MERRIMACK COLLEGE

CSC6023 - Advanced Algorithms

Course Introduction and Asymptotic Notations - Week 1

Patrick Neff

You're Welcome! Week 1

This is a very fast paced course, so this is the course introduction AND the revision of the asymptotic notation for algorithms efficiency

MERRIMACK COLLEGE

Agenda Week 1 Presentation

Course Introduction

- Course format
  a. Evaluation
  b. Timeline

Asymptotic Notation

- Basics
  a. Number of operations
  b. Big-O, Big-Omega, and Big-Theta
- MCSS - a problem to exercise
  a. The polynomial - cubic - solution
  b. The polynomial - quadratic - solution
  c. The linear solution

Course Format

People

Instructor

- Patrick Neff
  - neffp@merrimack.edu

Tutors Available

- Check the Canvas announcements for the current tutor availability

Read the Syllabus!!

Course Format

Live sessions

Read the Syllabus!!

MERRIMACK COLLEGE

8-week long course

Each week typically has:

- A **live weekly class** session **Monday 6:30-8:30** PM EST via Zoom
  - Worksheet assignment during live class with deadline for Friday
  - Project assignment during live class with deadline for next Monday
  - Discussions due Friday, with 2 comments due the following Monday
- **Office hours Thursday 6:00-7:00** PM via Zoom
- Quiz available Friday
  - Due next Monday

Course Format Evaluation

Read the Syllabus!

You will be evaluated based on

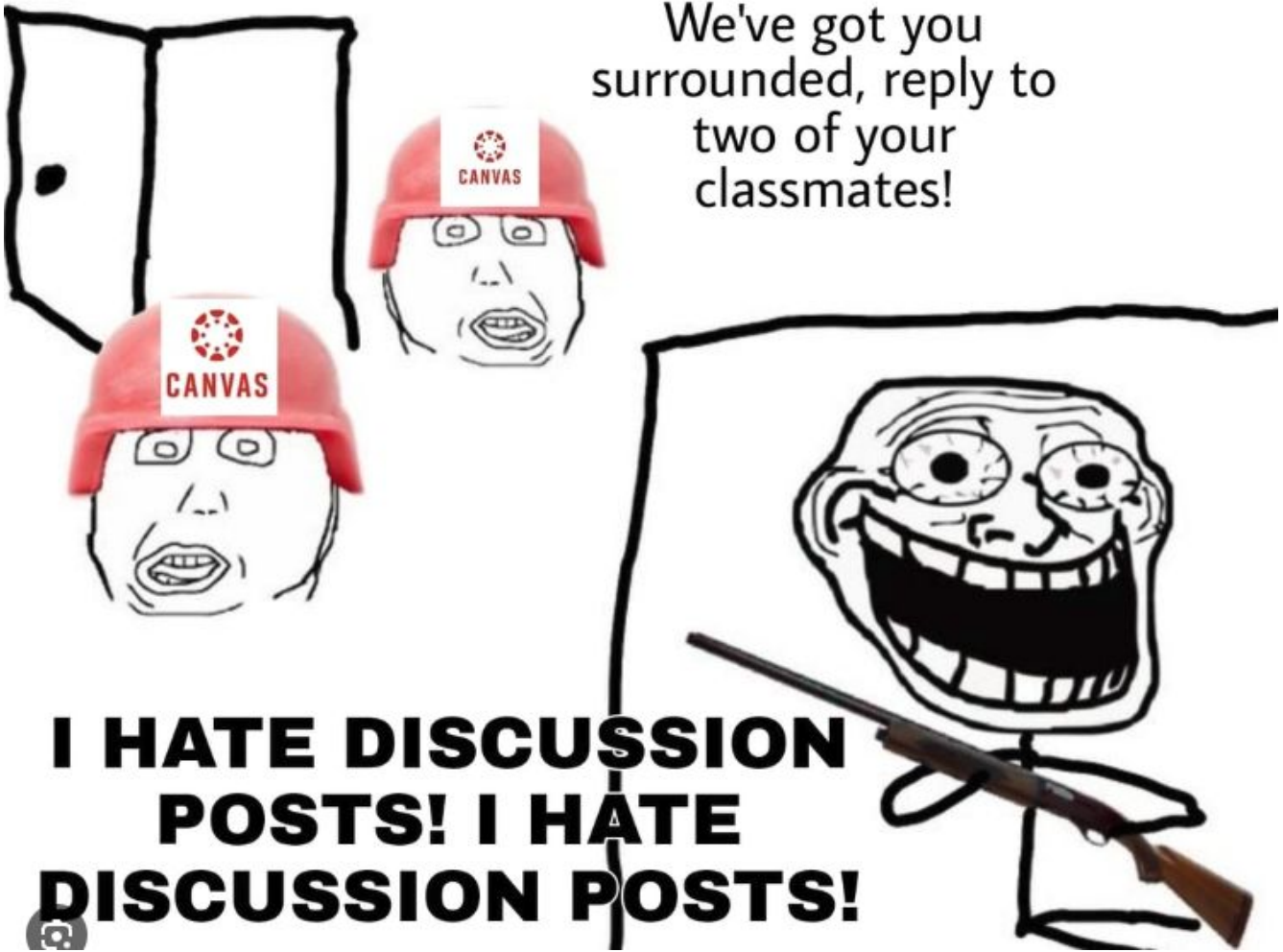
Assignment Type	Points	Percentage
Coding Projects	100 each	35%
Worksheets	100 each	21%
Quizzes	100 each	21%
Discussions	100 each	8%
Final Exam	100	20%

