Khwarizmi

- A detailed step-by-step sequence of computations used to accomplish some task:
  - Start from initial **input**
  - **Finite list of well-defined instructions** which proceed through finite number of successive states
  - Terminates at final ending state and produces **output**

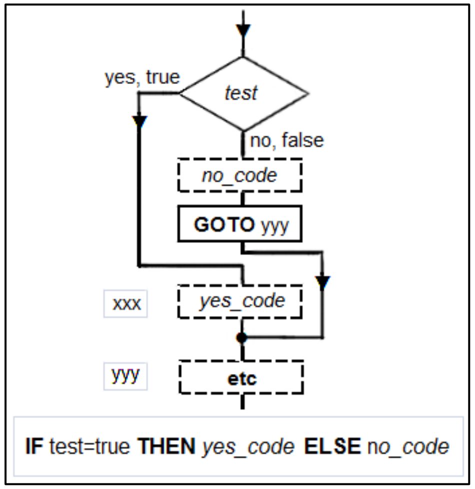
Algorithm Representations

```
int Multiply(int input1, int input2)
{
      int theProduct = 0;
      for(int i = 0; i < input2; i = i + 1)
            theProduct = theProduct + input1;
      return theProduct;
}
```

Computer code

```
For(each node $n_i$ in $N$)
  For (each node $n_j$ in $N$, $n_j \neq n_i$)
    $d_{ij}$= Euclidean distance ($n_i, n_j$)
    apply expansion force $X_{ij} = -G/d_{ij}$
    If(edge ($n_i, n_j$) in $E$)
      apply string force $S_{ij} = s(d_{ij} - d_o)$
```

Psuedocode

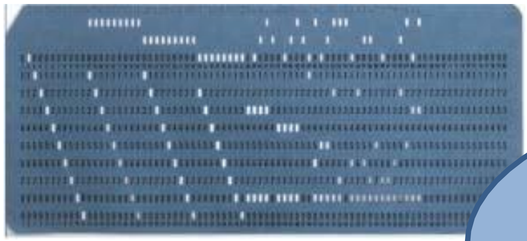


Flow charts

```
Goal: Write a program that guesses an
  integer number that the user is thinking of.

  It should repeatedly ask the user if
  some number 'guess' is their hidden
  number and continue until they respond: "yes".
```

Text/sentences



Punch cards

More important:
- What an algorithm does
- Runtime analysis

Why study Algorithms?

Algorithms require resources during computation to solve problem

> Time: how many steps (relative to input size) does it take to solve a problem
>
> Space: how much memory does it take to solve a problem

Other than performance: correctness, robustness, and simplicity are important

Study of Algorithms

Complexity Theory: deals with resources required during computation to solve a given problem (we focus on counting operations and different classes of problems)

Overview (this unit)

- What is an algorithm, Examples of algorithms, Why study them
- Running time, Input size
- Cost of operations
- Operation counting and T(N): Linear search

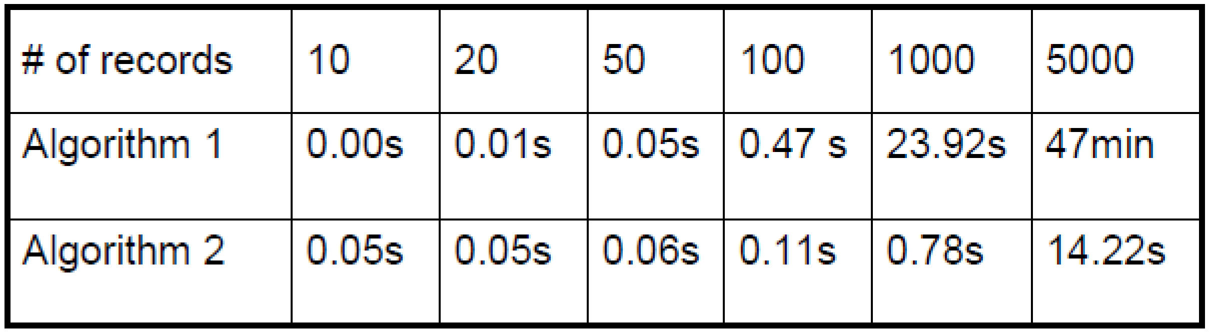
Running Time

- **size of the input**
- **organization of the input**
- **Optimal operating temperature**
- **Number of processor cores**

- Generally, we seek **upper bounds on the running time,**

Input Size

- Runtimes in seconds of two fictional algorithms for processing employees' records



# of records	10	20	50	100	1000	5000
Algorithm 1	0.00s	0.01s	0.05s	0.47 s	23.92s	47min
Algorithm 2	0.05s	0.05s	0.06s	0.11s	0.78s	14.22s

- Which algorithm is better?
- Algorithm 2 is faster when processing more than 100, and even faster for more than 1000 records

Input Size

- Implement algorithm and measure with a clock actual runtimes for different input sizes (e.g. 10, 20, 50, 100, 1000, 5000 employees’ records)
- Problem: What is the runtime for other input sizes (e.g. 5, 25, 1000000 employees’ records)?
- What is the runtime for any input size?
- Actual runtime can be estimated
- We will analyse algorithms, and come up with classes of runtimes based on input size

Cost of Operations

When we do operation counting, these all take same amount of time (1 operation)

Arithmetic operations:

+, -, ==, <, >	fast
*	slower
/	slower still
x^y, sin(x)	even slower. Why?

