p -> thing
inside p is a number which is the address for a thing and some sort of marker for what that thing should be. (there actually isn't a marker, the compiler just compiles code ++s etc correctly based on size of data structure)

C is a call by value language
I make a copy of the parameter so you can't do something stupid like increment(4) and then 4 is now 5. FORTRAN is weird

```
x = 3;
void incr(int *z){
       *z = *z +1;
}
incr(&x);
// x now equals 4
```

```
char * x;
void ch(char **p){
       *p = x;
}
```

p is a pointer to a pointer to a character, if you want to change what the pointer it points to is pointing at.
functions create their own stack versions of inputs but you can change what they're pointing at because the function variable points at the same place.

look up registers
0(fp)
register fp is a pointer in the processor.
you reference things in relation to the fp, variables on the stack are just aliases for fp(-some number) the stack gets built behind fp.
registers are not in memory, they are special things that the computer knows about.
There's a special register that points to the next command the computer needs to do.

You can do integer things to pointers. And pointer things to integers. But you shouldn't. Modern compilers won't let you. But most old ones will let it fly.

```
for i = 1 to n-1 {
        swap a[i] with the min(a,i,n)
}
```

if you swap with self you just did an idempotent operation;

BUBBLE SORT again, why are we doing this again? Why!!!!!!

does anyone still not know how to do a goddamn bubble sort this is an honors class for godsake.
seriously we were told this was going to be a hard class. I understand being nice, but I signed up for getting my ass kicked and this lecture is boring my ass off. uuuuuggggghhhhh. that's not true the pointer section was good conformation and and registers was awesome.

Heap: A full binary tree, where the value of any parent is greater than that of its children

```
heapsort(a,first,last)
buildheap(a,first,last)
i = last to first;
swap(a[i] with a[first])
fixheap(a,first,i-1)
```

Wednesday 10/28/15

Queues, stacks, and sorts.
1 thing from each week we've done. write a little code, not a lot of code, won't kill you if you forgot a ; write clearly write concisely.

Event Based programming, the core data structure is going to be a priority queue. Heaps are priority queue. Discrete event simulations, also continuous event sims

make 101 prof show lognlogn data structure

```
heap sort{
        while heap not empty{
                grab top element
                put rightmost leaf at top
                fix rest of heap.
        }
}

fixheap(a,first,last)
        found = false;
        father = first;
        greater = max_child(a,father,last);
        while(father ≤ last/2 and !found){
                if(a(Father) < a(greater)){
```

```
                swap(a[father] a[greater])
                father = greater;
            }
            else{
                found = true;
            }
        }
}
```

Heap sort is always O(nlogn) even if you see the adversary

Quick sort sans arrays let it begin
I present to you the Stack. Last in first out
You can push something on the stack
You can pop something off a stack
You can check if the stack is empty

```
push(x){
    //array s[p++]=x
    //linked list *p++=x
}
pop(){
    //array return s[p--]
    //linked list x=*p--
}
```

What makes sense for a heap
pop
insert
empty?
Might want makeheap, you can also call insert on every elem of an unordered list.

Stacks have a temporal order. ordered by time.
Heaps do not have a temporal order.

Queue (first in first out)

head, tail,
enqueue (add to trail of queue)
dequeue (pull from head of queue)
empty?
(sometimes concat)
(sometimes full?)
(sometimes peak which allows you to look at head without taking it off)

Let's make a queue out of arrays

```
enqueue
    q[t++%N]=x

dequeue
    y = q[t++%N]
```

%N makes it circular so it can go around however big we want the queue to be.

```
empty
    h == t
full
    (t++%N) == h
```

This is called a circular buffer, encouraged to write this code.
LIFO FIFO

heaps are a priority queue still a queue even though it's not temporally ordered.

```
Qsort(A)
    left = []
    mid = []
    right = []
    pivot = peek(A)
    foreach x in A (while A not empty dequeue)
        if x == pivot
            mid = concat(mid, x)
        else if x ≤ pivot
            left = concat(left, x)
        else
            right = concat(right, x)
concat(Qsort(left), mid, Qsort(right))
```

best case nlogn, worst case is n^2, average is going to tend towards nlogn.

the adversary for quicksort is a sorted array.

If you use a heap instead of a queue it automatically spits out a sorted list aka the adversary

READ THE WHOLE C BOOK!!!!!!!

Hardware:
know the 2 kinds of transistors
give me a NAND Gate, and a not gate can you make an and gate
You should be able to add two binary numbers
You should be able to 2s compliment 2 binary numbers
You should be able to multiply 2 binary numbers
Base conversions. Do it by long division.
Memory is approx how much is memory faster than harddrive 100k a million
How many megas in a kilobyte etc whats bigger than what.
You will not need a calculator