

CS 5008

FINAL QUIZ REVIEW

FULL SEMESTER REVIEW

LOGAN SCHMIDT

FINAL QUIZ GUIDELINES

- The quiz is on Canvas, labeled “Final Quiz”, and it will open at 9:32am PT and close at 11:35am PT. You have 120 minutes to complete it from when you start it, up to 11:35am; the extra is ‘buffer’ time .
- The quiz is meant to take ~1-1.5 hours, but you will have the full time to take it.
- This quiz is to be completed **without** any help from outside sources, including websites, other course materials, your own course notes, the Internet, etc. It is "closed book".
- You may not copy answers from another student, or allow other students to copy your answers.
- Cheating of any kind is considered a violation of Northeastern University’s academic integrity policy and will be reported to the Dean of Student Affairs.

FINAL QUIZ GUIDELINES

- 1. You may not consult, reference, or look at any outside sources, including other course materials or your own notes, during the quiz.
- 2. You may not use any additional electronic devices during this quiz.
- 3. You may not visit any websites or have any other programs running on your computer other than this Canvas webpage during this quiz, and the Zoom meeting for this class.
- 4. If you are a remote student, you must turn on your camera for the duration of this test.
- 5. You may not communicate with other students or people other than the instructor for the duration of this test.

FINAL QUIZ GUIDELINES

- Instructions for this quiz:

Once you have answered a question on this quiz, you cannot go back to it or see it again. Answer all questions to the best of your ability. Each individual question is worth the same amount of points (1); there are 30 total questions.

If you have questions, you may send a private message to the instructor in the Zoom meeting asking for clarification. If you are in the class in person, you may approach the front of the classroom and ask the instructor your question.

- PS: monitoring for cheating is the absolute –worst- part of my job. Please don't give me any reason to think you're cheating.

AGENDA

1. Linked Lists and Doubly Linked Lists, Stacks and Queues
 2. Assembly language, debugging, the compiler
 3. Sorting: brute force, quadratic sorts, divide & conquer
 4. Trees: Binary Search Trees
 5. Hashmap, Min Heaps, Priority Queues
 6. Graphs: adjacency list vs adjacency matrix, BFS, DFS
 7. Greedy algorithms
 8. Dynamic programming
-

WARNING

- The following slides were assembled by students. The instructor has gone through them and corrected most errors of statement and emphasis.
 - However! The best way to review is by going back through the videos, LEARNING ACTIVITIES, and labs to make sure that you understand the topics we've covered in this course
 - Don't rely only on these slides – these were an in-class exercise, which honestly wasn't as successful as I hoped
 - Do your own review, and make sure you review the learning activities and ask questions about any questions you don't feel you understand.
-

LINKED LISTS –

- Basic Operations
 - Create empty linked list - $O(1)$
 - Add node to front $O(1)$
 - Insert node at position or back - $O(n)$
 - Remove node at position - $O(n)$ unless you have pointers to node & node before
- Time complexity & Space complexity
- Comparison between array & linked list
 - Dynamically Resized Arrays / vectors
- Common mistakes:
 - Malloc & Free -> Valgrind (When to malloc or free?)
 - Segfault -> Malloc, Free, NULL pointers

DOUBLY LINKED LISTS

- Basic Operations
 - Create list/node
 - Add node
 - pushFront
 - pushback
 - Insert at position
 - Remove node
- Time complexity & Space complexity
- Comparison between array & linked list
 - Dynamically Resized Arrays
- Common mistakes:
 - Malloc & Free -> Valgrind (When to malloc or free?)
 - Segfault -> Malloc, Free, NULL pointers

STACKS – ABSTRACT DATA STRUCTURE

- Data Structure (LIFO)
 - Array
 - LinkedList
- Basic Operations
 - Create stack/node
 - Add node (push)
 - Remove node (pop)
- Time complexity & Space complexity
- Common mistakes:
 - Malloc & Free -> Valgrind (When to malloc or free?)
 - Segfault -> Malloc, Free, NULL pointers

QUEUES – ABSTRACT DATA STRUCTURE

- Data Structure (FIFO)
 - Array
 - LinkedList
- Basic Operations
 - Create queue/node
 - Add node (pushBack)
 - Remove node (popFront)
- Time complexity & Space complexity
- Common mistakes:
 - Malloc & Free -> Valgrind (When to malloc or free?)
 - Segfault -> Malloc, Free, NULL pointers

ASSEMBLY LANGUAGE

C is actually built on top of assembly language.

Assembly language is the human readable version of machine code that the machine understands.(ie.binary 1010101, x86-64)

Learning assembly will be beneficial for
understanding how to debug our programs,
perform reverse engineering,
just write some really fast and optimized programs,
understand and implement compilers.

