COSC-302

EXAM 1 REVIEW

by Tyler Senter and Brian Horsburgh

Overview

Sorting
- Basic Algorithms
- Computational Cost
- What is stability?
- Examples
- List-Based Sorting
  - Smart pointers

Containers
- Trees
  - B-Tree
  - Red-Black Tree
  - Binary Search Tree
- Binary Heap

Overview (cont.)

Recurrence Relations

- Telescoping

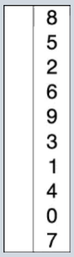
Binary vs. Formatted I/O

- C Style
- C++ Style

Basic Algorithms

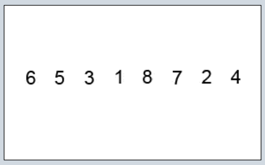
Selection Sort

1. Select the smallest element in the right subarray
2. Swap that element with the right-most element of the left subarray



Bubble Sort

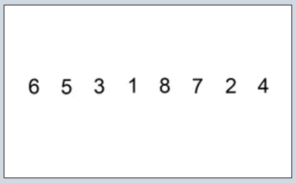
1. Compare two adjacent elements
2. If $\mathrm{rhs} < \mathrm{lhs}$, swap the elements



Basic Algorithms (cont.)

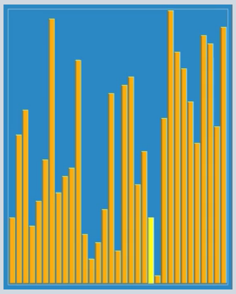
Insertion Sort

1. Grab the left-most element in the unsorted list
2. Insert that element into its proper spot in the sorted list



Shell Sort

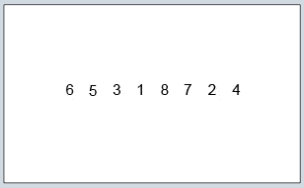
1. Compare two adjacent elements
2. If $\text{rhs} < \text{lhs}$, swap the elements



Basic Algorithms (cont.)

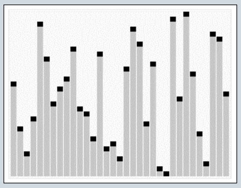
Mergesort

1. Split the list into two sublists recursively until they reach a length of 1
2. Connect the lists back in the same order by placing the smallest elements first



Quicksort

1. Choose a pivot and arrange all elements so anything less than the pivot is to its left and anything greater is to its right
2. Take the 2 sublists created and do this again until the lists reach length 1



Computational Cost

	Selection	Bubble	Insertion	Shell	Mergesort	Quicksort
Best Case	$O(N^2)$	$O(N)$	$O(N)$	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$
Average Case	$O(N^2)$	$O(N^2)$	$O(N^2)$	varies*	$O(N \log N)$	$O(N \log N)$
Worst Case	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N \log N)$	$O(N^2)$**

* Depends on the gap sequence:

$O(N^2)$: $\frac{N}{2^k} = N, \frac{N}{2}, \cdots, 4,2,1$

$O(N^{1.5})$: $2^k - 1 = 1,3,7,15, \ldots$

** Depends on the pivot selection

Stability in Sorting

A sorting algorithm is **stable** if elements considered “equal” keep their initial order.

Mergesort, **insertion sort**, and **bubble sort** are stable algorithms, whereas **quicksort**, **shell sort**, and **selection sort** are unstable.

std::sort(…) implements **introsort** (hybrid of quicksort and heapsort), but is “unstable”. It begins as quicksort but will fall back to heapsort if the recursion runs too deep (the maximum recursive level is based on the number of elements)

std::stable_sort(…) implements **mergesort**, and is stable

Sorting Examples

Array: E L Q K M A S M

List-Based Sorting

Insertion and selection can be applied to linked lists be using an array of “smart pointers”.

A **smart pointer** overloads the dereference operator to point to the data it represents (similar to an iterator). This way, the sorting algorithm sorts smaller elements (the smart pointers) instead of the entire element being sorted.

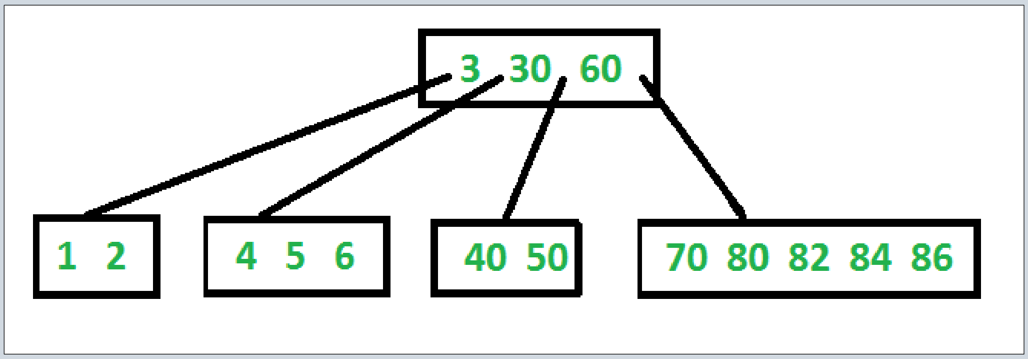
After the array is sorted, the nodes need to be relinked to the proper order. The final node must have its `next` member set to `NULL`.

