Lecture 12
Midterm Exam Review

EECS 281: Data Structures & Algorithms

Time and Location

- When: Wednesday October 28th, 7:00pm (Michigan time, so 7:10) - 8:40pm (90min)
- Where: (by uniqname)

Room	First Uniqname In This Room	Last Uniqname In This Room
CHRYS220	aaleung	danilvnh
DOW1013	danjchoi	haohanx
BBB1670	haolu	jtrate
EECS1500	juelinw	kxrich
CHRYS133	lbarwiko	mibrow
DOW1017	michxie	ntenc
DOW1014	nvogler	richen
DOW1010	richwu	slafeir
DOW2150	sltou	tianmu
DOW2166	tianpeng	wckryska
DOW1006	weihsinc	yergicol
BBB1690	yhpham	yuke

Policies

- Closed book and closed notes
- One "cheat sheet", limited to 8.5"x11", single-sided, hand-written, with your name on it
- No calculators or electronics of any kind
- Engineering Honor Code applies

Don’t forget: Written Portion!

The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2015

MIDTERM EXAM
Written Portion
Wednesday October 28, 2015
7:10AM – 8:40PM (90 minutes)

Name: _____ **Uniqname:** _____

Student ID: _____

Uniqname of person to your left: _____

Uniqname of person to your right: _____

Honor Pledge:
“I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code.”

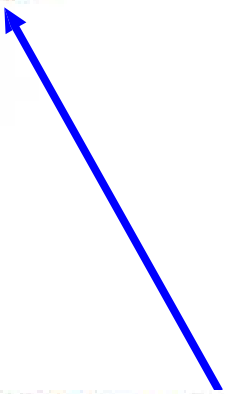
Signature: _____

Fill this out LEGIBLY, and sign the Honor Pledge

Don’t forget: Multiple-Choice!

The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2015

MIDTERM EXAM
Multiple-Choice Portion, KEY 1
Wednesday October 28, 2015
7:10AM – 8:40PM (90 minutes)



- **Record your NAME, STUDENT ID number and exam KEY number on the Scantron form. There will be a penalty for incorrectly filling out the form.**
- There is no need to record your section number, all Scantrons will go into one pile.