Late Penalties: All weekly evaluated tasks are due by their stated deadline; the late penalties are a reduction of 10% of the evaluation, plus 2% for each full day of delay.

There is a cut-off final deadline at the last day of class (last Friday of the 8th week) - a hard deadline.

The hard deadline for the final exam is last week's Sat.

- There are 8 modules, the first 7 of which all follow the same assignment pattern including:
  - 1 graded discussion (initial post due Friday by midnight; two responses to others' posts due by the next Monday at midnight)
  - 1 worksheet, generally a simpler coding task or tasks (due Friday of the week by midnight)
  - 1 programming assignment, generally a more complex coding task (due Monday of the next week by midnight)
  - 1 quiz (available from Friday of each module's week and due the next Monday by midnight - in other words you have four days to complete each quiz; I encourage you to leave them open and to revisit them before submitting)
- The 8th module/week has no assignments except a final exam which must be taken in a four hour block of time. The exam is open book and open notes; you may use all course projects/worksheets for the exam but you may not use the internet (besides documentation referenced in the course materials). The final exam is due by midnight of Module 8's Saturday.
- This is a full course, so please try to be mindful of deadlines. I try to grade quickly so it helps if all assignments are in on time. However, this course is designed to be convenient for working professionals so I will try to be as flexible as possible if you let me know you need more time WELL BEFORE THE DEADLINE and you only do this once or twice.


We've got you surrounded, reply to two of your classmates!
CANVAS
CANVAS
I HATE DISCUSSION POSTS! I HATE DISCUSSION POSTS!

Read the Syllabus!!

Course Format

You final letter grade will follows this scale

A	A-	B+	B	B-	C+	C	C-	F
95 and up	90 to 94.9	87 to 89.9	83 to 86.9	80 to 82.9	77 to 79.9	73 to 76.9	70 to 72.9	69.9 and low

Evaluation

There are no grade D+, D, or D- in graduate courses.

An average grade B is required to the Masters of Science in Computer Science.

Course Format

Timeline

8-week course

1. Asymptotic Notation Review
2. Dynamic Programming
3. Greedy Algos
4. Amortized Analysis
5. Linear programming
6. Randomized Algorithms
7. Advanced Data Structures
8. Approximation Algorithms

Read the Syllabus!!

“Algorithm” etymology

What is an “algorithm”?

