Analysis of Algorithms

Introduction

The Course

- Purpose: a rigorous introduction to the design and analysis of algorithms
  - Not a lab or programming course
  - Not a math course, either
- Textbook: *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein
  - The “Big White Book”
  - Second edition: now “Smaller Green Book”
  - An excellent reference you should own

The Course

- Grading policy:
  - Homework: 5%
  - Quizzes: 20%
  - Mid Term: 25%
  - Final: 50%

Algorithm

- What is the best Algorithm for a given Problem?
- Three Things you will learn
  - Design a good Algorithm
  - Analyze it.
  - Know when to stop (lower bounds)
- The word “algorithm” derived from Mohammed Al-Khowarizmi, 9th century Persian mathematician.

Definition

- An Algorithm is any well defined computational procedure that some value, or set of values, as input and produces some value, or set of values, as output.
- A correct algorithm halts with the correct output for every instance. We can then say the algorithm solves the problem.

Famous Algorithms

- Construction of Euclid
- Newton’s root finding
- Fast Fourier Transform
- Compression (Huffman, Lempel-Ziv, GIF, MPEG)
- DES, RSA encryption
- Simplex algorithm for linear programming
- Shortest Path Algorithm (Dijkstra, Bellman Ford)
- Dynamic Programming
- Error correcting codes (CDs, DVDs)

Famous Algorithms

- TCP congestion control, IP routing
- Pattern matching (Genomics)
- Delaunay Triangulation (FEM. Simulation)

Course Outline

Review: Induction

- Suppose
  - S(k) is true for fixed constant k
    - Often k = 0
  - S(n) à S(n+1) for all n >= k
- Then S(n) is true for all n >= k

Proof By Induction

- Claim:S(n) is true for all n >= k
- Basis:
  - Show formula is true when n = k
- Inductive hypothesis:
  - Assume formula is true for an arbitrary n
- Step:
  - Show that formula is then true for n+1

Induction Example: Gaussian Closed Form

- Prove 1 + 2 + 3 + … + n = n(n+1) / 2
  - Basis:
    - If n = 0, then 0 = 0(0+1) / 2
  - Inductive hypothesis:
    - Assume 1 + 2 + 3 + … + n = n(n+1) / 2
  - Step (show true for n+1):

    1 + 2 + … + n + n+1 = (1 + 2 + …+ n) + (n+1)

    = n(n+1)/2 + n+1 = [n(n+1) + 2(n+1)]/2

    = (n+1)(n+2)/2 = (n+1)(n+1 + 1) / 2

Induction Example: Geometric Closed Form

- Prove $a^0 + a^1 + \dots + a^n = (a^{n+1} - 1)/(a - 1)$ for all $a \neq 1$
  - Basis: show that $a^0 = (a^{0+1} - 1)/(a - 1)$

    $a^0 = 1 = (a^1 - 1)/(a - 1)$
  - Inductive hypothesis:
    - Assume $a^0 + a^1 + \dots + a^n = (a^{n+1} - 1)/(a - 1)$
  - Step (show true for n+1):

    $a^0 + a^1 + \dots + a^{n+1} = a^0 + a^1 + \dots + a^n + a^{n+1}$

    $= (a^{n+1} - 1)/(a - 1) + a^{n+1} = (a^{n+1+1} - 1)/(a - 1)$

Analyzing Algorithm

How does the algorithm behave as the problem size gets very large?

- Running time
- Memory/storage requirements
- Bandwidth/power requirements/logic gates/etc.

Input Size

- Time and space complexity
  - This is generally a function of the input size
    - E.g., sorting, multiplication
  - How we characterize input size depends:
    - Sorting: number of input items
    - Multiplication: total number of bits
    - Graph algorithms: number of nodes & edges
    - Etc

Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function call most statements roughly require the same amount of time
    - y = m * x + b
    - c = 5 / 9 * (t - 32)
    - z = f(x) + g(y)
- We can be more exact if need be

Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee
- Average case
  - Provides the expected running time
  - Very useful, but treat with care: what is “average”?
    - Random (equally likely) inputs
    - Real-life inputs

An Example: Insertion Sort

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

30	10	40	20
1	2	3	4

i = ∅   j = ∅   key = ∅
A[j] = ∅   A[j+1] = ∅

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

30	10	40	20
1	2	3	4

i = 2   j = 1   key = 10
A[j] = 30   A[j+1] = 10

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

30	30	40	20
1	2	3	4

i = 2   j = 1   key = 10
A[j] = 30   A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

30	30	40	20
1	2	3	4

i = 2   j = 1   key = 10
A[j] = 30   A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

30	30	40	20
1	2	3	4

i = 2   j = 0   key = 10
A[j] = ∅   A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

30	30	40	20
1	2	3	4

i = 2   j = 0   key = 10
A[j] = ∅   A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 2   j = 0   key = 10
A[j] = ∅   A[j+1] = 10

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```

An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 3 j = 0 key = 10
A[j] = ∅ A[j+1] = 10

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 3 j = 0 key = 40
A[j] = ∅ A[j+1] = 10

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 3 j = 0 key = 40
A[j] = ∅ A[j+1] = 10

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 3 j = 2 key = 40
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 3 j = 2 key = 40
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 3 j = 2 key = 40
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 4 j = 2 key = 40
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 4 j = 3 key = 20
A[j] = 40 A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 4 j = 3 key = 20
A[j] = 40 A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 3 key = 20
A[j] = 40 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 3 key = 20
A[j] = 40 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 3 key = 20
A[j] = 40 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
     }
     A[j+1] = key
  }
}
```

An Example: Insertion Sort

10	30	30	40
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	30	40
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	30	40
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	30	30	40
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	20	30	40
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```


An Example: Insertion Sort

10	20	30	40
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Done!

Animating Insertion Sort

- Check out the Animator, a java applet at:

  http://www.cs.hope.edu/~alganim/animator/Animator.html

- Try it out with random, ascending, and descending inputs

Insertion Sort

What is the precondition for this loop?

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Insertion Sort

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```

How many times will this loop execute?

Insertion Sort

Statement	Effort
`InsertionSort(A, n) {`	
`for i = 2 to n {`	$c_1 n$
`key = A[i]`	$c_2(n-1)$
`j = i - 1;`	$c_3(n-1)$
`while (j > 0) and (A[j] > key) {`	$c_4 \Sigma t_j$
`A[j+1] = A[j]`	$c_5\, \Sigma(t_{j-1})$
`j = j - 1`	$c_6\, \Sigma(t_{j-1})$
`}`	
`A[j+1] = key`	$c_7(n-1)$
`}`	
`}`	

Analyzing Insertion Sort

- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \Sigma_{2 \leq j \leq n} t_j + c_5 \Sigma_{2 \leq j \leq n} (t_j-1) + c_6 \Sigma_{2 \leq j \leq n} (t_j-1) + c_7(n-1)$
- What can T be?
  - Best case -- inner loop body never executed
    - $t_i = 1$ è $T(n)$ is a linear function
  - Worst case -- inner loop body executed for all previous elements
    - $t_i = i$ è $T(n)$ is a quadratic function
  - Average case
    - ???