Registers in assembly we are limited to 16 general-purpose registers.

MOVL instruction moves an 32-byte word of data

MOVW instruction moves an 16-byte word of data

MOVQ instruction moves an 8-byte word of data

MOVB instruction moves a 1-byte of data

usage of register

%EAX represent a 4-byte word

%AX

%RAX

%AL

ex.

movl %al, %lx Pass the 32-bit EAX register value to the 32-bit EBX register value

DEBUGGING

- File Pointer in C and FILE/IO
- **Debug tools:**
 - a) 1. Adding print statements
 - b) 2. Use gdb
- **Use gdb to debug:**
 - Compile with “gcc -g debug.c -o debug”
 - Launch gdb with “gdb ./debug”
 - Run with “run”
 - “n” for executing a function call without stepping into it
 - “s” for step forward
 - “bt” for backtracing
 - “br” for inserting breaking points

```
int main(){  
    FILE* filePointer;  
    filePointer = fopen("data.txt","r");  
    int buffer; // Buffer holds integer values.  
    while(1==fscanf(filePointer,"%d",&buffer)){  
        printf("I read: %d\n",buffer);  
    }  
    fclose(filePointer);  
    return 0;  
}
```

THE COMPILER

- The compiler takes the output from the preprocessor and makes sure that the code is valid. Then, it will produce an assembly file.
- There are three parts of compiler front end: Lexical Analysis, Semantic Analysis, and Syntax
 - lexical analyzer is to convert strings to tokens so we can see if they are in order
 - The semantic analyzer evaluates the semantics of our tokens including the type being assigned to a variable, the return type of functions, and the number of arguments being passed to each function
 - We then can create an abstract syntax tree to ensure that we aren't using Java or Python syntax in our programs. Then, the assembly code will be generated.

BIG O AND ALGORITHMIC COMPLEXITY, BRUTE FORCE

- what's an algorithm:
 - a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
- what's RAM model of computation:
 - a “good enough” and “consistent” method to evaluate the performance of our algorithms. regardless of which language and machine you used
 - $+$ $-$ $*$ $/$: each one operation takes 1 time step
 - loops: each take a series of time steps
 - `Array[0]`: memory access: 1 time steps
- brute force: linear search (no sorting)

QUADRATIC SORTS

		bubble	selection	insertion
general description		compare adjacent items, swap the greater item to the right. bubble up the greatest item all the way to the right	at each round: select the smallest item in unsorted array, swap it with the front item of unsorted array.	for each item $A[j]$, insert it to a proper position to the subarray that is before it: $A[0, \dots, j-1]$.
time complexity	best	$O(n^2)$, even for an already sorted array, still need to go through all the comparing and swapping process	$O(n^2)$, same as bubble sort	$O(n)$, for an already sorted array, just compare adjacent items then its proper position is found, no swapping.
	worst	$O(n^2)$	$O(n^2)$ (slightly better than bubble sort, reduce searching times when sorting later part of the array)	$O(n^2)$, a reversed order array
space complexity		$O(1)$	$O(1)$	$O(1)$

MERGE SORT – DIVIDE AND CONQUER

MergeSort(Array,left,right)

