UCLA Samueli Computer Science

CS32: Introduction to Computer Science II

Discussion Week 10

Junheng Hao, Arabelle Siahaan

June 7, 2019

Announcements

- **Final exam is scheduled on June 8 (tomorrow)!**
- Check the course announcements on time and place for the final exam.
  - From **11:25am to 1:55pm** in **Haines 39** if your last name begins with K, A, L, A, H, C, A, P, T, U, R, or E.
  - from **11:40am to 2:10pm** in **Moore 100** if your last name begins with B, D, F, G, I, J, M, N, O, Q, S, V, W, X, Y, or Z.

Outline Today

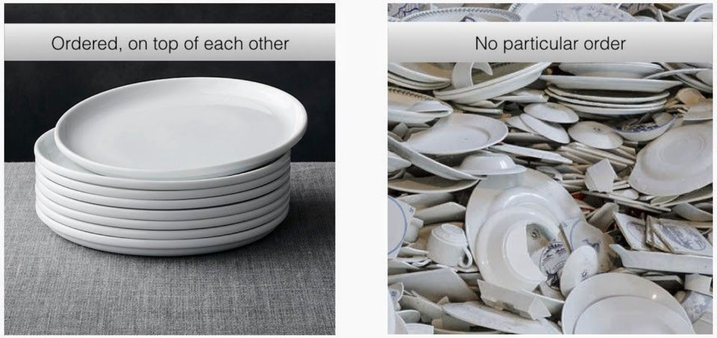
UCLA Samueli Computer Science

- Heap
- Priority queue
- Graph (not in final exam)
- Final exam review
- Q&A

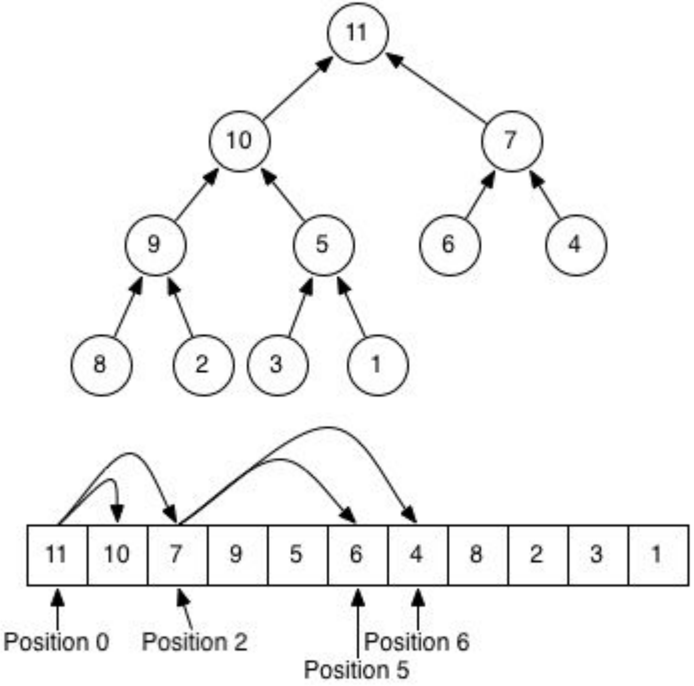
Heap

Definition and properties

- About heap
  - Heap is considered as complete binary tree.
  - Every nodes carries a value greater than or equal to its children (for MaxHeap).
  - Often implemented as an array.
  - Body structure of priority queue.

