CS106B FINAL REFERENCE SHEET

For-each loop iteration over collection (not `Stack`,`Queue`,`PriorityQueue`): `for (type name : collection) { ... }`

** All Big-Oh runtimes listed are average-case; some methods perform differently under various cases.*

Vector<T>

v.**add**(*val*) or *v* **+=** *val*	append value to end of vector	O(1) *
v.**clear**()	remove all elements	O(1)
v.**get**(*i*) or *v*[*i*]	return value at given index	O(1)
v.**insert**(*i*, *val*)	insert value at given index, shifting subsequent values right	O(N)
v.**isEmpty**()	return **true** if no elements	O(1)
v.**remove**(*i*)	remove value at given index, shifting subsequent values left	O(N)
v.**set**(*i*, *val*) or *v*[*i*] = *val*	replace value at given index	O(1)
v.**size**()	return count of elements	O(1)
v.**subList**(*start*, *length*)	create vector copy containing subrange of elements	O(N)

Grid<T>

g.**get**(*row*, *col*) or *g*[*row*][*col*] or *g*[*location*]	return value at given row/column location	O(1)
g.**inBounds**(*row*, *col*) or *g*.**inBounds**(*location*)	return **true** if given row/column location is within grid bounds	O(1)
g.**locations**()	return **GridLocationRange** for entire grid	O(1)
g.**numCols**()	return count of columns	O(1)
g.**numRows**()	return count of rows	O(1)
g.**set**(*row*, *col*, *val*) or *g*[*row*][*col*] = *val* or *g*[*location*] = *val*	replace value at given row/column location	O(1)

GridLocation

GridLocation(*row*, *col*)	constructor
loc.**row**	access row field
loc.**col**	access col field

GridLocationRange

GridLocationRange(*startRow*, *startCol*, *endRow*, *endCol*)	constructor, start/end locations are inclusive
r.**contains**(*loc*)	return **true** if location contained in range
r.**isEmpty**()	return **true** if range is empty
r.**startLocation**() *r*.**endLocation**()	return start/end as **GridLocation**
for (GridLocation *loc*: *r*)	iterate over locations in range

Stack<T>

s.**clear**()	remove all elements	O(1)
s.**push**(*val*)	add value to top of stack	O(1)
s.**pop**()	remove/return top value **pop**/**peek** error if empty	O(1)
s.**peek**()	return top value without removing	O(1)
s.**isEmpty**()	return **true** if no elements	O(1)
s.**size**()	return count of elements	O(1)

Queue<T>

q.**clear**()	remove all elements	O(N)
q.**enqueue**(*val*)	add value to back of queue	O(1)
q.**dequeue**()	remove/return front value **dequeue**/**peek** error if empty	O(1)
q.**peek**()	return front value without removing	O(1)
q.**isEmpty**()	return **true** if no elements	O(1)
q.**size**()	return count of elements	O(1)

Set<T>, HashSet<T>

s.**add**(*val*) or *s* **+=** *val*	add value to set; if a duplicate, no effect	O(log N), O(1)
s.**clear**()	remove all elements	O(N)
s.**contains**(*val*)	return **true** if value contained in set	O(log N), O(1)
s.**first**()	return first element from set (does not remove it)	O(log N), O(1)
s.**isEmpty**()	return **true** if no elements	O(1)
s1.**isSubsetOf**(*s2*)	return **true** if ***s2*** contains all elements of ***s1***	O(N)
s.**remove**(*val*) or *s* **-=** *val*	remove value from set if contained	O(log N), O(1)
s.**size**()	return count of elements	O(1)
s1 **==** *s2*, *s1* **!=** *s2*	operators for set equality testing	O(N)
s1.**unionWith**(*s2*)	change ***s1*** to add all elements of ***s2***	O(NlogN), O(N)
s1.**intersect**(*s2*)	change ***s1*** to remove all elements not in ***s2***	O(NlogN), O(N)
s1.**difference**(*s2*)	change ***s1*** to remove all elements of ***s2***	O(NlogN), O(N)

CS106B FINAL REFERENCE SHEET

Map<K,V>, HashMap<K,V>

m.**clear**()	remove all key/value pairs	O(N)
m.**containsKey**(*key*)	return **true** if map contains a pair for given key	O(log N), O(1)
m.**get**(*key*) or *m*[*key*]	return value paired with given key (or a default value such as **0**, **false**, **""** if key is not present)	O(log N), O(1)
m.**isEmpty**()	return **true** if no key/value pairs	O(1)
m.**keys**()	create **Vector** copy of all keys	O(N)
m.**put**(*key*, *val*) or *m*[*key*] = *val*	add a pairing of given key to given value	O(log N), O(1)
m.**remove**(*key*)	remove any existing pairing for given key	O(log N), O(1)
m.**size**()	return count of key/value pairs	O(1)
m.**values**()	create **Vector** copy of all values	O(N)

A for-each loop on a map iterates over the *keys*, not the *values*.

PriorityQueue<V>

pq.**clear**()	remove all entries	O(N)
pq.**dequeue**()	remove/return value of frontmost entry, frontmost = most urgent priority, **dequeue**/**peek** error if empty	O(log N)
pq.**enqueue**(*val, priority)*	add entry for value with given priority	O(log N)
pq.**isEmpty**()	return **true** if no entries	O(1)
pq.**peek**()	return value of frontmost entry	O(1)
pq.**peekPriority**()	return priority of frontmost entry	O(1)
pq.**size**()	return count of entries	O(1)

Lexicon

lex.**contains**(*word*)	return **true** if given word contained in lexicon	O(1)
lex.**containsPrefix**(*prefix*)	return **true** if any word in lexicon starts with given prefix	O(1)

string, strlib.h

str.**at**(*i*) or *s*[*i*]	return character at given 0-based index
str.**append**(*text*)	add text to end of string *(in-place)*
str.**compare**(*str2*)	return -1, 0, or 1 depending on relative ordering
str.**erase**(*i*, *length*)	delete text of given length starting at given index *(in-place)*
str.**find**(*text*)	return first index of matching text (or **string::npos** if not found)
str.**insert**(*i*, *text*)	add text at a given index *(in-place)*
str.**length**() or *str*.**size**()	return count of characters
str.**replace**(*i*, *length*, *text*)	replace given length chars at given index with text *(in-place)*
str.**substr**(*start*, *length*) or *str*.**substr**(*start*)	return new string consisting of length characters from given start index if length argument omitted, grabs from start index to end of string
endsWith(*str*, *suffix*), **startsWith**(*str*, *prefix*)	return **true** if string begins or ends with the given prefix/suffix
integerToString(*i*), **stringToInteger**(*str*)	conversion between number and string
stringContains(*str*, *text*)	return **true** if text contained in string
stringSplit(*str*, *separator*)	divide a string into Vector of substrings divided by separator
toLowerCase(*str*), **toUpperCase**(*str*)	return new upper/lowercase string

random.h

randomChance(*probability*)	return random **bool** of **true/false** with the given probability of **true** from 0..1
randomInteger(*min*, *max*)	return random integer in range [*min-max*], inclusive

SimpleTest

```
STUDENT_TEST("Example test cases") {
     Vector<int> v;
     EXPECT(v.isEmpty());
     EXPECT_EQUAL(1 + 2, 3);
     EXPECT_ERROR(empty[0]);
}
```