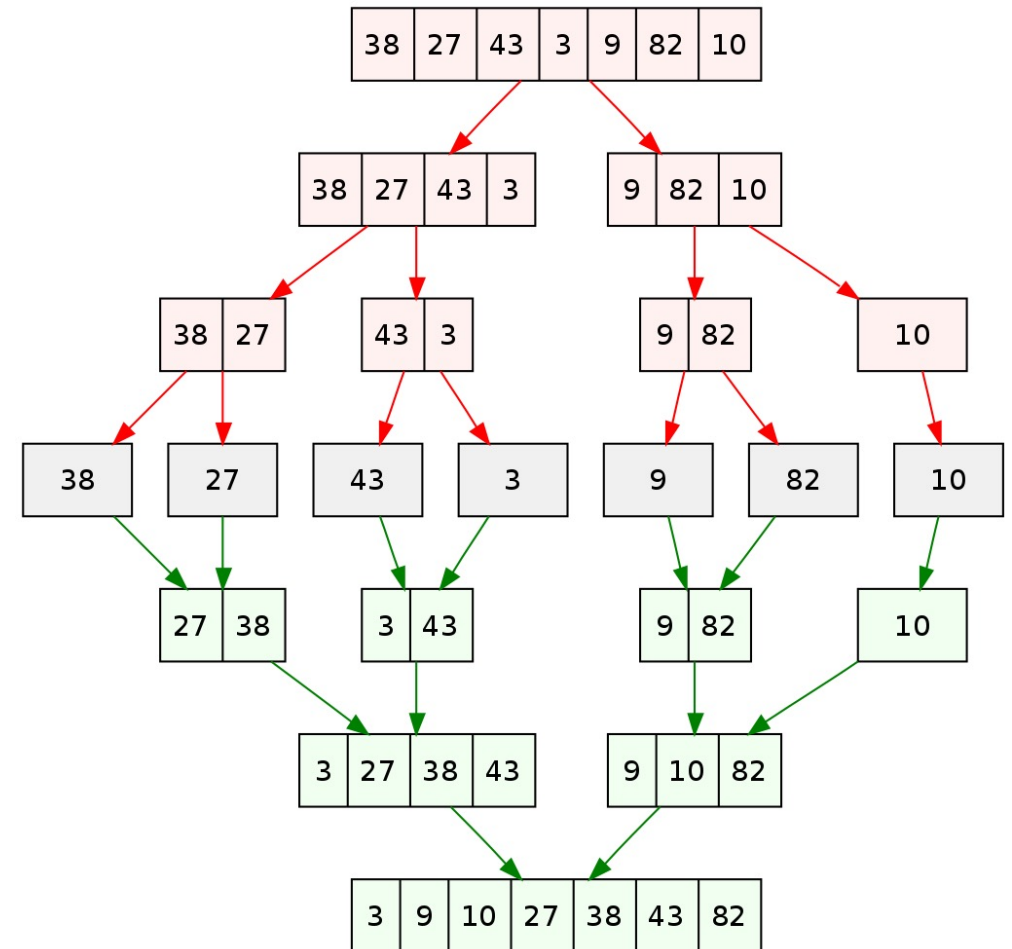
If ArraySize>1:

Divide Array in half

Call MergeSort on 1st half

Call MergeSort on 2nd half

Merge two results



MERGE SORT – TIME COMPLEXITY

- Time Complexity: $O(n \log n)$
- Substitution Method:

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kcn$$

$$n = 2^k$$

$$\frac{n}{2^k} = 1$$

$$T\left(\frac{n}{2^k}\right) = T(1) = \text{constant}$$

$$T(n) = T(1)n + cn \log_2 n$$

$$T(n) = n \log_2 n$$

MERGE SORT – TIME COMPLEXITY

- Time Complexity: $O(n \log n)$
- Master Theorem:

If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ for constants $a > 0$, $b > 1$, $d > 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

for merge sort: $a=2$ $b=2$ $d=1$

$$T(n) = O(n \log n)$$

TREES

- The differences between tree (hierarchical structure) and linked lists (linear structure)
- The different types of nodes in a tree (parent node, left child node, right child node – although this is just for a binary tree. A parent can have many children!)
- There is just one add method for trees because we cannot add to the “front” or the “back” of a tree
- The property of a complete tree: last level has all nodes to the left
- The property of a balanced tree: the height of left and right sides differ by no more than 1
- Trees are a recursive data structure where any parent can be considered the root of its own sub-tree; thus trees can be built recursively
- Always check for NULL pointers in any data structure – the add function for trees, as an example of this, always check the root to see if it's NULL

BINARY SEARCH TREE

- Binary search tree always has a structure where the left and right nodes are added according to a rule: whether the values they hold are greater than the root, or less than the root
- What's the maximum height of a binary search tree? Why?
- What's the worst case time complexity for searching in a (balanced) binary search tree?
- How is the height of a BST related to its worst case time complexity for searching?
- What is a 'balanced' binary tree?
- How could a binary tree become "unbalanced"?
- What is the worst case time complexity for searching in a balanced binary search tree? What is the worst case time complexity for searching in an unbalanced binary search tree?

HEAPS AND MIN HEAPS

- Heaps

1)order property: for each node in the heap, whether the value in a node is greater than its child node.

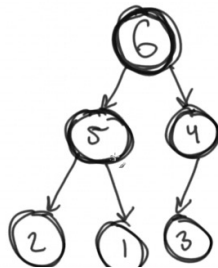
2)shape property: the tree is perfectly balanced and that the elements at the bottom level are pushed as far to the left as possible.

Min Heap: the value of each node is greater or equal to its parent node. The root will be the smallest value node.