Answer: Assume Random Access Machine (RAM) Model

Operations and Instructions

Operations in a high-level language may require many machine-level instructions.

Question: What instructions are required by these operations?

x = a + b;

(1) fetch value of a; (2) fetch value of b; (3) add; (4) assign result to x

x = theArray[i];

(1) fetch base address (BA) of 'theArray'; (2) fetch value of i; (3) check that i is within bounds of 'theArray'; (4) add i and BA; (5) use this address to fetch value of theArray[i]; (6) assign this value to x

x = sqrt(y);

(1) fetch the value of y; (2) invoke square root function with this value as input; (3) assign result to x

The number of instructions are different, but (usually) we still count 1 operation

Cost of Operations

The basic operations are faster on integer data than they are on floating-point data

But worrying about integer vs. float is too detailed here, so we still count 1 operation!

Data types:

– Primitive types: *int*, *char*, *boolean* are fast

– Compound types: *arrays*, *classes* are slower

Overview (this unit)

- What is an algorithm, Examples of algorithms, Why study them
- Running time, Input size
- Cost of operations
- Operation counting and T(N): Linear search

Linear Search

Example of array of integers, and one integer to search for in array

isInArray returns true if the number is in the array

```
boolean isInArray(int theArray[], int iSearch)
{
      int N = theArray.length;

      for (int i = 0; i < N; i++)
            if (theArray[i] == iSearch)
                  return true;

      return false;
}
```

Which operations are executed every time around the loop?

Operation Counting

```
boolean isInArray(int theArray[], int iSearch)
{
      int N = theArray.length;
                        o1    o3
      for (int i = 0; i < N; i++)
                          o2
            if (theArray[i] == iSearch)
                  return true;

      return false;
}
```

Operations o1, o2, o3 executed every time around *for* loop

Operation Counting

How many times each operation is executed?

	o1	o2	o3
iSearch is the first element	1	1	0
iSearch is the second element	2	2	1
…		…	
iSearch is the last element	N	N	N-1
iSearch is not in theArray	N+1	N	N

Which are the best and worst cases?

– Best case: 1 loop; Worst case: N loops

Which will be the average case for randomly distributed values?

- (1 + 2 +…+ N)/N = (N + 1)/ 2 loops [Arithmetic series]

Working out T(N)

Time function T(N):

– The time a program takes to act on an input of size N

Let the time program takes in a single loop be C_1 (constant)

(this is for operations o1, o2, o3 together)

What will be the average looping time?

$C_1 \times (N + 1) \times 0.5$

Working out T(N)

Time function T(N):

– The time a program takes to act on an input of size N

Function has one-off initialization cost C_2 (e.g. int N = theArray.length)

What is the overall program time (average case)?

$$T(N) = (C_1 \times 0.5 \times (N + 1)) + C_2$$
$$T(N) = (C_1 \times 0.5 \times N) + (C_1 \times 0.5) + C_2$$
$$T(N) = (C_1 \times 0.5) \times N + (C_1 \times 0.5) + C_2$$

So $T(N) = A \times N + B$, with $A = C_1 \times 0.5$ and $B = (C_1 \times 0.5) + C_2$

T(N) is Linear

$T(N) = A \times N + B$

If we plot this, we get a line with slope A

The time taken by this program is directly proportional to the size of the input:

–doubling N (approximately) doubles the time taken

– tripling N (approximately) triples the time taken

Overview

- More examples of operation counting and T(N)
  - Case 1: Constant
  - Case 2: Linear (average of N integers)
  - Case 3: Linear (find minimum of N integers)
  - Case 4: Logarithmic
  - Case 5: Quadratic, Cubic
  - Case 6: Exponential

Overview

- More examples of operation counting and T(N)
  - Case 1: Constant
  - Case 2: Linear (average of N integers)
  - Case 3: Linear (find minimum of N integers)
  - Case 4: Logarithmic
  - Case 5: Quadratic, Cubic
  - Case 6: Exponential

Case 1: Averaging Function

Write a function avg5 which takes one argument (an array of 5 integers), and returns the average of the integers

```
double avg5(int theArray[])
{
      int total = 0;
      for (int i=0; i<5; i++)
            total += theArray[i];
      double avg = ((double) total) / 5.0;
      return avg;
}
```

Case 1: Operation Counting

```
double avg5(int theArray[])
{
      int total = 0;
                      o1   o3
      for (int i=0; i<5; i++)
                   o2
            total += theArray[i];
      double avg = ((double) total) / 5.0;
      return avg;
}
```

Operations o1, o2, o3 executed every time around *for* loop

Case 1: Operation Counting

How many times each operation is executed?

	o1	o2	o3
all cases	6	5	5

Which are the best and worst cases?

