Memory Complexity

Let’s talk about Big O for memory

Objectives

- Understanding the basis about expressing space complexity of our solutions

What do we measure?

- **Space** – How much space does my solution need?
- As with performance, this space is described in terms of the input.
- When describing, be clear if you talk about “additional space”

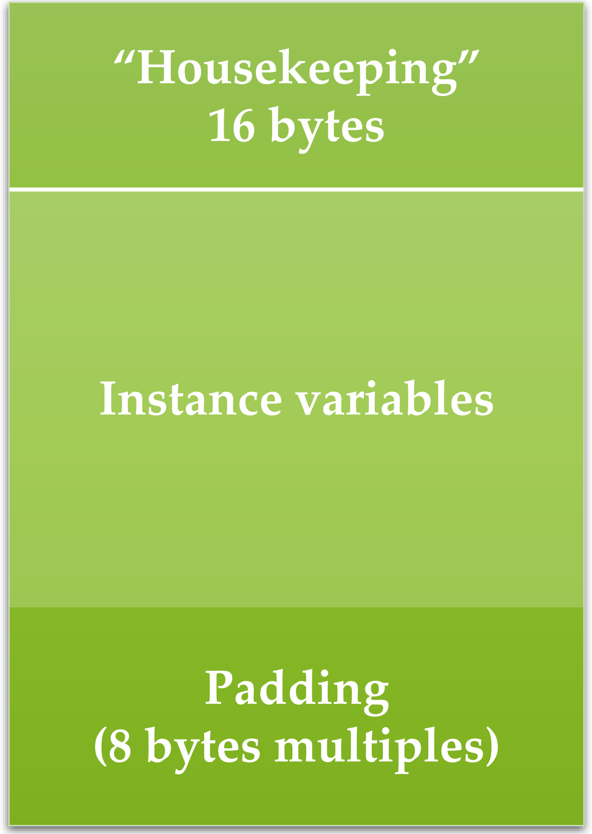
Memory in Java

- In Java, we mostly work with 64 bits JVMs. This is, we have 8 bytes pointers.
- So, how much space do primitives use?

Type	Size (bytes)
boolean	1
byte	1
short/char	2
int	4
float	4
long	8
double	8

Memory in Java

- What about objects?



“Housekeeping”
16 bytes

Instance variables

Padding
(8 bytes multiples)

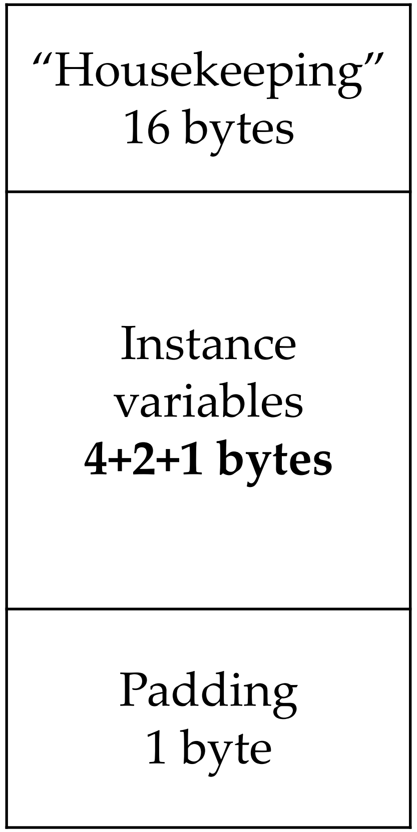
Memory in Java

- For example…

```
class TestClass{

    private int age = 20;
    private char c = 'a';
    private boolean b = true;

}
```



“Housekeeping”
16 bytes

Instance
variables
4+2+1 bytes

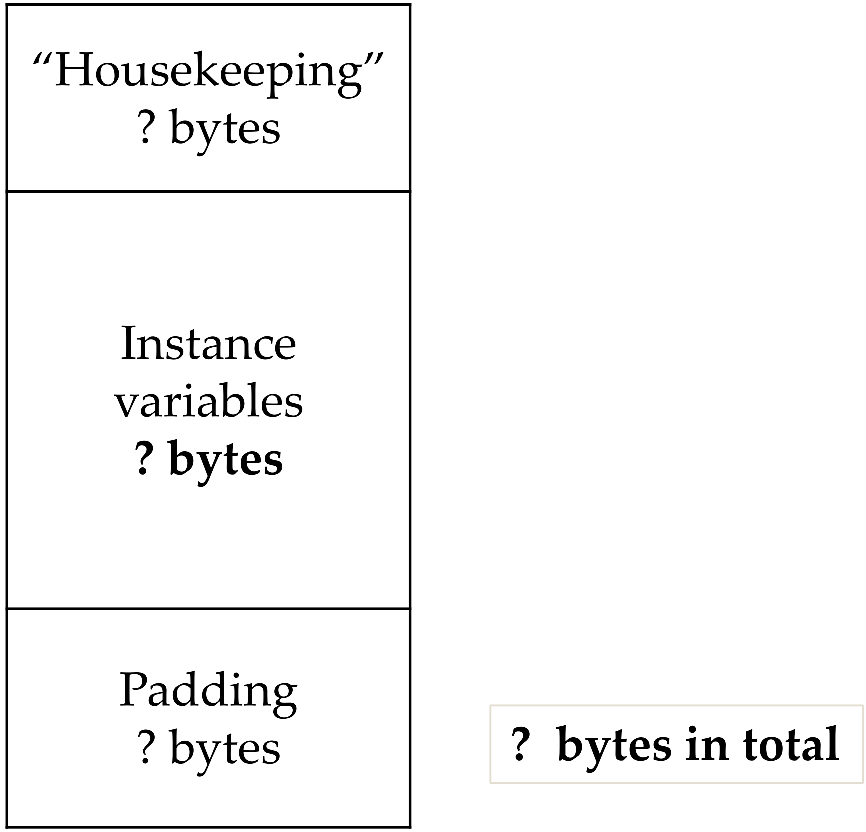
Padding
1 byte

24 bytes in total

Memory in Java

- For example…

```
public final class String{
    private char[] value;
    private int offset;
    private int count;
    private int hash;
}
```