“The word ‘algorithm’ itself is quite interesting; at first glance it may look as though someone intended to write ‘logarithm’ but jumbled up the first four letters. The word did not appear in *Webster’s New World Dictionary* as late as 1957; we find only the older form ‘algorism’ with its ancient meaning, i.e., the process of doing arithmetic using Arabic numerals. In the middle ages, abacists computed on the abacus and algorists computed by algorithm. Following the middle ages, the origin of this word was in doubt, and early linguists attempted to guess at its derivation by making combinations like *algiros* [painful] + *arithmos* [number]; others said no, the word comes from ‘King Algor of Castile.’ Finally, historians of mathematics found the true origin of the word algorithm: it comes from the name of a famous Persian textbook author, Abu Ja‘far Mohammed ibn Musa al-Khowarizmi (c. 825) - literally, ‘Father of Ja‘far, Mohammed, son of Moses, native of Khowarizm.’ Khowarizm is today the small Soviet city of Khiva. Al-Khowarizmi wrote the celebrated book Kitam al jabr w’al-muqabala (‘Rules of restoration and reduction’); another word, ‘algebra,’ stems from the title of his book, although the book wasn’t really very algebraic.”

(Page 1)
-The Art of Computer Programming: Volume 1/Fundamental Algorithms
Donald E. Knuth, 1968

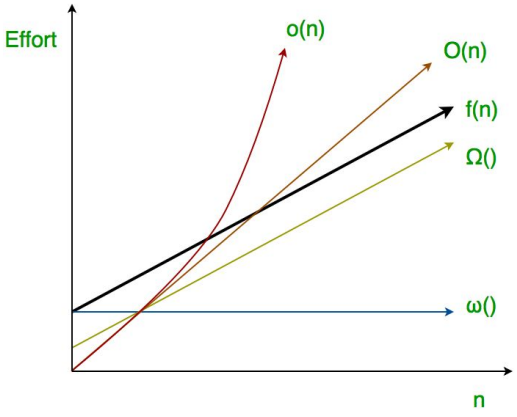
https://en.wikipedia.org/wiki/Muhammad_ibn_Musa_al-Khwarizmi

By the 1950s the word “algorithm” with the modern meaning of a precise method for a mathematical task was in use, most prominently with “Euclid’s algorithm,” a process for finding the greatest common divisor of two numbers.

Why study algorithms?

1. Traditionally an essential part of a computer science curriculum
2. Important for understanding what exactly we are asking the computer to do
3. Important for understanding scalability
4. Job interviews

Week 1 Asymptotic notations review



MERRIMACK COLLEGE

How to know what to expect for your algorithm?

When you code your algorithm there is an amount of time that the code is expected to take considering the "size" of your problem. This amount of time is commonly referred as time complexity of the algorithm and it is frequently expressed by an asymptotic notation.

Asymptotic?

Etymology

From the Greek:

a = non

symptein = to meet

Meaning: not meeting

We call asymptotic notation because it involves bounding functions. The actual function of our algorithm ***does not meet*** the asymptote as they approach infinity.

Basics: Number of Operations

Counting the loops works usually well, but it is a simplification

Do you remember recursion?

Giving an algorithm

- What to expect in terms of relevant operations
- For example, to find the largest element in an array

```
def larg(a):
    largest = a[0]
    for x in a:
        if (x > largest):
            largest = x
    return largest
```

```
def larg(a):
    largest = a[0]
    for i in range(1,len(a)):
        if (a[i] > largest):
            largest = a[i]
    return largest
```

- A *n* sized array needs about *n* comparisons
- Another example, find the largest element in a square matrix
- A square matrix of order *n* needs about n^2 comparisons

```
def larg(a):
    largest = a[0][0]
    for i in range(len(a)):
        for j in range(len(a)):
            if (a[i][j] > largest):
                largest = a[i][j]
    return largest
```

Basics: Number of Operations

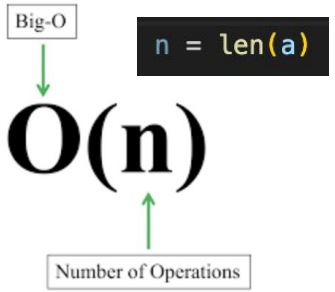
For iterative code, you can count the nested loops

For recursive code you should estimate the number of calls

Giving an algorithm

- What to expect in terms of relevant operations
- For example, to find the largest element in an array

```
def larg(a):
    largest = a[0]
    for i in range(1,len(a)):
        if (a[i] > largest):
            largest = a[i]
    return largest
```