– Best case: 5 loops; Worst case: 5 loops

Which will be the average case?

– “Average” case: 5 loops

Case 1: Working out T(N)

Time function T(N):

– The time a program takes to act on an input of size N

Let the time program takes in a single loop be C_1 (constant)

(this is for operations o1, o2, o3 together)

What will be the **worst case** looping time?

C_1 x 5

Case 1: Working out T(N)

Time function T(N):

– The time a program takes to act on an input of size N

Function has one-off initialisation cost C_2 (e.g. int total = 0)

What is the overall program time?

$$T(N) = C_1 \times 5 + C_2$$

So T(N) = Constant

Case 1: T(N) is Constant

T(N) = Constant

The time taken by this program is independent of the size of the input:

– N is the size of array which can be more than 5, but since we only care about the first 5 values, increasing N does not change the time taken

Overview

- More examples of operation counting and T(N)
  - Case 1: Constant
  - Case 2: Linear (average of N integers)
  - Case 3: Linear (find minimum of N integers)
  - Case 4: Logarithmic
  - Case 5: Quadratic, Cubic
  - Case 6: Exponential

Case 2: Averaging N Function

```
double avgN(int theArray[])
{
      int N = theArray.length;
      int total = 0;
      for (int i=0; i<N; i++)
            total += theArray[i];
      double avg = ((double) total) / ((double) N);
      return avg;
}
```

Which operations are executed every time around the loop?

Case 2: Operation Counting

```
double avgN(int theArray[])
{
      int N = theArray.length;
      int total = 0;
                         o1   o3
      for (int i=0; i<N; i++)
                      o2
            total += theArray[i];
      double avg = ((double) total) / ((double) N);
      return avg;
}
```

Operations o1, o2, o3 executed every time around *for* loop

Case 2: Operation Counting

How many times each operation is executed?

	o1	o2	o3
all cases	N+1	N	N

Which are the best and worst cases?

– Best case: N loops; Worst case: N loops

Which will be the average case?

– “Average” case: N loops

Case 2: Working out T(N)

Let the time program takes in a single loop be C_1 (constant) (this is for operations o1, o2, o3 together)

What will be the worst case looping time?

$C_1 \times N$

Function has one-off initialisation cost C_2 (e.g. int total = 0)

What is the overall program time?

$T(N) = C_1 \times N + C_2$

Case 2: T(N) is Linear

$T(N) = C_1 \times N + C_2$

Try plot this in graph

The time taken by this program is directly proportional to the size of the input

Case 3: Simple Example

Example of array of integers, and find minimum integer in array

Case 3: Function

Exercise (5 min)

Write a function findMin which takes one argument (an array of integers), and returns the index of the array location that contains the smallest integer in array

```
int findMinIndex(int theArray[])
{


}
```

Case 3: Function

```
int findMin(int theArray[])
{
       int N = theArray.length; // assume at least 1
       int smallest_i = 0; //Assume smallest value at index 0
       for (int i=1; i<N; i++)
              if (theArray[i] < theArray[smallest_i])
                     smallest_i = i;
       return smallest_i;
}
```

Which operations are executed every time around the loop?

Case 3: Operation Counting

```
int findMin(int theArray[])
{
      int N = theArray.length;
      int smallest_i = 0;
                   o1    o3
      for (int i=1; i<N; i++)
                          o2
            if (theArray[i] < theArray[smallest_i])
                  smallest_i = i; // o4
      }
      return smallest_i;
}
```

Case 3: Operation Counting

How many times each operation is executed?

	o1	o2	o3	o4
all cases	N	N-1	N-1	0 to N-1

Which are the best and worst cases?

– Best case: N-1 loops; Worst case: N-1 loops

Which will be the average case?

