

# lecture 3

- memory addresses
  - reference variables
  - arrays

Preliminaries

- insertion sort (arrays)
- Linear Data Structures

## Primitive Types (in Java)

	# bits	# bytes
boolean	1	1
byte	8	1
char	16	2
short	16	4
int	32	8
long	64	8
float	32	4
double	64	8

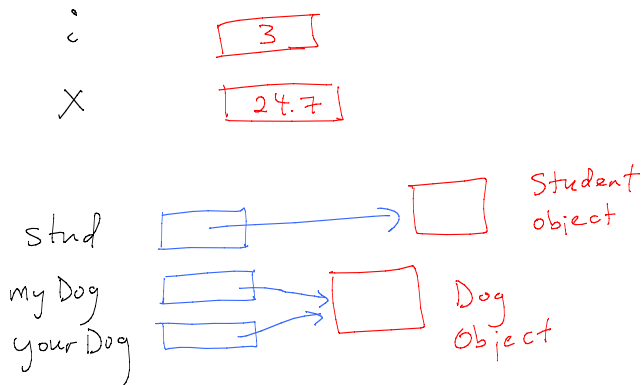
## Java Classes

- String } predefined
- : }
- Student } defined by you
- Dog }
- : }

## Variable Names

primitive type		
	byte	b;
	int	i;
	char	c;
	float	f;
	double	x;
reference type	Student	stud;
	Dog	myDog;

How to think about variable names ?



$2^{32} - 1$  bytes

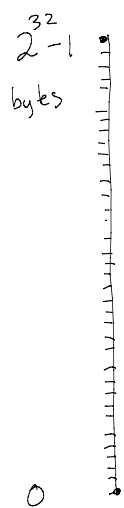
A variable name stands for a number, namely a memory address where data is stored.

(The compiler replaces the variable name by that address.)

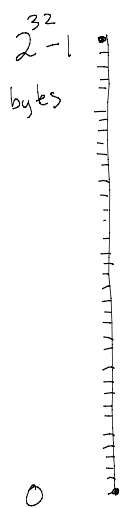
Memory addresses are 32-bit numbers, i.e.  $0, 1, \dots, 2^{32} - 1$

(More recent computers use 64 bit addresses.)

This slide was added after the lecture....



$2^{10}$  bytes = 1 KB  $\approx 10^3$  bytes  
 $2^{20}$  bytes = 1 MB  $\approx 10^6$  bytes  
 $2^{30}$  bytes = 1 GB  $\approx 10^9$  bytes  
 $2^{32}$  bytes = 4 GB



What are these 32 bit memory addresses?  
 Are they locations in "RAM" or on the "hard disk"?  
 No! Rather they are addresses of bytes in the memory of an abstract machine. (For a Java program, this is called the "Java Virtual Machine".)

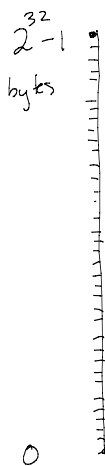
### Analogy

My postal address is:  
 3480 University St, room 318  
 H3A 2A7, Canada

This is not a physical address. Rather, the physical address is (latitude, longitude, altitude).

(Virtual) Memory Addresses

Physical addresses



This mapping is done by the computer (operating system). The Java programmer does not have to think about it.



```
Student s = new Student();
```

constructs an object somewhere in memory (size of object depends on class)