"Housekeeping"
? bytes

Instance
variables
? bytes

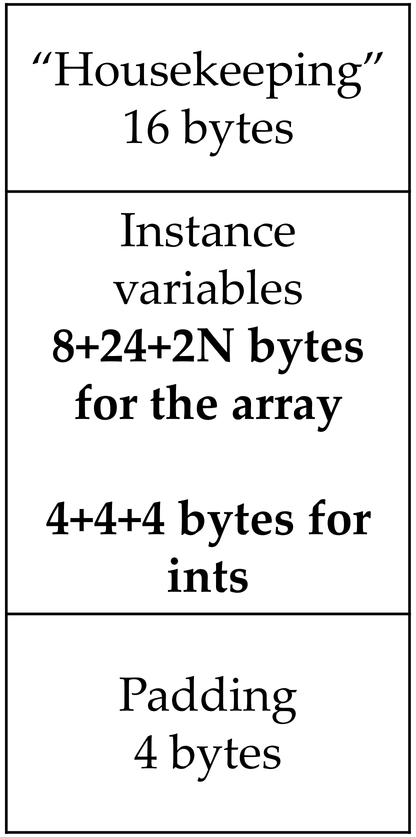
Padding
? bytes

? bytes in total

Memory in Java

- For example…

```
public final class String{
    private char[] value;
    private int offset;
    private int count;
    private int hash;
}
```



“Housekeeping”
16 bytes

Instance variables
8+24+2N bytes for the array

4+4+4 bytes for ints

Padding
4 bytes

References are **8 bytes**

Arrays have an overhead of **24 bytes**
(HK=**16**, length=**4**, padding=**4**)
+ (type of array * size (N)),
in this case 2N

2N + 64 bytes in total

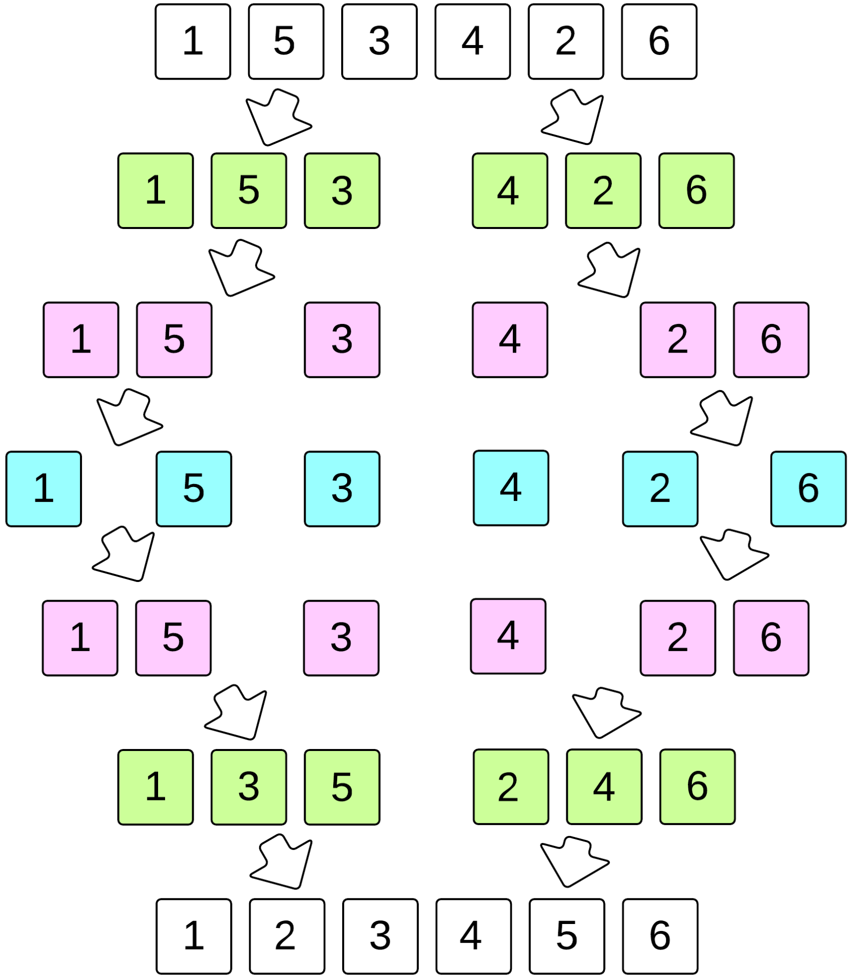
Big O

- So, as in the case of performance, we drop constants since they become irrelevant as time passes.

- So **2N + 64** is still **O(N)**

- This means that if we need an **additional** data structure to hold our entire input, the space complexity would be **O(N)**

Merge sort case



1 5 3 4 2 6

1 5 3 4 2 6

1 5 3 4 2 6

1 5 3 4 2 6

1 5 3 4 2 6

1 3 5 2 4 6

1 2 3 4 5 6

Any questions?