

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>

<code>v.add(val)</code> or <code>v += val</code>	appends value to end of vector	$O(1)$ *
<code>v.clear()</code>	removes all elements	$O(1)$
<code>v.get(i)</code> or <code>v[i]</code>	returns value at given index	$O(1)$
<code>v.insert(i, val)</code>	inserts at given index, shifting subsequent values right	$O(N)$
<code>v.isEmpty()</code>	returns <code>true</code> if there are no elements	$O(1)$
<code>v.remove(i)</code>	removes value at given index, shifting subsequent values left	$O(N)$
<code>v.set(i, val)</code> or <code>v[i] = val</code>	replaces value at given index	$O(1)$
<code>v.size()</code>	returns number of elements	$O(1)$
<code>v.subList(start, length)</code>	returns new vector containing subrange of elements	$O(N)$

Grid<T>

<code>g.get(row, col)</code> or <code>g[row][col]</code> or <code>g[location]</code>	returns value stored at given row/column location	$O(1)$
<code>g.inBounds(row, col)</code> or <code>g.inBounds(location)</code>	returns <code>true</code> if given row/column index is within $(0, 0) \dots (R, C)$	$O(1)$
<code>g.numCols()</code>	returns number of columns C	$O(1)$
<code>g.numRows()</code>	returns number of rows R	$O(1)$
<code>g.set(row, col, val)</code> or <code>g[row][col] = val</code> or <code>g[location] = val</code>	changes value stored at given row/column location	$O(1)$

GridLocation

<code>GridLocation(row, col)</code>	constructor
<code>loc.row</code>	access row field
<code>loc.col</code>	access col field

GridLocationRange

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

Stack<T>

<code>s.clear()</code>	removes all elements	$O(1)$
<code>s.push(val)</code>	adds value to top of stack	$O(1)$
<code>s.pop()</code>	removes/returns top value pop/peek error if empty	$O(1)$
<code>s.peek()</code>	returns top value without removing	$O(1)$
<code>s.isEmpty()</code>	returns <code>true</code> if no elements	$O(1)$
<code>s.size()</code>	returns number of elements	$O(1)$

Queue<T>

<code>q.clear()</code>	removes all elements	$O(N)$
<code>q.enqueue(val)</code>	adds value to back of queue	$O(1)$
<code>q.dequeue()</code>	removes/returns front value dequeue/peek error if empty	$O(1)$
<code>q.peek()</code>	returns front value without removing	$O(1)$
<code>q.isEmpty()</code>	returns <code>true</code> if no elements	$O(1)$
<code>q.size()</code>	returns number of elements	$O(1)$

Set<T>, HashSet<T>

<code>s.add(val)</code> or <code>s += val</code>	adds to set; if a duplicate, no effect	$O(\log N)$, $O(1)$
<code>s.clear()</code>	removes all elements	$O(N)$
<code>s.contains(val)</code>	returns <code>true</code> if value is found in the set	$O(\log N)$, $O(1)$
<code>s.first()</code>	returns first element from set (does not remove it)	$O(\log N)$, $O(1)$
<code>s.isEmpty()</code>	returns <code>true</code> if there are no elements	$O(1)$
<code>s.isSubsetOf(s2)</code>	returns <code>true</code> if <code>s2</code> contains all elements of <code>s</code>	$O(N)$
<code>s.remove(val)</code> or <code>s -= val</code>	removes value from set, if present	$O(\log N)$, $O(1)$
<code>s.size()</code>	returns number of elements	$O(1)$
<code>s1 == s2</code> , <code>s1 != s2</code>	operators for set equality testing	$O(N)$
<code>s1.unionWith(s2)</code>	returns set of all elements of <code>s2</code> and <code>s1</code>	$O(N)$
<code>s1.intersect(s2)</code>	returns set of all elements in both <code>s1</code> and <code>s2</code>	$O(N)$
<code>s1.difference(s2)</code>	returns set of all elements <code>s1</code> not in <code>s2</code>	$O(N)$

CS106B FINAL REFERENCE SHEET

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

<code>m.clear()</code>	removes all key/value pairs	O(N)
<code>m.containsKey(key)</code>	returns true if map contains a pair for the given key	O(log N), O(1)
<code>m.get(key)</code> or <code>m[key]</code>	returns value paired with the given key (or a default value such as 0 , false , "" if key is not present)	O(log N), O(1)
<code>m.isEmpty()</code>	returns true if there are no key/value pairs	O(1)
<code>m.keys()</code>	returns a Vector copy of all keys in the map	O(N)
<code>m.put(key, val)</code> or <code>m[key] = val</code>	adds a pairing of the given key to the given value	O(log N), O(1)
<code>m.remove(key)</code>	removes any existing pairing for the given key	O(log N), O(1)
<code>m.size()</code>	returns number of key/value pairs	O(1)
<code>m.values()</code>	returns a Vector copy of all values in the map	O(N)

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

PriorityQueue<V>

<code>pq.clear()</code>	removes all entries	O(1)
<code>pq.dequeue()</code>	removes/returns value of frontmost entry, frontmost = most urgent priority, dequeue/peek error if empty	O(log N)
<code>pq.enqueue(val, priority)</code>	adds entry for value with given priority	O(log N)
<code>pq.isEmpty()</code>	returns true if no entries	O(1)
<code>pq.peek()</code>	returns value of frontmost entry	O(1)
<code>pq.peekPriority()</code>	returns priority of frontmost entry	O(1)
<code>pq.size()</code>	returns number of entries	O(1)

Lexicon

<code>L.contains(word)</code>	returns true if the word is found in the lexicon	O(log N)
<code>L.containsPrefix(text)</code>	returns true if any word starts with this prefix text	O(log N)

Strings

<code>str.at(i)</code> or <code>s[i]</code>	character at a given 0-based index in the string
<code>str.append(str)</code>	add text to the end of a string (<i>in-place</i>)
<code>str.compare(str)</code>	return -1, 0, or 1 depending on relative ordering
<code>str.erase(i, length)</code>	delete text from a string starting at given index (<i>in-place</i>)
<code>str.find(str)</code> <code>str.rfind(str)</code>	returns the first or last index where the start of the given string or character appears in this string (or string::npos if not found)
<code>str.insert(i, str)</code>	add text into a string at a given index (<i>in-place</i>)
<code>str.length()</code> or <code>str.size()</code>	number of characters in this string
<code>str.replace(i, len, str)</code>	replaces len chars at given index with new text (<i>in-place</i>)
<code>str.substr(start, length)</code> or <code>str.substr(start)</code>	returns the next length characters beginning at index start (inclusive); if length is omitted, grabs from start to the end of the string
<code>endsWith(str, suffix)</code> , <code>startsWith(str, prefix)</code>	returns true if the string begins or ends with the given prefix/suffix
<code>integerToString(int)</code> , <code>stringToInteger(str)</code>	returns a conversion between numbers and strings
<code>stringContains(str, substr)</code>	true if substr contained in str
<code>stringSplit(str, separator)</code>	breaks apart a string into a Vector of substrings divided by separator
<code>toLowerCase(str)</code> , <code>toUpperCase(str)</code>	returns an upper/lowercase version of a string
<code>trim(str)</code>	returns string with any surrounding whitespace removed

SimpleTest

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