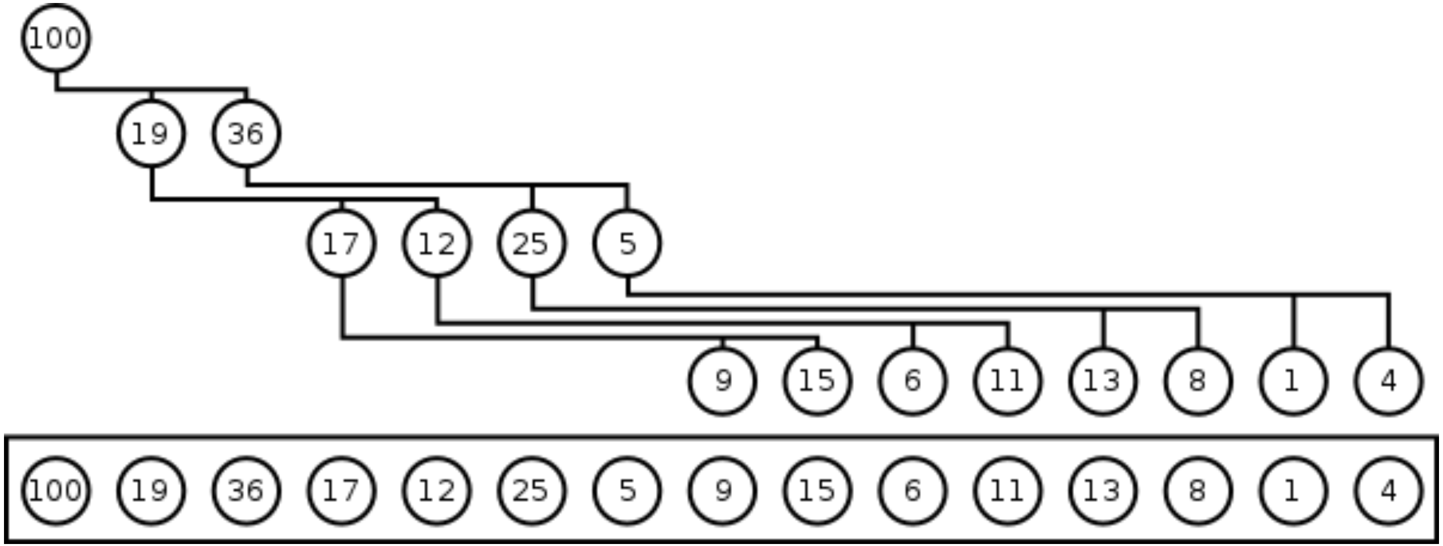


Stack **Heap**



Heap

Implementation by arrays



(Almost) full binary tree in an array implementation.

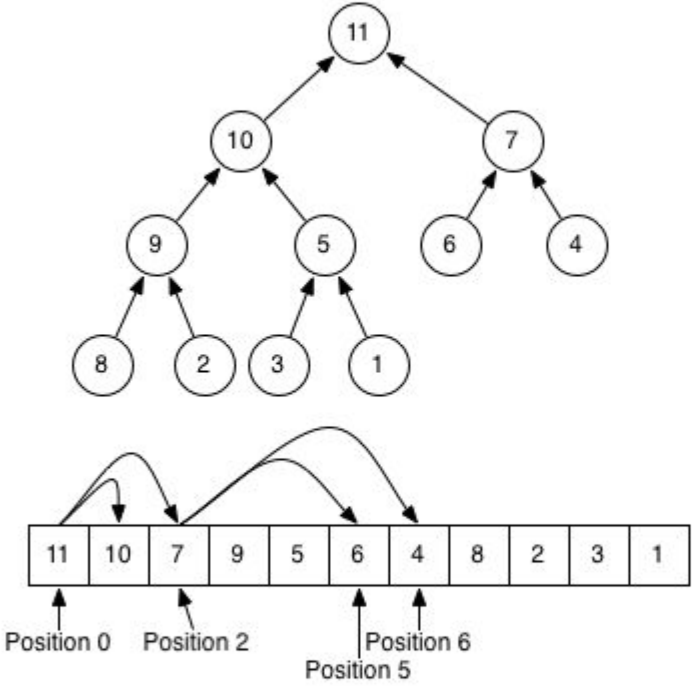
Heap

Standard operations

UCLA Samueli Computer Science

- Three operations of heaps
  - Find Max (search)
  - Insert Node (insert)
  - Delete Max (delete)

- How to implement `FindMax()` function of a heap?
  - Well, that is just too obvious!



