CSC 325 Algorithms & Advanced Data Structures

https://courses.missouristate.edu/anthonyclark/325/
Alternative Course Titles

CSC 325 Becoming a Computer Scientist (and not just a programmer)

CSC 325 Preparing for Whiteboard Interviews

CSC 325 How to Get a Job
Today

Supplemental Materials
- Chapter 2 of Introduction to Algorithms, Third Edition
- https://www.toptal.com/developers/sorting-algorithms/

Learning Objectives
- Course description
- Algorithms warm-up
Instructor

Anthony Clark, PhD

- AnthonyClark@MissouriState.edu
- Office: Cheek Hall 307
- Office hours:
  - MWF from 10:00 AM to 11:00 AM
  - TU from 12:00 PM to 1:00 PM
- or by appointment
Course Materials

Course Website (https://courses.missouristate.edu/anthonyclark/325)
• Syllabus
• Lecture notes
• Assignment descriptions
• Exam prep

Slack (https://missouri-state-csc325.slack.com/)
• Ask questions, discussions, etc.

Book
• *Introduction to Algorithms*, Third Edition (Cormen, Leiserson, Rivest, Stein)
Topics (many relate to interview questions)

Algorithm analysis
- Asymptotic notations: $w, W, q, O, o$
- Growth of functions
- Solving recurrences
- Amortized analysis

Advanced data structures
- Hash tables
- Heaps
- Red-black trees
- Disjoint-sets
- Graphs

Algorithm design approaches
- Incremental approach
- Divide-and-conquer
- Randomized algorithms
- Dynamic programming
- Greedy algorithms
- Approximation algorithms
- Others: Brute-force algorithms, etc.

NP-Completeness
- Strategies for hard problems
**Grading**

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
<th>Grade</th>
<th>Minimum Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quizzes</td>
<td>10%</td>
<td>A (4.00)</td>
<td>&gt;=90%</td>
</tr>
<tr>
<td>Assignments</td>
<td>50%</td>
<td>A- (3.70)</td>
<td>86%</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>20%</td>
<td>B+ (3.30)</td>
<td>83%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>20%</td>
<td>B (3.00)</td>
<td>80%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B- (2.70)</td>
<td>76%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C+ (2.30)</td>
<td>73%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C (2.00)</td>
<td>70%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C- (1.70)</td>
<td>66%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D+ (1.30)</td>
<td>63%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D (1.00)</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F (0.00)</td>
<td>&lt;60%</td>
</tr>
</tbody>
</table>
Grading

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quizzes</strong></td>
<td>10%</td>
</tr>
<tr>
<td>Assignments</td>
<td>50%</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>20%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>20%</td>
</tr>
</tbody>
</table>

- Unannounced
- Worth 10 points each (max)
- Receive 3 points for turning something in
- Receive 7 points for “correct” answers
- Receive 5 points for giving feedback on the course
- **Always work in groups**
# Grading

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quizzes</td>
<td>10%</td>
</tr>
<tr>
<td>Assignments</td>
<td>50%</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>20%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>20%</td>
</tr>
</tbody>
</table>

- Approximately 10 assignments
- Turned in using [autogradr.com](http://autogradr.com) or [gradescope.com](http://gradescope.com)
  - Automated grading for functionality
  - Can submit multiple times for grading
- Usually completed in pairs
  - Will be compared against each other using MOSS or equivalent
Grading

Quizzes : 10%
Assignments : 50%
Midterm Exam : 20%
Final Exam : 20%

- Final is not comprehensive
- Usually multiple-choice and short answer
- Can use a single 8.5” by 11” note sheet (both sides, no restrictions)
Class Policies (full text in syllabus)

[https://courses.missouristate.edu/anthonyclark/325/](https://courses.missouristate.edu/anthonyclark/325/)

- Attendance is not mandatory, but you will likely miss quizzes
- Academic Dishonesty (you are responsible for knowing the University’s honor code)
- Nondiscrimination
- Disability Accommodation
- Cell Phone
- Religious Accommodation
- Emergency Response
Storm Shelter and Evacuation

• Shelter Information (in case of severe weather).
  • Evacuate floors 1, 2, and 3 using Center, North and West stairs;
  • Shelter in basement interior hallway.

• Evacuation Instructions (in case the building needs to be evacuated for events such as fire, gas leak, etc.)
  • West to Siceluff Hall 1st Floor Classrooms and Lobby;
  • Overflow, Plaster Student Union Lower Level
Quiz
There are $N$ students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature, i.e., if $A$ is a friend of $B$ and $B$ is a friend of $C$, then $A$ is also a friend of $C$.

