mp_train due Wednesday!

Optional extra credit will be worth extra credit, currently 0 points

Can request a 24-hour extension policy

https://forms.gle/1KreTgcfJb1fXQeQ6

Request one-on-one office hours by emailing with a list of times
Reminder: Sign up for exam 1!

Sign up at: https://cbtf.engr.illinois.edu/sched

Email DRES accomodations: http://cbtf@illinois.edu/
Exam 1: Python Fundamentals

**Conditionals:** How to write, how to read, relation to logic

**Lists:** How to look up elements, how to write 1D and 2D lists

**Loops:** How to write and evaluate loops
Exam 1: Logic

**Logic Operators:** Understand truth tables and underlying meaning

**Logic Equivalencies:** Negation, converse, contrapositive

**How to write, read, evaluate and code:**

- Compound propositions and predicates
- Quantifiers and compound quantifiers
Exam 1: Lists

**List ADT:** How to write and use common list functions

**Linked List:** How to write and use, Big O efficiency of methods

**Array List:** How to write and use, Big O efficiency of methods
Exam 1: Stacks and Queues

How to write and use, Big O efficiency, how order is maintained
Exam 1: Coding Questions

You will have a choice of coding questions on exam

1. Code up a compound predicate with quantifiers

2. Create a 2D matrix with a specific set of values
Learning Objectives

Motivate the need for sorting

Sample the space of available sorting algorithms

Introduce recursion
The Sorting Problem

Given a collection of objects, $C$, with comparable values, order the objects such that $\forall x \in C, x_i \leq x_{i+1}$

Input: 8 4 3 1 2 5 6 9 0 7

Output: 0 1 2 3 4 5 6 7 8 9
Sorting leads to efficient searching

Search(7)
Sorting leads to better visualization
Sorting is a fundamental problem in CS

Many algorithms begin with or include a sorting step

Fundamental sorting algorithms are great for mastering concepts

Sorting algorithms are a classic introduction to algorithms
Optimizing sort is an ongoing challenge

**GraySort:** Sort rate (TBs / minute) achieved while sorting a very large amount of data (currently 100 TB minimum).

**CloudSort:** Minimum cost (Dollars) for sorting a very large amount of data on a public cloud. (currently 100 TB).

**MinuteSort:** Amount of data that can be sorted in 60 seconds or less.

**TeraByeSort:** Elapsed time to sort 1 TB of data

Competition details: [http://sortbenchmark.org/](http://sortbenchmark.org/)
Sorting Algorithms we will discuss:

- SelectionSort
- InsertionSort
- MergeSort
- QuickSort
Sorting by hand (humanSort)

Input:

Output:
Claim: Any deterministic comparison-based sorting algorithm must perform $O(n \log n)$ comparisons to sort $n$ objects.
SelectionSort

1. Find the $i$-th smallest value
2. Place it at position $i$ via swap
3. Repeat for $0 \leq i \leq n - 1$
SelectionSort Efficiency

\[ (n=4) \]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
1 & 2 & 3 & \\
2 & 3 & & \\
& & & \\
\end{array}
\]
SelectionSort Efficiency (large n)
def selectionSort(inList):
    n = len(inList)
    for i in range(n):
        mindex = i
        for j in range(i+1, n):
            if inList[j] < inList[mindex]:
                mindex = j
        inList[i], inList[mindex] = inList[mindex], inList[i]
InsertionSort

1. Divide array into two parts
2. Insert the first unsorted item into the sorted position
3. Repeat until all items are sorted
InsertionSort “Insert”

1 2 4 5 7 3 8

1 2 4 5 3 7 8

1 2 4 3 5 7 8

1 2 3 4 5 7 8
def insertionSort(inList):
    n = len(inList)
    for i in range(1, n):
        val = inList[i]
        j = i - 1
        while j >= 0 and inList[j] > val:
            inList[j+1]=inList[j]
            j -= 1
        inList[j+1]=val
Selection vs InsertionSort

def selectionSort(inList):
    n = len(inList)
    for i in range(n):
        mindex = i
        for j in range(i+1, n):
            if inList[j] < inList[mindex]:
                mindex = j
        inList[i], inList[mindex] = inList[mindex], inList[i]

def insertionSort(inList):
    n = len(inList)
    for i in range(1, n):
        val = inList[i]
        j = i - 1
        while j >= 0 and val < inList[j]:
            inList[j+1]=inList[j]
            j -= 1
        inList[j+1]=val
Sorting Algorithms we will discuss:

SelectionSort

InsertionSort

MergeSort

QuickSort
Divide and Conquer Algorithms

*Recursively break a problem into sub-problems until the problems become simple enough to solve directly*
Recursion

The process by which a function calls itself directly or indirectly is called recursion.
Recursive For Loop

```python
for i in range(10):
    print(i)

def recursiveFor(n):
    if n == 0:
        print(n)
        return
    recursiveFor(n-1)
    print(n)
```
Recursion

**Base Case:** What is the smallest sub-problem? What is the trivial solution?

**Recursive Step:** How can I systematically reduce my problem to an easier one?

**Combining:** How can I build my solution from recursive pieces?
def findMax(inList):

Recursive findMax
def fib(n):
    # Recursive Fibonacci algorithm
Recursive Array Sorting

Base Case:

Recursive Step:

Combining: