CS106B Instructor: Cynthia Lee

Spring 2016 Practice Exam

# LIBRARY REFERENCE SHEET

*This document has been updated from the midterm to include data structures we covered after the midterm.*

---

**Summary of Relevant Data Types**

We tried to include the most relevant member functions for the exam, but not all member functions are listed. You are free to use ones not listed here that you know exist. **You do <u>not</u> need `#include`.**

```
class string {
  bool empty() const; // O(1)
  int size() const; // O(1)
  int find(char ch) const; // O(N)
  int find(char ch, int start) const; // O(N)
  string substr(int start) const; // O(N)
  string substr(int start, int length) const; // O(N)
  char& operator[](int index);  // O(1)
  const char& operator[](int index) const; // O(1)
};
string toUpperCase(string str);
string toLowerCase(string str);

class Vector {
  bool isEmpty() const; // O(1)
  int size() const; // O(1)
  void add(const Type& elem); // operator+= used similarly – O(1)
  void insert(int pos, const Type& elem); // O(N)
  void remove(int pos); // O(N)
  Type& operator[](int pos); // O(1)
};

class Grid {
  int numRows() const; // O(1)
  int numCols() const; // O(1)
  bool inBounds(int row, int col) const; // O(1)
  Type get(int row, int col) const; // or operator [][] also works –  O(1)
  void set(int row, int col, const Type& elem); // O(1)
};
```

```
class Stack {
  bool isEmpty() const; // O(1)
  void push(const Type& elem); // O(1)
  Type pop(); // O(1)
};

class Queue {
 bool isEmpty() const; // O(1)
 void enqueue(const Type& elem); // O(1)
 Type dequeue(); // O(1)
};

class Map {
 bool isEmpty() const; // O(1)
 int size() const; // O(1)
 void put(const Key& key, const Value& value); // O(logN)
 bool containsKey(const Key& key) const; // O(logN)
 Value get(const Key& key) const; // O(logN)
 Value& operator[](const Key& key); // O(logN)
};
```
*Example for loop:* `for (Key key : mymap){…}`

```
class HashMap {
 bool isEmpty() const; // O(1)
 int size() const; // O(1)
 void put(const Key& key, const Value& value); // O(1)
 bool containsKey(const Key& key) const; // O(1)
 Value get(const Key& key) const; // O(1)
 Value& operator[](const Key& key); // O(1)
};
```
*Example for loop:* `for (Key key : mymap){…}`

```
class Set {
 bool isEmpty() const; // O(1)
 int size() const; // O(1)
 void add(const Type& elem); // operator+= also adds elements – O(logN)
 bool contains(const Type& elem) const; // O(logN)
};
```
*Example for loop:* `for (Type elem : mymap){…}`

```
class Lexicon {
  int size() const; // O(1)
  bool isEmpty() const; // O(1)
  void clear(); // O(N)
  void add(string word); // O(W) where W is word.length()
  bool contains(string word) const; // O(W) where W is word.length()
  bool containsPrefix(string pre) const; // O(W) where W is pre.length()
```

```
};
Example for loop: for (string str : english){…}

/* GRAPH-RELATED FUNCTIONS */

struct Vertex  {
    std::string name;
    Set<Edge*> arcs;
    Set<Edge*>& edges;
    bool visited;
    Vertex* previous;
    Color getColor();
    setColor(int color);
};
struct Edge {
    Vertex* start;
    Vertex* end; //alias for finish, which also works
    double weight; //alias for cost, which also works
    bool visited;
};

class BasicGraph : public Graph<Vertex, Edge> {
public:
BasicGraph();
    bool isNeighbor(Vertex* v1, Vertex* v2) const; // O(log V)
    bool isNeighbor(string name1, string name2) const; // O(log V)
    const Set<Vertex*> getNeighbors(Vertex* vertex) const; // O(log V)
    const Set<Vertex*> getNeighbors(string vertex) const; // O(log V)
    Edge* getEdge(Vertex* v1, Vertex* v2) const; // O(log V + log E)
    Edge* getEdge(std::string v1, std::string v2) const; // O(log V + log E)
    const Set<Edge*>& getEdgeSet() const; // O(log V)
    const Set<Edge*>& getEdgeSet(Vertex* v) const; // O(log V)
    const Set<Edge*>& getEdgeSet(std::string v) const; // O(log V)
    Vertex* getVertex(std::string name) const; // O(log V)
    const Set<Vertex*>& getVertexSet() const; // O(log V)
    bool containsEdge(Vertex* v1, Vertex* v2) const; // O(log E)
    bool containsEdge(string name1, string name2) const; // O(log E)
    bool containsEdge(Edge* edge) const; // O(log E)
    void resetData(); // O(V + E)

    /* For the graph problem, you are not allowed to edit the graph
     * structure, so we omitted addVertex/removeVertex, addEdge/removeEdge */
}
```