A friend circle is a group of students who are directly or indirectly friends.

You must write a function `friendCircles` that returns the number of friend circles in a class. Its argument, `friends`, is an $N \times N$ matrix that comprises characters $Y$ or $N$. If $\text{friends}[i][j]$ is $Y$ then the $i^{th}$ and $j^{th}$ students are friends, otherwise they are not friends.

Constraints:
- $1 \leq N \leq 300$.
- Each element of friends will be $Y$ or $N$.
- The number of rows and columns in `friends` will be equal.
- $\text{friends}[i][i]$ is $Y$, where $0 \leq i < N$.
- $\text{friends}[i][j] = \text{friends}[j][i]$, where $0 \leq i < j < N$.

Sample input 1:

```
YYNN
YYYN
NYYN
NNNY
```

Sample output 1:

```
2
```

Sample input 2:

```
YNNNN
YNvNN
NNYNN
NNYN
NNNNY
```

Sample output 2:

```
5
```
Warm-Up

Insertion Sort
• Input : a sequence of numbers
• Output : a reordering of the input into non-decreasing order

We want to
• See how to specify an algorithm
• Argue that it correctly sorts
• Analyze its running time
Insertion sort

1. // A is an array (base 1 index)
2. INSERTION-SORT (A)
3.     for j = 2 to A.length
4.         key = A[j]
5.     
6.         // Insert A[j] into sorted
6.         // sequence A[1..j-1]
7.         i = j - 1
8.         while i > 0 and A[i] > key
10.        i = i - 1
11.     A[i + 1] = key
Insertion sort – Proof of correctness

Lemma (loop invariant)

- At the start of each iteration of the for loop, the subarray A[1..j-1] consists of the elements originally in A[1..j-1], but in sorted order.

What is a lemma?
- an intermediate theorem in a proof

What is a theorem?
- a proposition that can be proved by a chain of reasoning

INSERTION-SORT (A)
1. for j = 2 to A.length
2. key = A[j]
3. i = j - 1
4. while i > 0 and A[i] > key
6. i = i - 1
7. A[i + 1] = key
Insertion sort – Proof of correctness

Lemma (loop invariant)
• At the start of each iteration of the for loop, the subarray A[1..j-1] consists of the elements originally in A[1..j-1], but in sorted order.

General conditions for loop invariants
1. Initialization: The loop invariant is satisfied at the beginning of the loop.
2. Maintenance: If the loop invariant is true before the ith iteration, then the loop invariant will be true before the i+1 iteration.
3. Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

INSERTION-SORT (A)
1. for j = 2 to A.length
2. key = A[j]
3. i = j - 1
4. while i > 0 and A[i] > key
6. i = i - 1
7. A[i + 1] = key
Insertion sort – Proof of correctness

1. **Initialization**: The loop invariant is satisfied at the beginning of the loop.

**Lemma (loop invariant)**

- At the start of each iteration of the for loop (lines 1-8), the subarray $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order.

- When $j = 2$, the subarray is $A[1]$, which is the first element of the array. The single element subarray is sorted.

**INSERTION-SORT (A)**

1. `for j = 2 to A.length`
2. `key = A[j]`
3. `i = j - 1`
4. `while i > 0 and A[i] > key`
6. `i = i - 1`
7. `A[i + 1] = key`
Insertion sort – Proof of correctness

2. **Maintenance**: If the loop invariant is true before the \(i\)th iteration, then the loop invariant will be true before the \(i+1\) iteration.

Lemma (loop invariant)

- At the start of each iteration of the for loop (lines 1-8), the subarray \(A[1..j-1]\) consists of the elements originally in \(A[1..j-1]\), but in sorted order.

- Suppose \(A[1..j-1]\) is sorted. Informally, the loop operates by moving \(A[j-1], A[j-2], \) etc. to the right until it finds the position of \(A[j]\). Next, \(j\) is incremented.

```
INSERTION-SORT (A)
1. for j = 2 to A.length
2. key = A[j]
3. i = j - 1
4. while i > 0 and A[i] > key
6. i = i - 1
7. A[i + 1] = key
```
Insertion sort – Proof of correctness

3. **Termination**: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

Lemma (loop invariant)

- At the start of each iteration of the for loop (lines 1-8), the subarray `A[1..j-1]` consists of the elements originally in `A[1..j-1]`, but in sorted order.

- The loop terminates when `j = n + 1`. Given the initialization and maintenance results, we have shown that: `A[1..n+1-1] → A[1..n]` in sorted order.

