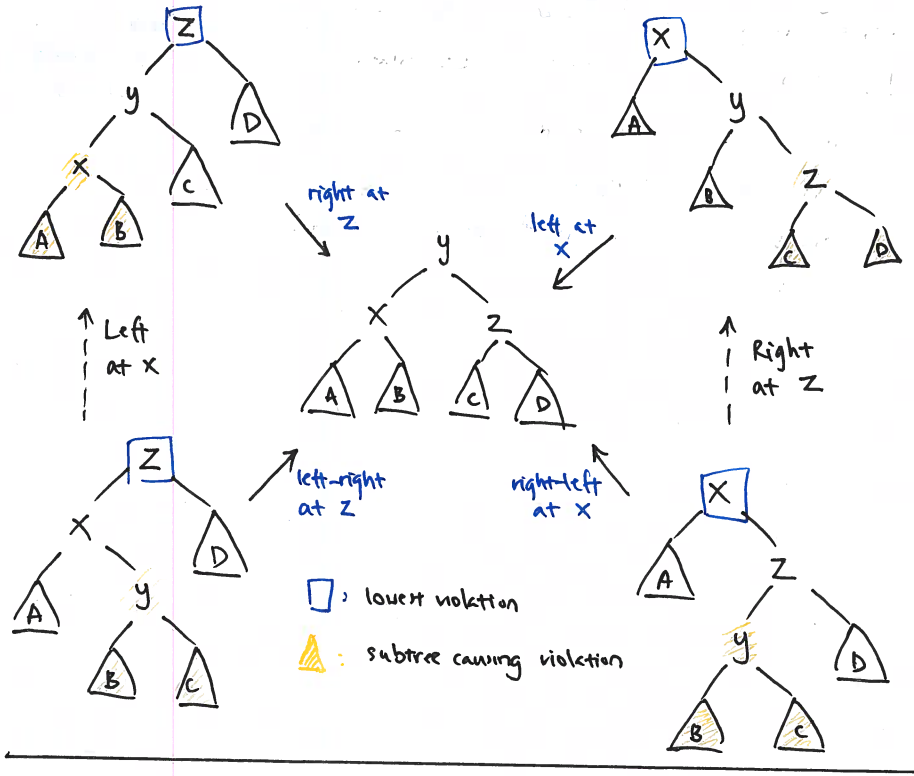
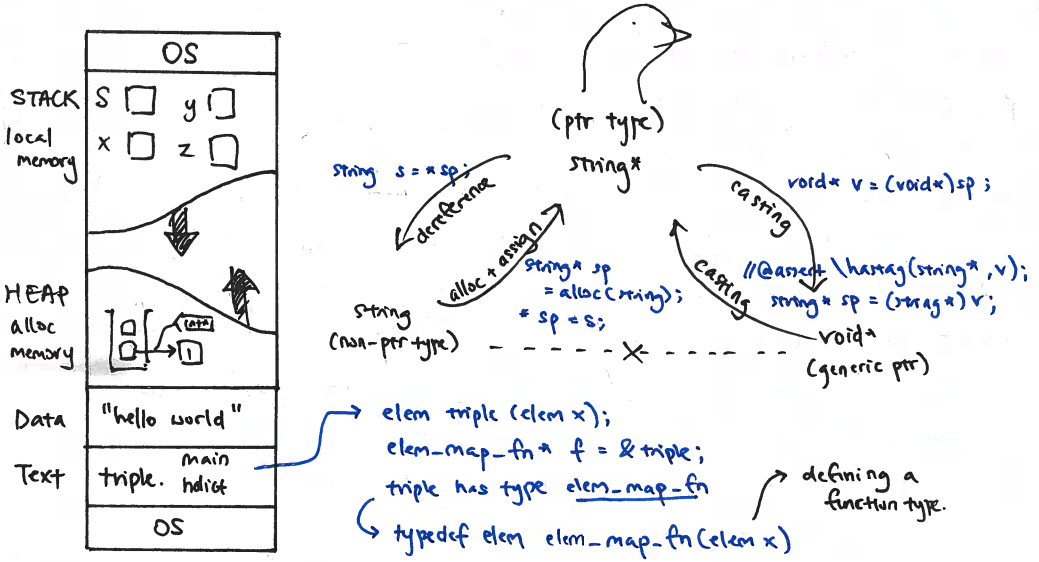




Dec	Bin	Hex	
0	0000	0	
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	
7	0111	7	int_max()
8	1000	8	int_min()
9	1001	9	
10	1010	A	
11	1011	B	
12	1100	C	
13	1101	D	
14	1110	E	
15	1111	F	

Two's complement: every k-bit word corresponds to a # between -2^{k-1} and $2^{k-1}-1$ ↳ >, <, ... need to account for overflow

→ int_max()+1 == int_min()

→ -int_min() == int_min() → -x == ~x+1

$x \ll k = x \cdot 2^k$, $x \gg k = \lfloor \frac{x}{2^k} \rfloor$, $0 \le k < 32$ ↳ sign extension!

