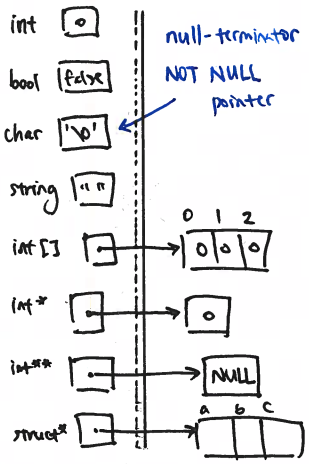
Pointers & Structs




```
struct a_header {
    string name;
    int    age;  };
```
①

typedef struct ... ---

→ concrete type definition

④ client type defn.
typedef a* a_t;

```
typedef struct stack_header stack;
typedef stack* stack_t;

stack* S = alloc(stack);

stack_t S = alloc(stack_t);   // doesn't work.
```

representation invariant: ②

```
bool is_ssa(ssa* A) {
    if (A == NULL) return false;
    //@ assert A->length == \length(A->data);
    return is_sorted(A->data, 0, A->length); }
```

↳ used in implementation ③ func. implementations.

```
// typedef ___* ssa_t;
ssa_t ssa_new(int size)
//@ requires 0 ≤ size;  //@ requires \result != NULL;
//@ ensures ssa_len(\result) == size;

void ssa_set(ssa_t A, int i, string x)
//@ requires A != NULL;
//@ requires 0 <= i && i < ssa_length(A);
```

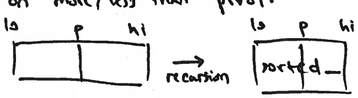
library interface ↳ pseudo typedef ① abstract type name

no //@ ensures != NULL

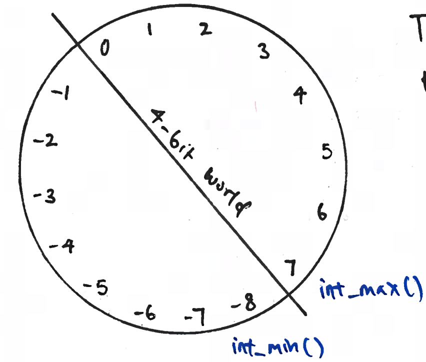
↳ function prototypes. ⑤

* when a pointer is set to be equal to another pointer, the first pointer points to where the second pointer points **.

1. binsearch: find middle, throw away half
2. selectionsort: traverse through elements after current, swap with smallest.
3. quicksort: find pivot, separate/sort based on more/less than pivot.



4. mergesort: halve until 0/1 elements, merge
   ↳ int mid = lo + (hi - lo)/2;



Dec		Bin	Hex
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	-8	1000	8
9	-7	1001	9
10	-6	1010	A
11	-5	1011	B
12	-4	1100	C
13	-3	1101	D
14	-2	1110	E
15	-1	1111	F

$f \in O(g)$: there exists natural number n_0 and a real $c > 0$ st. for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

* simplest & tightest bounds!

Two's complement: every k-bit word corresponds to a # between -2^{k-1} and $2^{k-1}-1$

→ int_max() + 1 == int_min()   1 byte = 8 bits

→ -int_min() == int_min()   int: 32-bit

→ -x == ~x + 1   ith bit: (x >> i) & 1

$x << k = x \cdot 2^k$

$x >> k = \lfloor \frac{x}{2^k} \rfloor$ → sign extension!

$0 \leq k < 32$

$a_n 2^n + \ldots + a_0 2^0$

Decimal → Binary → Hex

/2 repeatedly, reverse order of remainder.   4 bits = 1 Hex

arithmetic operators

- +, -, * : handled using modular arithmetic (also ==, !=)
- >, >=, <, <= : ... two's complement.
  ↳ need to account for overflow
- (x/y)*y + (x%y) = x, rounds towards 0.
  //@ requires y != 0; !(x == int_min() && y == -1);

bitwise operations: &, |, ~, ^ (xor)
↳ ints can encode bit patterns.

	linear search	binary search	selection sort	merge sort	quicksort
best	O(1)	O(1)	O(n²)	O(n log n)	O(n log n)
avg.	O(n)	O(log n)	O(n²)	O(n log n)	O(n log n)
worst	O(n)	O(log n)	O(n²)	O(n log n)	O(n²)
			in-place¹	stable²	in-place

1. constant amount of memory is allocated
2. relative order of duplicate elements doesn't change aft. sorting

"loop runs constant no. of times" "each operation has constant cost" "each run takes O(_)" "loop takes time in O(_) each iteration"

Loop Invariants

1. INIT: code before loop, preconditions, (loop) initialization
2. PRES: LI (prev. iteration), loop guard, code in loop (current iteration)
3. EXIT: LIs, negation of loop guard, code after loop
4. TERM: the expression ___ strictly increases/decreases on each iteration, but cannot go above/below ___ on which the loop guard is false and the loop terminates.

SAFETY: A[i] : 0<=i<n, p != NULL, y!=0

CORRECTNESS: preconditions imply postconditions, function behaves.