### INSERTION-SORT (A)

1. for `j = 2` to `A.length`
2. key = `A[j]`
3. `i = j - 1`
4. while `i > 0` and `A[i] > key`
6. `i = i - 1`
7. `A[i + 1] = key`
Insertion sort – Running time

Analyze using the **RAM** (random access machine) model

- Instructions are executed one after another (no parallelism)
- Each instruction takes a constant amount of time
  - Arithmetic (+, -, *, /, %, floor, ceiling)
  - Data movement (load, store, copy)
  - Control (branching, subroutine calls)

**Ignores memory hierarchy!** *(don’t forget: linked lists are awful)*

- This is a very simplified way of looking at algorithms
- Compare algorithms while ignoring hardware
Insertion sort – Running time

On what does the running time depend?
• Number of items to sort (example, 1000 vs 3 numbers)

```plaintext
INSERTION-SORT (A)
1. for j = 2 to A.length
2. key = A[j]
3. i = j - 1
4. while i > 0 and A[i] > key
6. i = i - 1
7. A[i + 1] = key
```
Insertion sort – Running time

On what does the running time depend?
• Number of items to sort (example, 1000 vs 3 numbers)
• How much are they already sorted
  • The hint here is that the inner loop is a **while** loop (not a for loop)

**INSERTION-SORT** (A)
1. **for** j = 2 to A.length
2.  key = A[j]
3.  i = j - 1
4.  **while** i > 0 and A[i] > key
6.   i = i - 1
7.   A[i + 1] = key
**INSERTION-SORT (A)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Cost</th>
<th>Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>for j = 2 to A.length</td>
<td>c1</td>
<td>n</td>
</tr>
<tr>
<td>2.</td>
<td>key = A[j]</td>
<td>c2</td>
<td>n-1</td>
</tr>
<tr>
<td>3.</td>
<td>i = j - 1</td>
<td>c3</td>
<td>n-1</td>
</tr>
<tr>
<td>4.</td>
<td>while i &gt; 0 and A[i] &gt; key</td>
<td>c4</td>
<td>depends</td>
</tr>
<tr>
<td>6.</td>
<td>i = i - 1</td>
<td>c6</td>
<td>depends - 1</td>
</tr>
<tr>
<td>7.</td>
<td>A[i + 1] = key</td>
<td>c8</td>
<td>n-1</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>INSERTION-SORT (A)</th>
<th>Cost</th>
<th>Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. for j = 2 to A.length</td>
<td>2</td>
<td>n</td>
</tr>
<tr>
<td>2. key = A[j]</td>
<td>2</td>
<td>n-1</td>
</tr>
<tr>
<td>3. i = j - 1</td>
<td>2</td>
<td>n-1</td>
</tr>
<tr>
<td>4. while i &gt; 0 and A[i] &gt; key</td>
<td>3</td>
<td>depends</td>
</tr>
<tr>
<td>6. i = i - 1</td>
<td>2</td>
<td>depends - 1</td>
</tr>
<tr>
<td>7. A[i + 1] = key</td>
<td>3</td>
<td>n-1</td>
</tr>
</tbody>
</table>
```
**INSERTION-SORT** (A)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>j = 2</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td><strong>while</strong> j &lt;= A.length</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>key = A[j]</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>i = j - 1</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td><strong>while</strong> i &gt; 0 and A[i] &gt; key</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>i = i - 1</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>A[i + 1] = key</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>j = j + 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Cost</strong></th>
<th><strong>Executed</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>n-1</td>
</tr>
<tr>
<td>2</td>
<td>n-1</td>
</tr>
<tr>
<td>3</td>
<td>depends</td>
</tr>
<tr>
<td>4</td>
<td>depends - 1</td>
</tr>
<tr>
<td>2</td>
<td>depends - 1</td>
</tr>
<tr>
<td>3</td>
<td>n-1</td>
</tr>
<tr>
<td>2</td>
<td>n-1</td>
</tr>
</tbody>
</table>

Total Running Time = 1 + n + (n - 1)(2 + 2 + 3x + (x - 1)(4 + 2) + 3 + 2) = 4n + 9xn - 9x - 2
What is the best case scenario? We pass in an already sorted array A.

