Memory allocator algorithms
four points to cover
1) How to track free blocks
2) How to choose free blocks when allocating
3) What to do w/ extra space after allocation
4) What to do w/ blocks when they are freed

```c
int** a = malloc (100);
```
For algorithms
cover
free blocks
use free blocks when allocating
with extra space after allocation
with blocks when they are freed

alloc (100)

Struct block
    void * start;
    int size;
    3

block
    Opt. Padding
    Payload
    size 004

encoding allocation in
size field

8-byte aligned

10111000
0010
allocated bit
\( \text{size divisible by 8 (size \& 1 == 0)} \)
\( \text{L} \) \( \text{unallocated} \)
\( \text{L} \) \( \text{allocated} \)

the "real" size of an allocated block
size \& (-1 << 3)

Struct block*
b;

b->size
b = 104
b = 73 (real size) 72
2) How to choose a block?
   1) First-fit \( \Rightarrow \) first block that's big enough
   2) Best-fit \( \Rightarrow \) block big enough
   3) Next-fit \( \Rightarrow \) "next" block that's big enough

   + Fast
   - Slow
   - Fragmentation

   + Least frag.
   - Slow
   \( \sim \) Ok fragmentation
   + Fast

3) Extra space after allocation
   - Use the whole block \( \Rightarrow \) internal fragmentation
   - Split block into 2 blocks
q-malloc (48)
free(a)

Can't use whole 16 bytes because need space for header.

What do we do when freeing:
- Coalesce free blocks together.
  - Straight forward if next block is also free.
  - What about prev. block?

Don't coalesce? → False fragmentation
- Traverse the whole list
Putting it all together - assignment 5
Mem_alloc 3 you implement
Mem_free 3

-7 variation of implicit free list
-7 Do NOT change headers/structs
-7 test-driven development
-7 best-fit algorithm
-7 linked-lists

-7 I did it ~100 LOC
-7 lots of tricky pointer arithmetic
Tracking free blocks

hash table
binary tree
linked list

1. Where to put these?

```
struct free_node {
    struct free_node *next;
    void *free_block;
    int size;
}
```

Explicit free lists

Implicit free lists