Heap & Heapsort

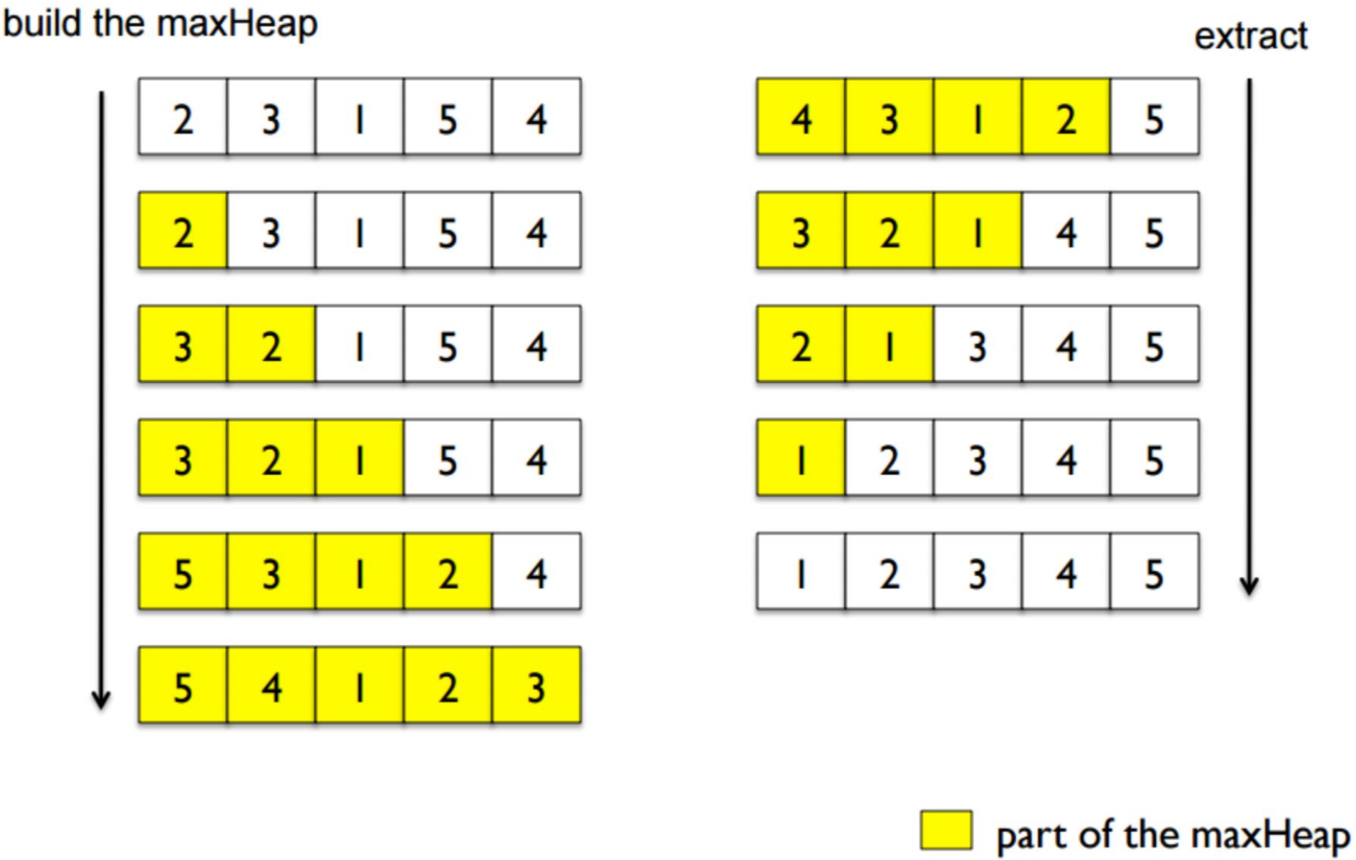
Complexity of heap operations

- Find Max → O(1)
- Insert → O(log *n*)
- Delete Max Node → O(log *n*)

6 5 3 1 8 7 2 4

- Bonus: How can you sort based on heap?
  - Insert all elements into a heap.
  - Extract the maximum element from the heap one by one.
  - Check the example in [Wikipedia]
- What is the complexity of heapsort?
  - $O(n \log n)$

In-place Heapsort (with an array)



Heapsort Problem 1

Find k largest numbers

- How can we efficiently find ***k* largest numbers** from n numbers? (n>>k, but k is not small)
    - Sort? → O(*nlogn*)
    - Scan *k* times by linear search? → O(*nk*)

- Use heapsort
    - Only keep the largest
    - Whether to use MaxHeap or MinHeap?
    - Overall complexity?

Min Heap!
Pop out the `min` from heap and insert a number, if it's larger than `min`.

O(*nlogk*)

Heapsort Problem 2

Find median from a streaming data

UCLA Samueli Computer Science

How to find the median of the streaming data?

