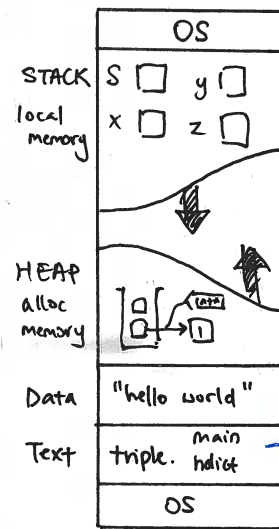


Dec	Bin	Hex
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	-8	1000
9	-7	1001
10	-6	1010
11	-5	1011
12	-4	1100
13	-3	1101
14	-2	1110
15	-1	1111

Two's complement: every k-bit word corresponds to a # between -2^{k-1} and $2^{k-1}-1$.
 $\rightarrow \text{int_max}() + 1 == \text{int_min}()$
 $\rightarrow -\text{int_min}() == \text{int_min}()$
 $\rightarrow -x == \sim x + 1$
 $x \ll k = x \cdot 2^k, \quad x \gg k = \lfloor \frac{x}{2^k} \rfloor, \quad 0 \leq k < 32$
 $a_0 2^0 + \dots + a_{n-1} 2^{n-1}$
 Decimal \leftrightarrow Binary \leftrightarrow Hex
 /2 repeatedly, reverse order of remainder
 4 bits = 1 Hex
 $(x/y) * y + (x \% y) = x$, rounds towards 0
 $\text{//@ requires } y \neq 0, !(x == \text{int_min}() \&\& y == -1);$
 (search) $O(1)$ $O(1)$ $O(n^2)$ $O(n \log n)$ $O(n \log n)$
 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 $O(n)$ $O(\log n)$ $O(n^2)$ $O(n \log n)$ $O(n^2)$

- ① inplace: constant amt. of memory allocated
- ② stable: relative order of duplicate elements doesn't change after sorting



OS
 STACK local memory
 HEAP alloc memory
 Data "hello world"
 Text triple, main, hdist, OS

string s = *sp;
 string* sp = alloc(string);
 *sp = s;
 void* v = (void*)sp;
 // @assert !hasobj(string*, v);
 string* sp = (string*)v;
 *sp = s;
 void* (generic ptr)

elem triple (elem x);
 elem-map-fn* f = &triple;
 triple has type elem-map-fn
 typedef elem elem-map-fn(elem x)

defining a function type.

Half-Generic Dict \rightarrow 1 type in 1 file
 typedef __* entry \rightarrow 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)

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
- after client interface & before library interface.
- Library Implementation

	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 ($|h_L - h_R| \leq 1$)

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

collisions unavoidable \rightarrow if $n > m$, pigeonhole principle \rightarrow certainty
 if $n > 1$, birthday paradox. \rightarrow probabilistic result

INIT: code before loop, preconditions, (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: $AC[i]$, $0 \leq i \leq n$, $p \neq \text{NULL}$, $y \neq 0$

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

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

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

CO/C: #use <util>, #use <conio>, header files (macro definitions conditional compilation) →
C: #include <stdio.h>, #include "lib/xalloc.h", #define, #if def DEBUG ... #endif

→ pointer arithmetic: $AC[i] \equiv *(A+i)$, $p \rightarrow \text{next} \equiv (*p).next$

Stack-allocated arrays: $\text{int } E[8]$, $\text{int } F[] = \{2, 4, 6\}$ ← 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)

→ $\text{char* } s1 = "1"$ struct point $q = \{x = 15\}$
 $\text{char* } s2 = \text{xmalloc}(\text{strlen}(s1) + 1);$
 $\text{strcpy}(s2, s1); \text{free}(s2);$
 $\text{char } s3[] = "world"$
 $\text{char } s4[] = \{'s', 'k', 'y', '\backslash 0'\}$

$\text{sizeof}(\text{char}) = 1 \text{ byte} = 8 \text{ bits}$

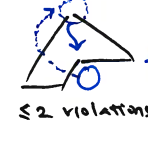
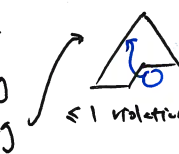
(new-t) exp → of old-t
 ↓
 ② int → char
 $0xF...F0F \rightarrow 0x0F$
 new-t can represent value of exp YES → value of exp preserved ①
 ↓
 new-t signed YES → I.D., often discard most signif. bits. ②
 ↓
 new-t larger than old-t YES → bits sign-extended ③
 NO ④ → least significant bits retained

enum seasons {A, B, C}; ③ char → uint
 $0xFD \rightarrow 0xF...FD$
 switch (A) {
 case A:
 break;
 default:
 printf("");
 }
 ④ char → uchar
 $0xFD \rightarrow 0xFD$

BSI: ordering invariant

AVL: height + ordering

Heap: shape + ordering



— = maintained.
 swap with child to highest priority (smallest for min-heap)

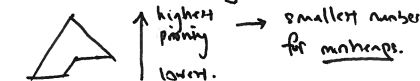
for node i : left child is $2i$, right child $2i+1$, parent $i/2$

! unsigned nums: don't compare with 0!

library fn: ensures is-stack(s)! also if add-elm, size++!

	AM (hasedge) → dense	AL → sparse (space)
space	$O(v^2)$	$O(v+e)$
hasedge	$O(1)$	$O(\min(v, e))$
addege	$O(1)$	$O(1)$
gethhsbr	$O(1)$	$O(1)$
iterate over hhsbr	$O(\min(v, e))$	$O(\min(v, e))$

DFS: stack, BFS: queue, A*: priority Q.
 implement as a heap* (type of bintree)
 implemented as array



* also any priority queue with Integer priorities.