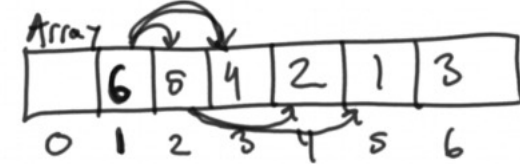
Putting nodes value in array:

parent node index*2 = left child index, parent node index*2+1 = right child index

This is a max heap->



This is a max heap in an array:



PRIORITY QUEUE


- Priority Queue
- Why do we have different big O numbers here? What conditions can impact the big O time complexity of priority queue operations?

add $O(n) = O(\log n)$

access $O(1)$

find $O(n)$

remove $O(n) = O(\log n)$



Binary (Min or Max) Heap

Find-min	Insert	Find	Delete-min
$O(1)$	$O(\log n)$	$O(n)$	$O(\log_2 n)$

GRAPHS

- Concepts:
- vertex, edge
- Degree – in degree & out degree
 - What kinds of graphs have degrees vs in-degrees and out-degrees!
- directed/undirected
- weighted/unweighted
- connected, strongly connected
- path, simple path
- cycle (how to detect a cycle)

GRAPHS: ADJACENCY LIST

- What does an adjacency list store, and how is it structured
- What is the big-O space complexity of an adjacency list?
- What is the big-O time complexity of an adjacency list?

GRAPHS: ADJACENCY MATRIX

- What does an adjacency matrix store, and how is it structured?
- What is the big-O space complexity of an adjacency matrix?
- What is the big-O time complexity of an adjacency matrix?

BREADTH-FIRST SEARCH

- What is the big-O time complexity for searching an entire graph using breadth-first search (BFS)? Does the representation of the graph (adjacency matrix vs list) make a difference?
- Perform BFS on a sample graph

DEPTH-FIRST SEARCH

- What is the big-O time complexity for searching an entire graph using depth- first search (DFS)? Does the representation of the graph make a difference?
- Perform DFS on a sample graph

GREEDY ALGORITHMS

- What is greedy algorithms
 - Is greedy always the best solution
- The Interval Scheduling Problem
 - The approach to solve the interval scheduling problem
 - Start Earliest
 - Smallest Interval First
 - Fewest Interruptions
 - Finish earliest
 - Which one is right? Why are the others wrong?
 - Method to prove greedy algorithm
 - Direct Proof
 - Proof by Contradiction
 - Proof by Induction

GREEDY ALGORITHMS

- What do you compare the greedy algorithm to in the proof by contradiction to show that it always stays ahead?
 - Itself
 - A random algorithm
 - The optimal algorithm
 - A Dynamic algorithm
- Dijkstra's algorithm
 - What is Dijkstra algorithm
 - How Dijkstra's algorithm works
 - Time and space complexity of Dijkstra's algorithm
 - Perform Dijkstra's algorithm on a sample graph (which node is reached first? Etc.)

DYNAMIC PROGRAMMING

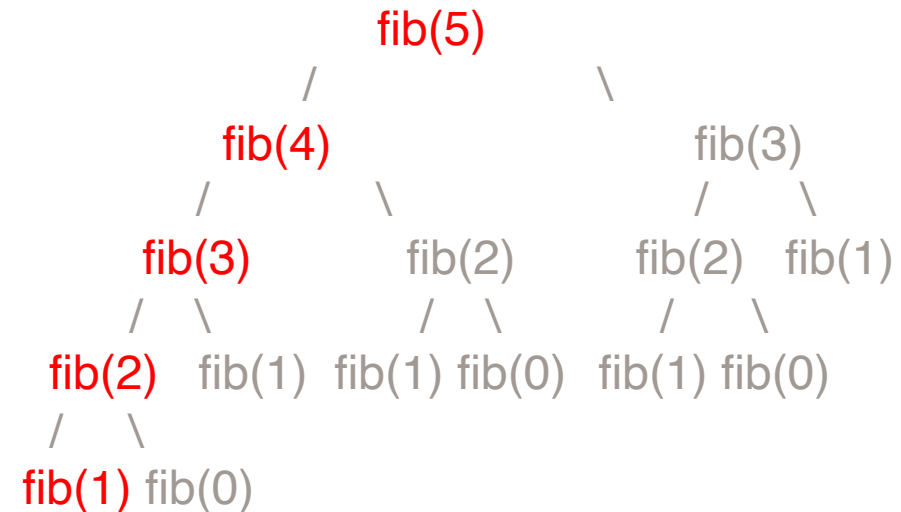
- Definition:
 - Break a big problem into smaller overlapping sub-problems, combined to solve the original one.
- Feature:
 - Memoization: store the results computed and save the time by looking them up instead of recomputing them.
- Examples:
 - Fibonacci
 - Longest common subsequence
 - Knapsack

DYNAMIC PROGRAMMING

- Fibonacci: Caching vs Computation

Table	
fib(0)	0
fib(1)	1
fib(2)	1
fib(3)	2
...	...

DP solution: $O(n)$



Naive solution: $O(2^n)$

DYNAMIC PROGRAMMING

- Longest common subsequence:

Time Complexity: $O(m*n)$

	y	i	k	e
b	0	0	0	0
i	0	1	1	1
k	0	1	2	2
e	0	1	2	3

Q & A

- Questions?
-

THANKS!

- Stay safe, be encouraged, & see you next week!