memory location `s` holds the address of an object

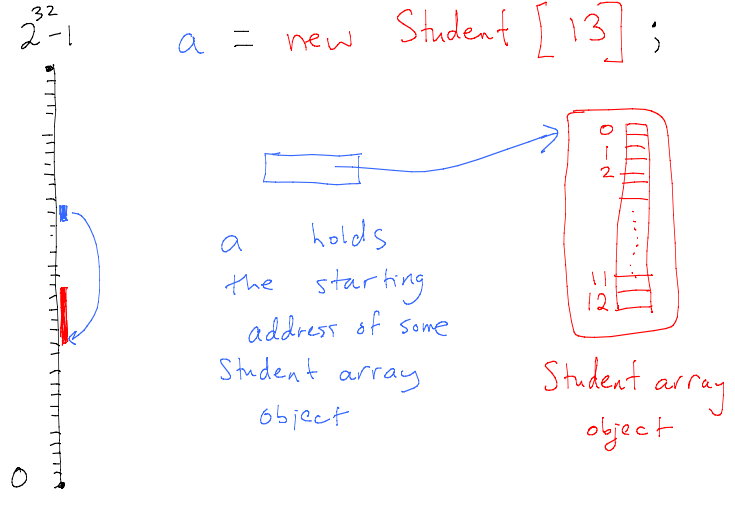
### Arrays

```
Student[] a = new Student[13];
```

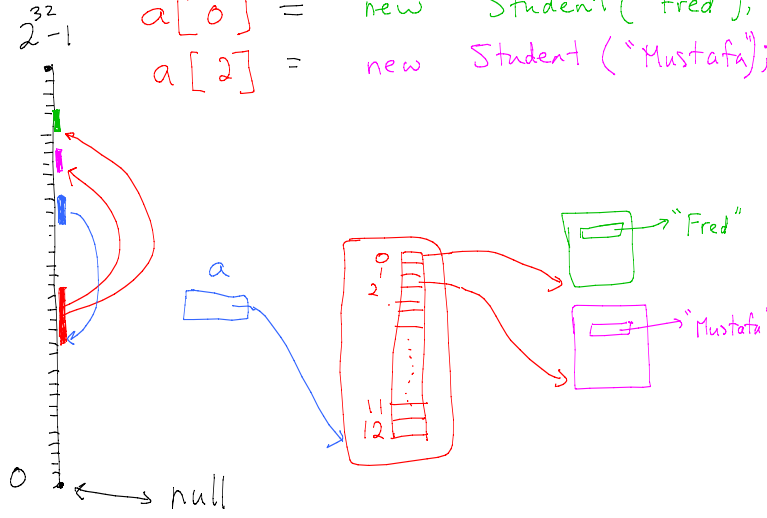
reference variable (the address of a Student array object)

constructs an object somewhere in memory (an array of 13 students)

```
Student [] a ;
a = new Student [ 13 ] ;
```

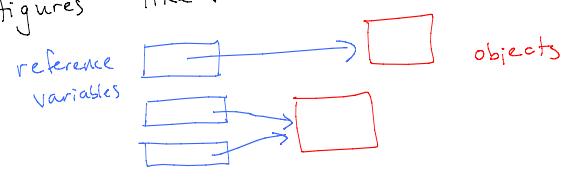


```
a[0] = new Student ("Fred");
a[2] = new Student ("Mustafa");
```



What's new here?

In COMP 202, you may have seen figures like:



But there was no notion of address.

Now, you should start to think about addresses as 32 bit numbers. (4 bytes)

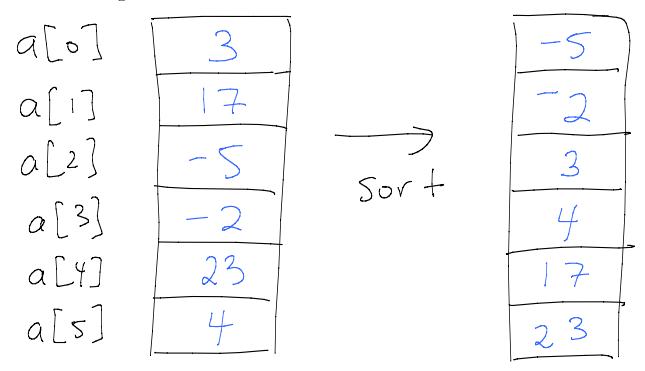
### Linear Data Structures

- arrays
- linked lists (next class)
- Java: ArrayList & Linked List
- stack
- queue

### Arrays

You have learned about arrays in COMP 202. (or equiv). Here I will give an example of how to use an array to solve a common problem.