B-Trees

B-Trees are multiway self balancing search trees that allow for multiple keys stored at one node

1. All leaves are at same level
2. A B-Tree is defined by the term *minimum degree* ‘t’, which is the minimum number of children a node can have
3. Every node except root must contain at least t-1 keys. Root may contain minimum 1 key
4. All nodes (including root) may contain at most 2t – 1 keys
5. Number of children of a node is equal to the number of keys in it plus 1

B-Tree (Example)



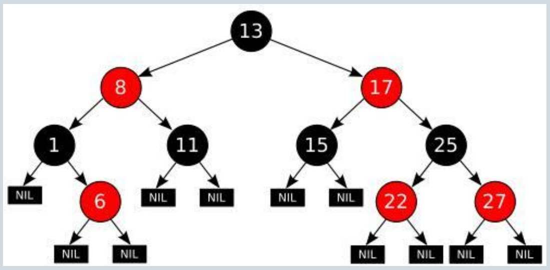
Red-Black Tree

Balanced binary search tree like AVL tree

Uses color of node to keep balanced

- Root node is always black
- A red node cannot have a red child or parent
- Every path from root to NULL has the same number of black nodes

Uses rotations and recoloring after every insert or delete to maintain balanced property

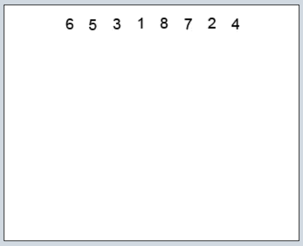


Binary Tree/Heap

A binary search tree is another way to create sorted data

Traversing a tree from left to right will give the sorted data

For a **heap,** the root element will be either the smallest or largest and a sorted list can be made by adding the root to an array and deleting it to get the next in line root



Binary Time Complexity

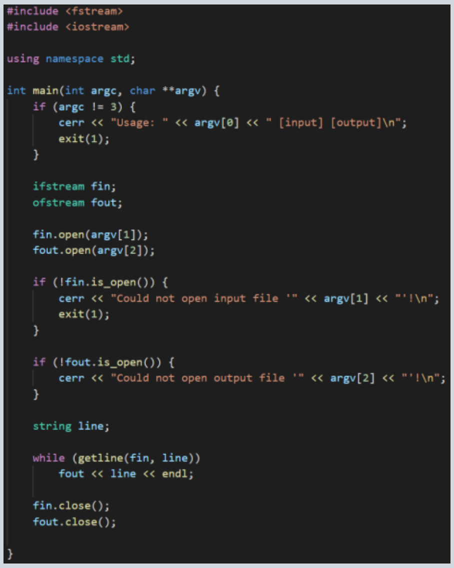
	Building	Sorting	Heapify
Binary Search Tree	$O(N \log N)$	$O(0)$**	N/A
Binary Heap	$O(N \log N)$*	$O(N \log N)$	$O(\log N)$

*Building a heap calls heapify *N* times
**A binary search tree is sorted by definition when built

Telescoping

Telescoping is a way to show that a certain algorithm has a particular Big-O complexity. The examples we have dealt with have shown that quicksort has a best case Big-O of $O(N \log N)$.

$$\left.\begin{aligned}
T(N) &= 2T\left(\frac{N}{2}\right) + Nc \Rightarrow \frac{T(N)}{N} = \cancel{\frac{T(N/2)}{N/2}} + c \\
&= 4T\left(\frac{N}{4}\right) + \frac{N}{2}c \Rightarrow \cancel{\frac{T(N/2)}{N/2}} = \cancel{\frac{T(N/4)}{N/4}} + c \\
&= 8T\left(\frac{N}{8}\right) + \frac{N}{4}c \Rightarrow \cancel{\frac{T(N/4)}{N/4}} = \cancel{\frac{T(N/8)}{N/8}} + c \\
&\vdots \\
&= \frac{T(N/N)}{N/N} = T(1) = \mathrm{c}
\end{aligned}\right\}
\begin{aligned}
&\Rightarrow \frac{T(N)}{N} = \overbrace{c + c + \cdots + c + c}^{\log N} \Rightarrow c \log N \\
&\Rightarrow T(N) = cN \log N \\
&\Rightarrow O(cN \log N) = O(N \log N)
\end{aligned}$$



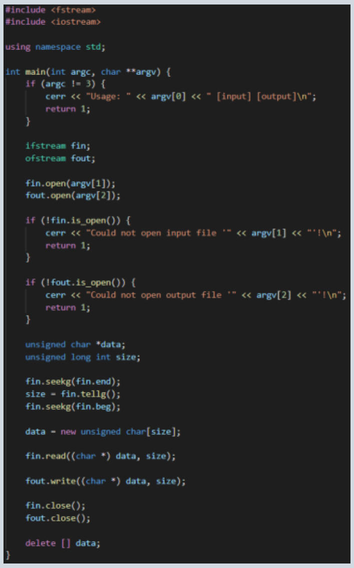
```cpp
#include <fstream>
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
    if (argc != 3) {
        cerr << "Usage: " << argv[0] << " [input] [output]\n";
        exit(1);
    }

    ifstream fin;
    ofstream fout;

    fin.open(argv[1]);
    fout.open(argv[2]);

    if (!fin.is_open()) {
        cerr << "Could not open input file '" << argv[1] << "'!\n";
        exit(1);
    }

    if (!fout.is_open()) {
        cerr << "Could not open output file '" << argv[2] << "'!\n";
    }

    string line;

    while (getline(fin, line))
        fout << line << endl;

    fin.close();
    fout.close();

}
```