That is, implementing the following program:

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

Here you may use two heaps to store all the coming data → find median in O(1) time.

```
class MedianFinder {
public:
  MedianFinder() {
    // construction
  }

  void addNum(int num) {
    // add new integers from stream
  }

  double findMedian() {
    // return the median
  }
private:
    // define your private data member(s)
};
```

Heapsort Problem 3

Merge *k* linked list

UCLA Samueli Computer Science

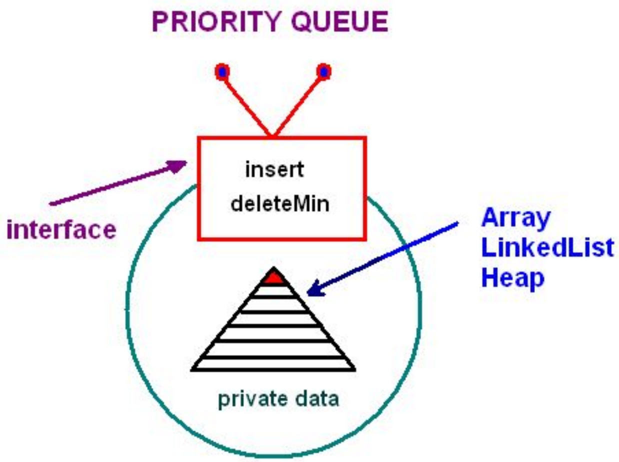
- How to merge k sorted linked lists? Each list has n nodes.
- Solution 1: Brute forth $\rightarrow O(nk^2)$
  - Keep linear searching the k heads and fetching the smallest until all lists are empty
- Solution 2: Use MinHeap $\rightarrow O(nk\log k)$
  - Insert the head of each list to the heap.
  - Each time we pop-out a node from the heap and append it to the result list, insert the next node of that node from its list to the heap (until heap is empty)
- Solution 3: Merge sort $\rightarrow O(nk\log k)$
  - Merge each pair of sorted lists. k sorted lists become k/2 sorted lists.
  - Repeat the list merge from k/2 to k/4… (until everything is merged into 1 list.)

Priority Queue

Definations & Properties

- Abstract data type (providing interface)
- Heap-based Priority Queue over other implementations:

Implementations	Insert	DeleteMin	FindMin
Ordered array	O(n)	O(1)	O(1)
Ordered list	O(n)	O(1)	O(1)
Unordered array	O(1)	O(n)	O(n)
Unordered list	O(1)	O(n)	O(n)
Binary Heap	**O(log n)**	**O(log n)**	**O(1)**



Credit to: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html

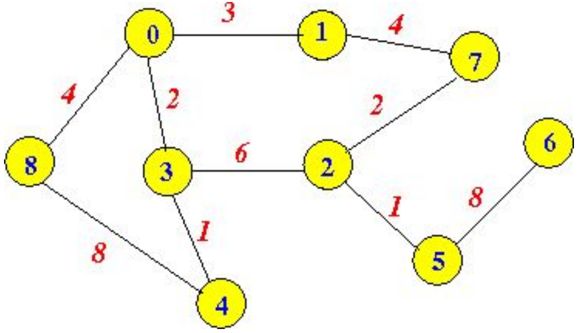
Graph

Introduction, BFS & DFS

UCLA Samueli Computer Science

- Terms
  - Nodes, edges
  - Adjacency matrix, adjacency list
- Complexity to store graph, add/remove vertex, add/remove edge
- Graph algorithms
  - BFS: Breadth First Search Graph traversal algorithm
  - DFS: Depth First Search Graph traversal algorithm [Link]
  - Dijkstra's Algorithm: Compute the minimum cost paths from a node (e.g., node 1) to all other node in the graph [Link]
  - Prim's Algorithm: finding Minimum cost Spanning Tree [Link]
  - More in CS180

Any other methods to store a graph?
Incidence Matrix, etc.



Final Review

UCLA Samueli Computer Science

- https://kycode.me/CS-32/
- http://getacollegelife.tumblr.com/post/70756494466/my-buddy-rachel-fangs-cs31-and-cs32-cheat
- Final exam practice
  - http://web.cs.ucla.edu/classes/spring19/cs32/Sampleproblems/ChangFinalPractice.pdf
  - http://web.cs.ucla.edu/classes/spring19/cs32/Sampleproblems/ChoiFinalPractice.pdf

Final Review: Topics in CS32

You really have learned a lot!