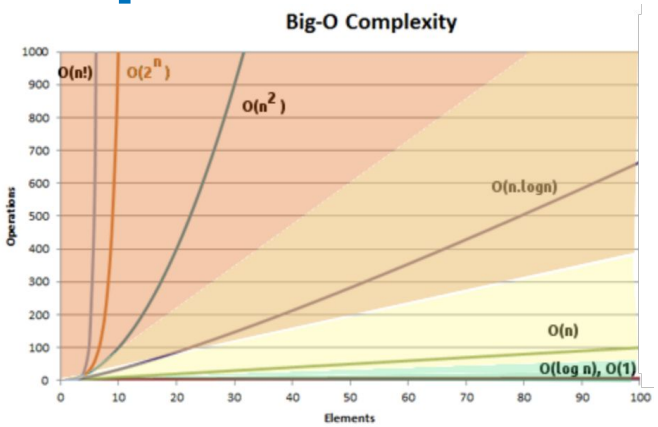


- A *n* sized array needs about *n* comparisons
- Another example, find the largest element in a square matrix
- A square matrix of order *n* needs about n^2 comparisons

```
def larg(a):
    largest = a[0][0]
    for i in range(len(a)):
        for j in range(len(a)):
            if (a[i][j] > largest):
                largest = a[i][j]
    return largest
```

$O(n^2)$

Basics Number of Operations

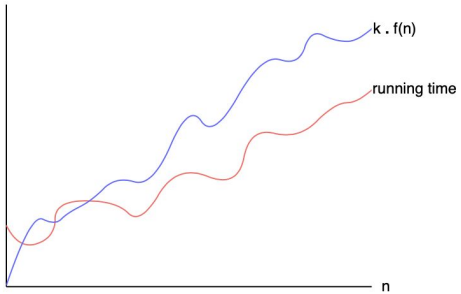


The classes of complexity

Big-O	complexity	examples
O(c)	constant	First element of an array
O(log n)	logarithmic	Binary search
O(n)	linear	Find the largest in an array
O(n log n)	log-linear	Merge sort
$O(n^c)$	polynomial	Selection sort (n^2 - quadratic)
$O(c^n)$	exponential	Find all subsets of a set (2^n)
O(n!)	factorial	Find all permutations of a set

- Big O is an upper bound... pessimistic (safe)

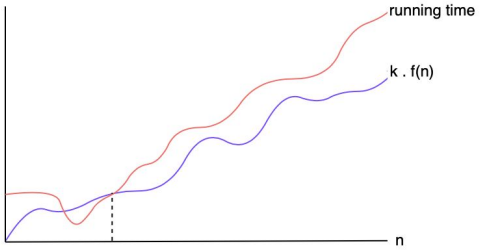
Basics Big-O, Big-Omega, and Big-Theta



The functions of complexity

- The **Big-O** is a function that is an upper bound of the algorithm
  - It safely states the longest the algorithm will take
- Formally, if a function *t(n)* denotes the number of operations taken by a given algorithm, this function is said to be in *O(g(n))* if *t(n)* is bounded above by some constant multiple of *g(n)* for all large *n*
  - For example, that an algorithm is in *O(log n)* means that this algorithm will require at most *t(n)* operations and *t(n) <= c log n* for all values of *n* above a certain (small) value

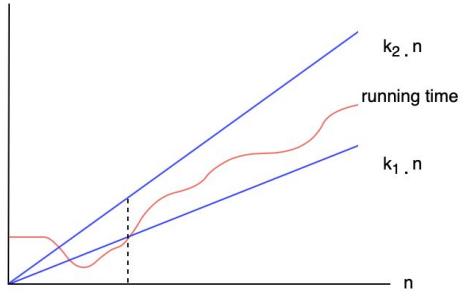
Basics Big-O, Big-Omega, and Big-Theta



The functions of complexity

- The **Big-Omega** is a function that is a lower bound of the algorithm
  - It safely states the shortest the algorithm will take
- Formally, if a function *t(n)* denotes the number of operations made by a given algorithm, this function is said to be in $\Omega(g(n))$ if *t(n)* is bounded below by some constant multiple of *g(n)* for all large *n*
  - For example, that an algorithm is in $\Omega(\log n)$ means that this algorithm will require at least *t(n)* operations and $t(n) >= c \log n$ for all values of *n* above a certain (small) value