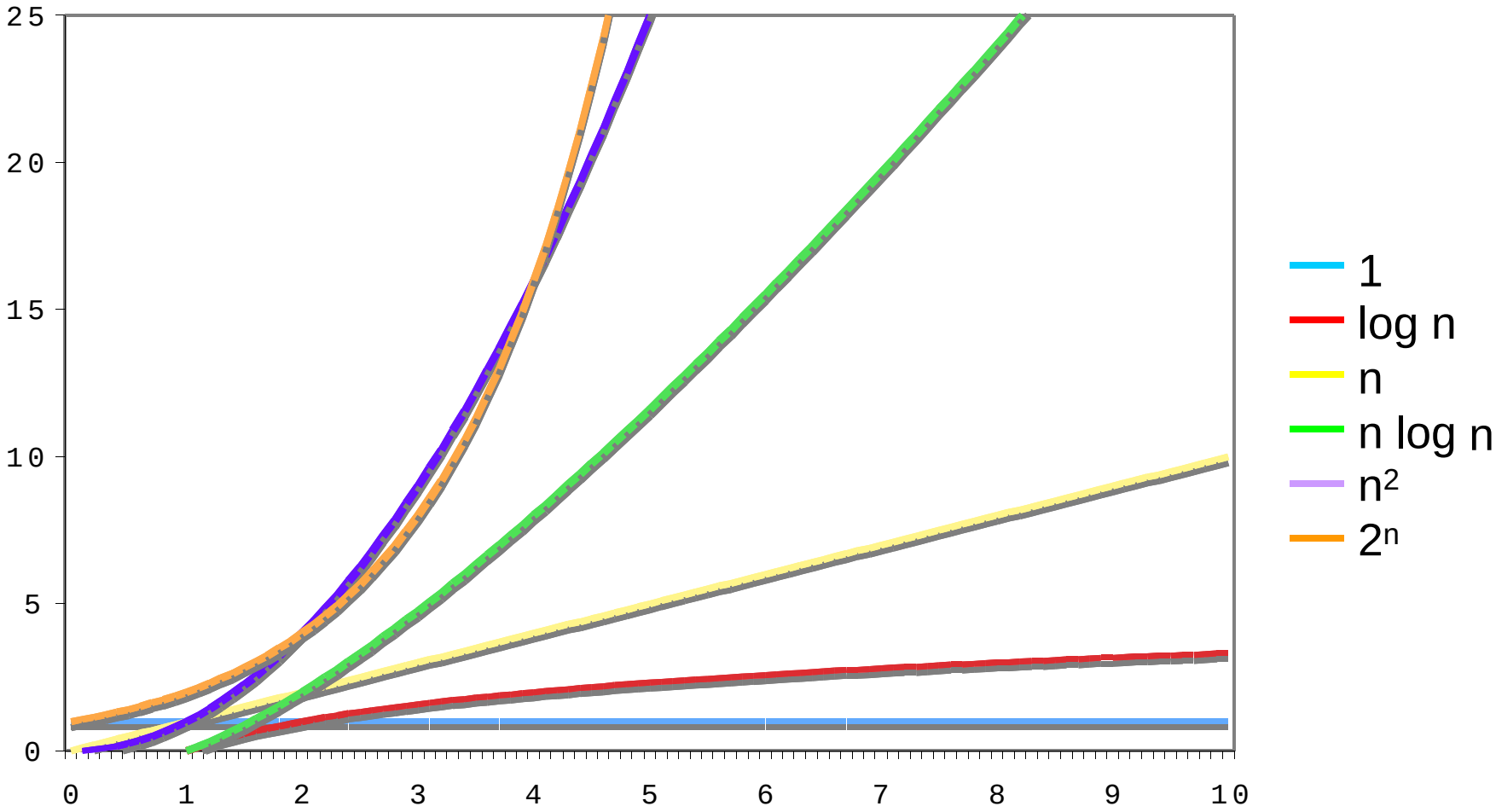
Study Materials

- Practice exam posted on Ctools
- Lecture slides and recordings
- In-class exercises
- Discussion materials
- Homeworks
- Projects
- Study group

Topics

- Everything we have covered so far, especially:
- Complexity analysis, including recurrences
- Contiguous (array) and linked containers
- Stacks, queues and priority queues
- Sorting and heaps

Complexity Analysis



What is the complexity? O(...)

```
int* bsearch (int* lo, int* hi, int val) {
  while (hi >= lo) {
    int* mid = lo + (hi - lo) / 2;
      if (*mid < val) lo = mid + 1;
      else if (*mid > val) hi = mid - 1;
      else return mid;
  }
  return nullptr;
}
```

```
void f(int *out, const int *in,
       int size) {
  for (int i = 0; i < size; ++i) {
    out[i] = 1;
    for (int j = 0; j < size; ++j) {
      if (i == j) continue;
      out[i] *= in[j];
    }
  }
}
```

What is the complexity? O(...)

- Write the recurrence relation
- Solve

```
void mergesort(Item a[], int left, int right) {
   if (right <= left) return;
   int mid = (right+left)/2;
   mergesort(a, left, mid);
   mergesort(a, mid+1, right);
   merge(a, left, mid, right);
}
```

Containers

- What is the best container if it will be used primarily to locate objects within it using binary search?
- What is the best container if new objects will often be added immediately before specific existing objects?
- What is the best container if you must store a small number of very large objects. Memory is scarce and the most important consideration is to store as many of these objects as possible in the available space?

- Options: singly-linked list, doubly-linked list, vector
- Also: WHY?

Containers

- What is the worst container if you must store a large number of one byte items and memory is the scarcest resource?
- What is the worst container if you will frequently insert new items anywhere within the structure?
- What is the worst container if you will frequently insert new items at the beginning of the structure?

- Options: singly-linked list, doubly-linked list, vector
- Also: WHY?

Stacks and queues

- Implement a queue using two stacks. Write the dequeue() function.

```
class MyQueue {
private:
  stack<int> s1, s2;
public:
  void enqueue(int num) {
    s1.push(num);
  }
  void dequeue();
  int front();
};
```

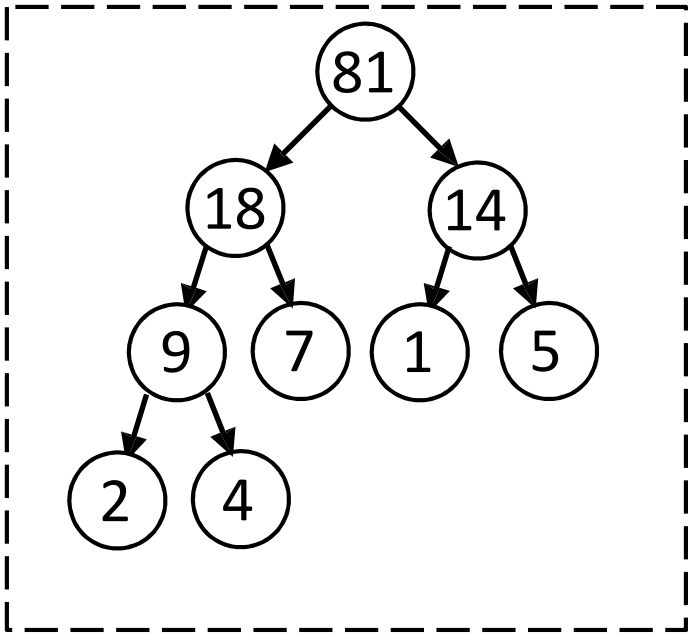
Sorting

Which sort is best?

- Array that is "almost" already sorted
- Very small array
- Medium size array
- Large array (about as big as main memory)
- Very large tape drive

You’re using a quicksort on a very large input, and it’s taking longer than normal. What happened?

Heaps



- Draw the underlying array for this heap
- Insert the value 47
  - Use fixUp()
- Draw the resulting tree and array

Heaps

- What is the complexity?

	Unordered Array	Ordered (Sorted) Array	Heap
create(range)			
push()			
top()			
pop()			