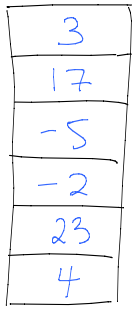
Sorting an array eg. int[]



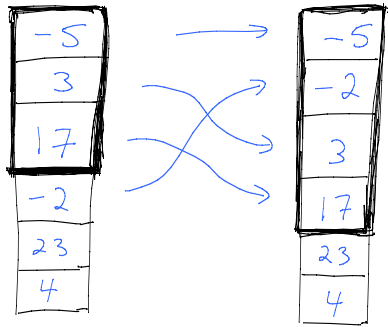
What is a good algorithm?

Sort elements  
 $0, \dots, k-1$

insert element  $k$



eg.  
 $k=3$



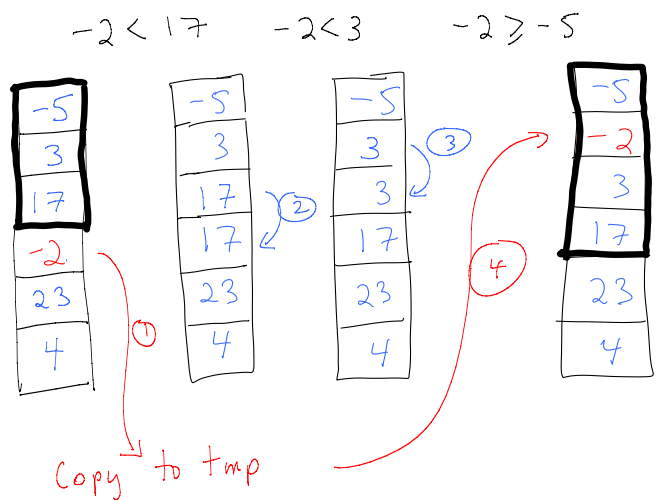
## "Insertion Sort"

for  $k = 1$  to  $N-1$  {  
 insert element  $k$  into  
 (sorted) elements  $0, \dots, k-1$   
 }

time

	$k=0$	1	2	3	4	5
$a[0]$	3	3	-5	-5	-5	-5
$a[1]$	17	17	3	-2	-2	-2
$a[2]$	-5	-5	17	3	3	3
$a[3]$	-2	-2	-2	17	17	4
$a[4]$	23	23	23	23	23	17
$a[5]$	4	4	4	4	4	23

How to insert at step  $k$ ?

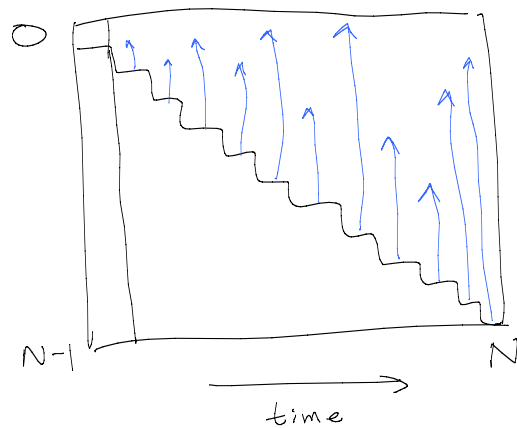


```

for  $k = 1$  to  $N-1$  {
  tmp ←  $a[k]$ 
   $i \leftarrow k$ 
  while  $i > 0$  &  $a[i-1] > tmp$  {
     $a[i] \leftarrow a[i-1]$ 
     $i \leftarrow i-1$ 
  }
   $a[i] \leftarrow tmp$ 
}

```

How long does the insertion sort algorithm take? (See lecture notes.)



Worst case:  
 $C_1 N + C_2 N^2$   
 Best case:  
 $C_1 N$