Basics Big-O, Big-Omega, and Big-Theta

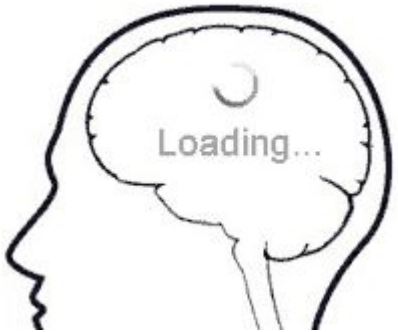


MERRIMACK COLLEGE

The functions of complexity

- The **Big-Theta** is a function that express both a lower and an upper bound of the algorithm
  - It states at the same time the information stated by Big-O and Big-Omega
- Formally, if a function *t(n)* denotes the number of operations taken by a given algorithm, this function is said to be in $\Theta(g(n))$ if *t(n)* is bounded above by some constant multiple of *g(n)* and bounded below by some constant multiple of *g(n)* for all large *n*
  - For example, that an algorithm is in $\Theta(\log n)$ means that this algorithm will require at least *t(n)* operations and $t(n) >= c_1 \log n$ and at most *t(n)* operations and $t(n) <= c_2 \log n$ for all values of *n* above a certain (small) value

MCSS
A problem to exercise



Maximum Contiguous Subsequence Sum

- Given an array of *n* integers (positives and negatives), what is the subsequence of contiguous elements that delivers the highest sum of its elements?
  - For example, for the array
    a = [-2 , 11, -4, 13, -5, 2]
    the answer is the sum 20 obtained by
    a[1:4] i.e. [-2 , **11**, **-4**, **13**, -5, 2]
- If the array was [5, -1, 56, -3, -18, 22, -9] what would be the answer?
  - Seriously, try it now!

MCSS
A problem to exercise

Maximum Contiguous Subsequence Sum

- a = [5, -1, 56, -3, -18, 22, -9]

5
-1
56
-3
-18
22
-9

5 -1
-1 56
56 -3
-3 -18
-18 22
22 -9

5 -1 56
-1 56 -3
56 -3 -18
-3 -18 22
-18 22 -9

5 -1 56 -3
-1 56 -3 -18
56 -3 -18 22
-3 -18 22 -9

5 -1 56 -3 -18
-1 56 -3 -18 56
56 -3 -18 22 -9

5 -1 56 -3 -18 56 -9

61

5 -1 56 -3 -18 22
-1 56 -3 -18 22 -9

- This is a difficult problem to solve without a systematic approach … how much does this one cost?

MCSS
A problem to exercise

MERRIMACK COLLEGE

Maximum Contiguous Subsequence Sum

- The naive solution: test all possible subsequences
  - For all elements i of the array
    - For all elements j from i to the end
      - Compute the sum from i to j
        - … and keep the largest sum

1

```
def MCSS(a):
    largest = 0
    for i in range(len(a)):
        for j in range(i,len(a)):
            acc = 0
            for k in range(i,j+1):
                acc += a[k]
            if (acc > largest):
                largest = acc
    return largest
```

- First loop to choose the start of the subsequence
- The second loop to choose the end of the subsequence
- The third loop to add it up

MCSS
A problem to exercise

Maximum Contiguous Subsequence Sum

- The naive solution: test all possible subsequences
  - For all elements i of the array
    - For all elements j from i to the end
      - Compute the sum from i to j
        - … and keep the largest sum

1

```
def MCSS(a):
    largest = 0
    for i in range(len(a)):
        for j in range(i,len(a)):
            acc = 0
            for k in range(i,j+1):
                acc += a[k]
            if (acc > largest):
                largest = acc
    return largest
```

- What is the complexity of this algorithm?

  Polynomial - n^c
  Cubic - n^3

- Can you do better?

MCSS
A problem to exercise

MERRIMACK COLLEGE

Maximum Contiguous Subsequence Sum

- A small improvement: test all possible subsequences, but sum as you go
  - For all elements i of the array
    - For all elements j from i to the end
      - Keep on computing the sum from i to j
        - … and keep the largest sum

2

```
def MCSS(a):
    largest = 0
    for i in range(len(a)):
        acc = 0
        for j in range(i,len(a)):
            acc += a[j]
            if (acc > largest):
                largest = acc
    return largest
```

- First loop to choose the start of the subsequence
- The second loop to choose the end of the subsequence and compute the sum as you go

MCSS
A problem to exercise

MERRIMACK COLLEGE

Maximum Contiguous Subsequence Sum

- A small improvement: test all possible subsequences, but sum as you go
  - For all elements i of the array
    - For all elements j from i to the end
      - Keep on computing the sum from i to j
        - … and keep the largest sum

2

```
def MCSS(a):
    largest = 0
    for i in range(len(a)):
        acc = 0
        for j in range(i,len(a)):
            acc += a[j]
            if (acc > largest):
                largest = acc
    return largest
```

- What is the complexity of this algorithm?