– Average case: N-1 loops

Case 3: Working out T(N)

Let the time program takes in a single loop be C_1 (constant)
(this is for operations o1, o2, o3, o4 together)

What will be the worst looping time?

$$C_1 \times (N-1)$$

Function has one-off initialisation cost C_2 (e.g. int N = theArray.length)

What is the overall program time?

$$T(N) = C_1 \times (N - 1) + C_2$$
$$T(N) = (C_1 \times N) - (C_1) + C_2$$
$$T(N) = C_1 \times N + (C_2 - C_1)$$

So $T(N) = A \times N + B$, with $A = C_1$ and $B = C_2 - C_1$

Case 3: T(N) is Linear

$T(N) = A \times N + B$

If we plot this, we get a line with slope A

The time taken by this program is directly proportional to the size of the input

Overview

- More examples of operation counting and T(N)
  - Case 1: Constant
  - Case 2: Linear (average of N integers)
  - Case 3: Linear (find minimum of N integers)
  - Case 4: Logarithmic
  - Case 5: Quadratic, Cubic
  - Case 6: Exponential

Case 4: Function

```
int logBaseTwoN(int N)
{
      int count = 0;
      while (N > 1) {
            count++;
            N = N/2;
      }
      return count;
}
```

Operation counting: Table of N vs. count

Case 4: Working out T(N)

Let the time program takes in a single loop be C_1 (constant) (this is for conditional check of while loop and statements in while loop)

What will be the worst looping time?

$C_1 \times \log_2 N$

Function has one-off initialisation cost C_2 (e.g. int count=0)

What is the overall program time?

$T(N) = C_1 \times \log_2 N + C_2$

Case 4: T(N) is Logarithmic

$T(N) = C_1 \text{ x } \log_2 N + C_2$

The time taken by this program is logarithmically proportional to the size of the input

[Our code/function logBaseTwoN() is accurate for N that is 2^x such that x is a natural number. It is for demonstration purposes only.]

Overview

- More examples of operation counting and T(N)
  - Case 1: Constant
  - Case 2: Linear (average of N integers)
  - Case 3: Linear (find minimum of N integers)
  - Case 4: Logarithmic
  - Case 5: Quadratic, Cubic
  - Case 6: Exponential

Case 5: Function

```
void quadratic(int N)
{
     int count = 0;
     for (int i=0; i<N; i++)
          for (int j=0; j<N; j++)
               count++;
}
```

Case 5: Working out T(N)

Let the time for a **single execution** of innermost loop be C_1
What will be the worst looping time over inner loop? C_1N

Let the time for a **single execution** of the outer loop excluding the time for execution of the inner loop (that is, i<N and i++ and j=0) be C_2
Therefore, a single execution of the outer loop takes in total $C_2 + C_1N$

So worst time over N executions of the outer loop?

$$(C_2 + C_1N)N$$

$$= N^2C_1 + NC_2$$

One-off initialisation cost C_3 (e.g. int count=0)
What is the overall time?

$$T(N) = N^2C_1 + NC_2 + C_3$$

Case 5: T(N) is Quadratic

$T(N) = N^2C_1 + NC_2 + C_3$

The time taken is quadratic with respect to input size N (because the highest order term is N^2)

Case 5: Function

Exercise (1 min)

Following the previous quadratic example, write code that takes cubic time:

Case 5: Function

Exercise (1 min)

Following the previous quadratic example, write code that takes cubic time:

```
void cubic(int N)
{
    int count = 0;
    for (int i=0; i<N; i++)
      for (int j=0; j<N; j++)
            for (int k=0; k<N; k++)
                  count++;
}
```

Case 5: Working out T(N)

By doing an analogous analysis as for quadratic, we get worst case time as

$$T(N) = N^3C_1 + N^2C_2 + NC_3 + C_4$$

Case 5: T(N) is Cubic

$T(N) = N^3C_1 + N^2C_2 + NC_3 + C_4$

The time taken is cubic with respect to input size N

(because the highest order term is N^3)

Overview

- More examples of operation counting and T(N)
  - Case 1: Constant
  - Case 2: Linear (average of N integers)
  - Case 3: Linear (find minimum of N integers)
  - Case 4: Logarithmic
  - Case 5: Quadratic, Cubic
  - Case 6: Exponential

Case 6: Function

```
print_list(anArray of size N)
{
      create newArray of size 2^N
      for (i=1 to 2^N) //inclusive of 2^N
            newArray[i] = “Exponential!”

      }
```

Case 6: Working out T(N)

Let the time program takes in a single loop be C_1 (constant)

What will be the worst looping time?

$C_1 \times 2^N$

Function has one-off initialisation cost C_2

What is the overall program time?

$T(N) = C_1 \times 2^N + C_2$

Case 6: T(N) is Exponential

$T(N) = C_1 \times 2^N + C_2$

The time taken by this program is exponential with respect to the input size. The time gets very large even for small values of N.

Summary

- More examples of operation counting and T(N)
  - Case 1: Constant
  - Case 2: Linear (average of N integers)
  - Case 3: Linear (find minimum of N integers)
  - Case 4: Logarithmic
  - Case 5: Quadratic, Cubic
  - Case 6: Exponential