**INSERTION-SORT (A)**

1. j = 2
2. while j <= A.length
3. key = A[j]
4. i = j - 1
5. while i > 0 and A[i] > key
7. i = i - 1
8. A[i + 1] = key
9. j = j + 1

<table>
<thead>
<tr>
<th>Cost</th>
<th>Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>n-1</td>
</tr>
<tr>
<td>2</td>
<td>n-1</td>
</tr>
<tr>
<td>3</td>
<td>depends</td>
</tr>
<tr>
<td>4</td>
<td>depends - 1</td>
</tr>
<tr>
<td>2</td>
<td>depends - 1</td>
</tr>
<tr>
<td>3</td>
<td>n-1</td>
</tr>
<tr>
<td>2</td>
<td>n-1</td>
</tr>
</tbody>
</table>

Total Running Time = 1 + n + (n - 1)(2 + 2 + 3x + (x - 1)(4 + 2) + 3 + 2)
= 4n + 9xn - 9x - 2
What is the **best case scenario?**

We pass in an already sorted array $A$.

---

**INSERTION-SORT ($A$)**

1. $j = 2$
2. while $j \leq A$.length
3. key = $A[j]$
4. $i = j - 1$
5. while $i > 0$ and $A[i] >$ key
7. $i = i - 1$
8. $A[i + 1] = key$
9. $j = j + 1$

<table>
<thead>
<tr>
<th>Cost</th>
<th>Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>2</td>
<td>$n-1$</td>
</tr>
<tr>
<td>2</td>
<td>$n-1$</td>
</tr>
<tr>
<td>3</td>
<td>depends</td>
</tr>
<tr>
<td>4</td>
<td>depends - 1</td>
</tr>
<tr>
<td>2</td>
<td>depends - 1</td>
</tr>
<tr>
<td>3</td>
<td>$n-1$</td>
</tr>
<tr>
<td>2</td>
<td>$n-1$</td>
</tr>
</tbody>
</table>

**Total Running Time**

$= 1 + n + (n - 1)(2 + 2 + 3x + (x - 1)(4 + 2) + 3 + 2)$

$= 4n + 9xn - 9x - 2$

$= 4n + 9n - 9 - 2$

$= 13n - 11$
**What is the worst case scenario?**

We pass in a reverse sorted array $A$.

<table>
<thead>
<tr>
<th>Insertion-Sort ($A$)</th>
<th>Cost</th>
<th>Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $j = 2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2. while $j \leq A.length$</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>3. key = $A[j]$</td>
<td>2</td>
<td>$n-1$</td>
</tr>
<tr>
<td>4. $i = j - 1$</td>
<td>2</td>
<td>$n-1$</td>
</tr>
<tr>
<td>5. while $i &gt; 0 \text{ and } A[i] &gt; key$</td>
<td>3</td>
<td>depends</td>
</tr>
<tr>
<td>7. $i = i - 1$</td>
<td>2</td>
<td>depends - 1</td>
</tr>
<tr>
<td>8. $A[i+1] = key$</td>
<td>3</td>
<td>$n-1$</td>
</tr>
<tr>
<td>9. $j = j + 1$</td>
<td>2</td>
<td>$n-1$</td>
</tr>
</tbody>
</table>

Total Running Time $= 1 + n + (n - 1)(2 + 2 + 3x + (x - 1)(4 + 2) + 3 + 2)$

$= 4n + 9xn - 9x - 2$
What is the **worst** case scenario? We pass in a reverse sorted array $A$. $x = n/2$ on average

<table>
<thead>
<tr>
<th>INSERTION-SORT ($A$)</th>
<th>Cost</th>
<th>Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $j = 2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2. while $j \leq A.length$</td>
<td>1</td>
<td>$n$</td>
</tr>
<tr>
<td>3. key = $A[j]$</td>
<td>2</td>
<td>$n-1$</td>
</tr>
<tr>
<td>4. $i = j - 1$</td>
<td>2</td>
<td>$n-1$</td>
</tr>
<tr>
<td>5. while $i &gt; 0$ and $A[i] &gt; key$</td>
<td>3</td>
<td>depends</td>
</tr>
<tr>
<td>7. $i = i - 1$</td>
<td>2</td>
<td>depends - 1</td>
</tr>
<tr>
<td>8. $A[i + 1] = key$</td>
<td>3</td>
<td>$n-1$</td>
</tr>
<tr>
<td>9. $j = j + 1$</td>
<td>2</td>
<td>$n-1$</td>
</tr>
</tbody>
</table>

Total Running Time = $1 + n + (n - 1)(2 + 2 + 3x + (x - 1)(4 + 2) + 3 + 2) = 4n + 9xn - 9x - 2 = 4n + 9nn/2 - 9n/2 - 2 = 4.5n^2 - 0.5n - 2$
Best, Worst, and Average

We usually concentrate on worst-case

• Gives an upper bound on the running time for any input
• The worst case can occur fairly often
• The average case is often relatively as bad as the worst case
Chapters from The book

Recommended, but not necessary
• Chapter 1: introduction to algorithms

This lecture
• Chapter 2: insertion sort, analyzing/designing algorithms

Next lecture
• Chapter 3: asymptotic notation

Next, next lecture
• Chapter 4: divide-and-conquer