  Polynomial - n^c
  Quadratic - n^2

- Better, but can you still do better?

MCSS
A problem to exercise

Maximum Contiguous Subsequence Sum

- What if you discard subsequences that start with negative values
  - Start with i as the first element
  - For all elements j of the array
    - Keep on computing the sum from i to j
      - … and keep the largest sum
    - If the current sum is negative discard it

3

```
def MCSS(a):
    largest, acc, i = 0, 0, 0
    for j in range(len(a)):
        acc += a[j]
        if (acc > largest):
            largest = acc
        elif (acc < 0):
            i = j + 1
            acc = 0
    return largest
```

- The first, and only, loop advances the subsequence window computing as you go, and changing the beginning when the current window has a negative sum

MCSS
A problem to exercise

Maximum Contiguous Subsequence Sum

- What if you discard subsequences that start with negative values
  - Start with i as the first element
  - For all elements j of the array
    - Keep on computing the sum from i to j
      - … and keep the largest sum
    - If the current sum is negative discard it

3

```
def MCSS(a):
    largest, acc, i = 0, 0, 0
    for j in range(len(a)):
        acc += a[j]
        if (acc > largest):
            largest = acc
        elif (acc < 0):
            i = j + 1
            acc = 0
    return largest
```

- What is the complexity of this algorithm?

  Linear - ***n***

- That is as good as it gets since at least all elements have to be added once

MCSS Hands-on

Task #1 for this week's worksheet

- Create a program that randomizes a vector of 1000 positive and negative integers, then it finds the maximum contiguous subsequence sum value
  - Prints out the maximum value

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room

Deadline: This Friday 11:59 PM EST

MCSS Hands-on

Task #2 for this week's worksheet

- Extends the program of Task #1 to output not only the maximum value, but also the initial and final index of the elements to compute the maximum value

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room

Deadline: This Friday 11:59 PM EST

First Assignment

Project #1 - this week's Assignment

- Create a program that implements a sort algorithm of your choice and applies it to a random vector of 1,000 elements
- Repeat the process applying it to random vectors of 2,000, 3,000, … up to 10,000 elements
- Compute the time complexity of your algorithm and verify if the time it takes to your 1,000 to 10,000 corresponds to the time complexity prediction.
- Besides the implementation of your program, write a short report describing your experiences and conclusion.

You can use cProfile to measure the time of your code execution

cProfile documentation page: https://docs.python.org/3/library/profile.html

First Assignment

Project #1 - this week's Assignment

- This program must be your own, do not use someone else's code
- Any specific questions about it, please bring to the Office hours meeting this Friday or contact me by email
- This is a challenging program to make sure you are mastering your Python programming skills, as well as your asymptotic analysis understanding
- Don't be shy with your questions

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room

Deadline: Next Monday 11:59 PM EST

Worksheet and Project Format

Format Guidelines:

Please submit both worksheets and projects in one zipped folder named according to the following example:

SmithJaneMod1Worksheet.zip

SmithJaneMod1PA.zip

(pa = Programming Assignment)

Please follow this format even if your work only requires one simple Python file.

That’s all for today folks!

This week’s tasks

- Discussion
  - Deadline: this Friday 11:59 PM EST
- Tasks #1 and #2 for the worksheet
  - Deadline: this Friday 11:59 PM EST
- Quiz #1 to be available this Friday
  - Deadline: Next Monday 11:59 PM EST
- Project #1 assignment
  - Deadline: Next Monday 11:59 PM EST

- Try all exercises seen in class and consult the reference sources, as the more you practice, the easier it gets

Next week

- Transform-and-conquer algorithms

- Don’t let work pile up!
- Don’t be shy about your questions

MERRIMACK COLLEGE

Have a Great Week!