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

Decimal ⇄ Binary ⇄ Hex

Decimal → Binary: /2 repeatedly, reverse order of remainder

Binary → Hex: 4 bits = 1 Hex

$(x/y) * y + (x \,\%\, y) = x$, rounds towards 0

//@ requires y != 0, !(x == int_min() && y == -1);

	insearch	binsearch	selectionsort	mergesort	quicksort
best	O(1)	O(1)	$O(n^2)$	O(n log n)	O(n log n)
avg.	O(n)	O(log n)	$O(n^2)$	O(n log n)	O(n log n)
worst	O(n)	O(log n)	$O(n^2)$ ①	O(n log n) ②	$O(n^2)$ ①

① inplace: constant amt. of memory allocated

② stable: relative order of duplicate elements doesn't change after sorting

Half-Generic Dict → 1 type in 1 file

typedef __* entry → client must split code into 2 files

typedef __ key

key entry_key(entry e)

int key_hash(key k)

bool key_equiv(key k1, key k2)

↳ client interface ↳ up to client to define.

Generic Dictionaries

typedef void* entry

typedef void* key

typedef key entry_key_fn(entry e)

typedef int key_hash_fn(key k)

typedef bool key_equiv_fn(key k1, key k2)

① concrete type definition

struct a_header {...}

typedef struct a_header a;

② representation invariant

bool is_ssa(ssa* A) {...}

↳ used in implementation

③ implementation of interface functions → Library Implementation

after client interface & before library interface.

	unsorted array	array sorted by key	linked list	hash table	BST	AVL	heaps
lookup	O(n)	O(log n)	O(n)	O(1) average	O(n)	O(log n)	O(1)
insert	O(1) amortized	O(n)	O(1)	O(1) average + amortized	O(n)	O(log n)	O(log n)
find_min	O(n)	O(1)	O(n)	O(n)	O(n)	O(log n)	O(1)

AVL: ordering invariant ($L < x < R$) + height invariant ($|h_L - h_R| \le 1$)

dictionary with n entries, table with capacity m → load factor is n/m; look-up: worst case O(n/m)

collisions unavoidable → if n > m, pigeonhole principle → certainty; if n > 1, birthday paradox → probabilistic result

INIT: code before loop, precondtions, (loop) initialization

PRES: LI (prev. iteration), Loop guard, code in loop (current iteration)

EXIT: LIs, negation of loop guard, code after loop

TERM: the expression ___ strictly increases / decreases on each iteration, but cannot go above / below ___ on which the LG is false & loop terminates.

SAFETY: A[i]. $0 \le i < n$, p != NULL, y != 0

CORRECTNESS: preconditions imply post conditions, function behaves as expected.

→ loop run constant # of times / each operation has constant cost / each run takes O(...) / loop takes time in O(...) each iteration

→ 1 token to add element, 1 token to allocate this position when adding new node.

CO/CI: #use <util>, #use <conio>, header files, macro definitions, conditional compilation

C: #include <stdio.h>, #include "lib/xalloc.h", #define, #ifdef DEBUG ... #endif

↳ pointer arithmetic: A[i] ≡ *(A+i), p->next ≡ (*p).next

Stack-allocated arrays: int E[8], int F[] = {2,4,6} ← s-a structs: access fields with dot notation

	writables	allocation	dealloc
DATA	NO	auto	N/A
Stack	YES	auto	auto (fn returns)
Heap	YES	manual (alloc)	manual (free)

char* s1 = "1" struct point q = {.x = 15}

char* s2 = xmalloc(strlen(s1)+1);
strcpy(s2, s1); free(s2);

char s3[] = "world"
char s4 [] = {'s', 'k', 'y', '\0'}

sizeof(char) = 1 byte = 8 bits

(new_t) exp → of old_t

① char x = -3 (0xFD)
int y = (int x)
is -3 (0xFFFF FFFD)

② int → char
0xF...F0F → 0x0F

new_t can represent value of exp — YES → value of exp preserved ①

↓

new_t signed — YES → I.D, often discard most signif. bits. ②

↓

new_t larger than old_t — YES → bits sign-extended ③

NO → ④ least significant bits retained

enum seasons {A, B, C};
switch (A) {
  case A:
    break;
  default:
    printf("4");
}

③ char → uint
0xFD → 0xF...FD

④ char → uchar
0xFD → 0xFD

	AM (dense / matrix)	AL (sparse / space)
space	$O(v^2)$	$O(v+e)$
hasedge	$O(1)$	$O(\min(v,e))$
addedge	$O(1)$	$O(1)$
getnbhr	$O(1)$	$O(1)$
iterate over nbhr	$O(\min(v,e))$	$O(\min(v,e))$

DFS: stack, BFS: queue, A*: priority Q.

implement as a heap* (type of bintree)

implemented as array

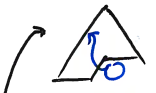
→ smallest number for minheaps.

* also any priority queue with integer priorities.

BST: ordering invariant

AVL: height + ordering

Heap: shape + ordering


≤ 1 violation


≤ 2 violations

___ : maintained.

→ swap with child w/ highest priority (smallest for min-heap)

for node i: left child is 2i, right child 2i+1, parent i/2

! unsigned nums: don't compare with 0!

library fn: ensures is_stack(S)! also if add_elem, size++!