Formatted (Text) Files

Certain files are stored in human-readable formats. These formats include .json, .xml, .cpp and more types. If a file of this type is opened in a primitive file-reading program, like Window's Notepad, the contents could be read normally.

In formatted files, every character takes exactly one byte of storage (so a 1 kilobyte file is 1024 bytes, or 1024 characters) due to the ASCII table.

Formatted files are read and written with the ifstream and ofstream classes, respectively. Files can then be read from and written to exactly like reading and writing standard input and output (cin/cout).



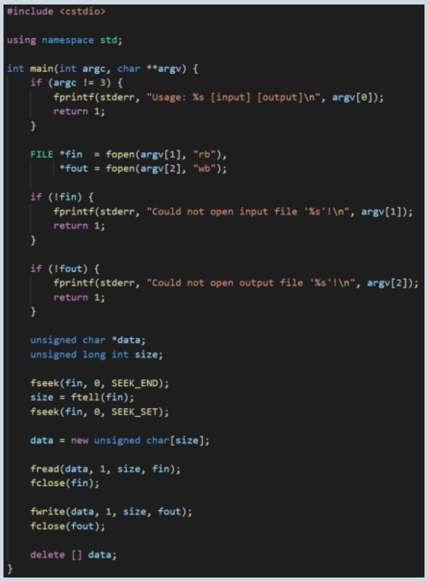
```cpp
#include <fstream>
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
    if (argc != 3) {
        cerr << "Usage: " << argv[0] << " [input] [output]\n";
        return 1;
    }

    ifstream fin;
    ofstream fout;

    fin.open(argv[1]);
    fout.open(argv[2]);

    if (!fin.is_open()) {
        cerr << "Could not open input file '" << argv[1] << "'!\n";
        return 1;
    }

    if (!fout.is_open()) {
        cerr << "Could not open output file '" << argv[2] << "'!\n";
        return 1;
    }

    unsigned char *data;
    unsigned long int size;

    fin.seekg(fin.end);
    size = fin.tellg();
    fin.seekg(fin.beg);

    data = new unsigned char[size];

    fin.read((char *) data, size);

    fout.write((char *) data, size);

    fin.close();
    fout.close();

    delete [] data;
}
```

Binary Files (C++ Style)

Certain files cannot be stored easily in a human-readable format, such as audio, video, and Microsoft Word documents. If a file of this type is opened in a primitive file-reading program, like Window's Notepad, the contents would be nonsensical and jumbled, if the file could even be opened.

In binary files, the meaning of every byte of data is up to the developer, so opening a file with the wrong program will likely create meaningless junk.

Binary files are read and written with a FILE pointer. Files can then be read from and written to using various helper functions, like fread, fwrite, fseek and more.



```
#include <cstdio>

using namespace std;

int main(int argc, char **argv) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s [input] [output]\n", argv[0]);
        return 1;
    }

    FILE *fin  = fopen(argv[1], "rb"),
         *fout = fopen(argv[2], "wb");

    if (!fin) {
        fprintf(stderr, "Could not open input file '%s'!\n", argv[1]);
        return 1;
    }

    if (!fout) {
        fprintf(stderr, "Could not open output file '%s'!\n", argv[2]);
        return 1;
    }

    unsigned char *data;
    unsigned long int size;

    fseek(fin, 0, SEEK_END);
    size = ftell(fin);
    fseek(fin, 0, SEEK_SET);

    data = new unsigned char[size];

    fread(data, 1, size, fin);
    fclose(fin);

    fwrite(data, 1, size, fout);
    fclose(fout);

    delete [] data;
}
```

Binary Files (C-style)

Certain files cannot be stored easily in a human-readable format, such as audio, video, and Microsoft Word documents. If a file of this type is opened in a primitive file-reading program, like Window's Notepad, the contents would be nonsensical and jumbled, if the file could even be opened.

In binary files, the meaning of every byte of data is up to the developer, so opening a file with the wrong program will likely create meaningless junk.

Binary files are read and written with a FILE pointer. Files can then be read from and written to using various helper functions, like fread, fwrite, fseek and more.

Formatted vs. Binary Files

Formatted Files

Advantages
- Useful for storing simple text data
- Human-readable

Disadvantages
- Cannot store complex data
- Not as efficient as binary files

Binary Files

Advantages
- Useful for storing complex data
- Data stored more efficiently

Disadvantages
- Not human-readable
- Each byte may have its own meaning or be part of a larger piece of data