- Modern features about C++
    - Resource management
    - Inheritance and polymorphism
    - Templates, Iterators, STL containers
- Data Structures
    - Array, Vector
    - Linked List
    - Stack, Queue
    - Tree, Heap, Graphs
    - Hash Table
- Algorithms and complexity analysis
    - Recursion
    - Big-O
    - Sorting

David's Reminders: Part 1

- Classes containing members of a class type -- order of construction and destruction, initializer lists.
- Destructors, copy constructors, and assignment operators.
- Base/derived classes and inheritance of members
- Automatic conversion of `Derived*` to `Base*` and `Derived&` to `Base&`
- Virtual functions, overriding member function implementations, pure virtual functions and abstract base classes, virtual
- Construction order and destruction order
- Recursion

David’s Reminders: Part 2

- Various sorting algorithms.
- Preorder traversal and postorder, traversal and then introduced binary trees and more tree algorithms especially using recursion.
- Data structures and corresponding complexity (those Big-Os)
- Clear on open hash tables, load factor, and so on.
    - They should almost *never* assume that collisions are impossible. Even if you have a hash table with 10000 buckets and are storing only 100 items, you may well have 2 of those keys that happen to end up in the same bucket. It's a common beginner mistake to write code that assumes that if a bucket is not empty, you've found the item and the first item in the bucket is it.

David's Reminders: Not in CS32 final

UCLA Samueli Computer Science

- Multiple inheritance, private inheritance, protected members in inheritance
- Details of AVL trees, 2-3 tree or red-black trees, or more advanced trees.
- Specific hash function (FNV-1)
- Graphs
- Accurate names of member function of STL containers.

"The important thing to know is imply that there exist algorithms that keep the trees more or less balanced so that the average and worst case insert, delete and lookup performance is **O(log *n*)**." -- David

Helpful Resources

UCLA Samueli Computer Science

- TA Links (check links)
- Previous TA Mark Edmonds's CS32 worksheet (Spring 2016): [Link]
- UCLA CS Practice Problems [Link]
- Final exam practice from instructors. Check announcements here! [Link]

Homework 5 Problem 4

- Background:
  - A `pair<T1, T2>` is a simple struct with two data members, one of type **`T1`** and one of type **`T2`**.
  - A `set<K>` and a `map<K, V>` are organized as approximately balanced binary search trees.
  - An **`unordered_set<K>`** and an `unordered_map<K, V>` are organized as hash tables that never allow the load factor to exceed some constant, and a loop that visits every item in a hash table of **N** items is **O(N)**.
  - For the keys to be hashed, the hash function used produces uniformly distributed results.

- Setting:
  - Suppose UCLA has **C** courses each of which has on average **S** students enrolled.
  - For this problem, courses are represented by strings (e.g. "CS 32"), and students by their int UIDs.
  - We will consider a variety of data structures, and for each determine the big-O time complexity of the appropriate way to use that data structure to determine whether a particular student **s** is enrolled in course **c**.

Homework 5 Problem 4 (Cont'd)

UCLA Samueli Computer Science

Data structure	Note & Task	Complexity
`vector<pair<string, list<int>>>`	Outer vector: a course and all the students in that class, with students being sorted in order. Task: whether a particular *s* is enrolled in *c*	$O(C+S)$
`map<string, list<int>>`	students in each list are in no particular order Task: whether a particular *s* is enrolled in *c*	$O(\log C+S)$
`map<string, set<int>>`	whether a particular *s* is enrolled in *c*	$O(\log C+\log S)$
`unordered_map<string, set<int>>`	whether a particular *s* is enrolled in *c*	$O(\log S)$
`unordered_map<string, unordered_set<int>>`	whether a particular *s* is enrolled in *c*	$O(1)$
`map<string, set<int>>`	a particular *c* to write the id numbers of *all* the students in that course in sorted order	$O(\log C+S)$
`unordered_map<string, unordered_set<int>>`	for a particular course *c* to write the id numbers of *all* the students in that course in sorted order (perhaps using an additional container to help with that)	$O(S\log S)$
`unordered_map<string, set<int>>`	for a particular student *s* to write *all* the courses that student is enrolled in, in no particular order.	$O(C\log S)$

Problem Checklist

UCLA Samueli Computer Science

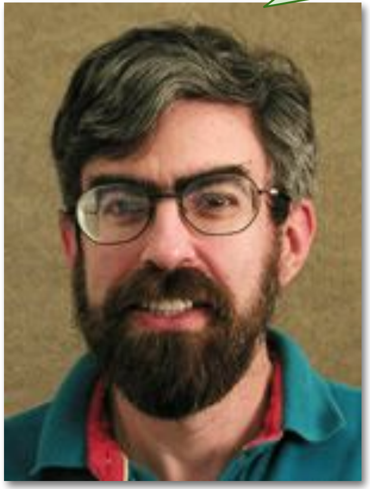
after midterm 2 ...

1. Infix to postfix conversion, postfix to infix conversion, postfix expression evaluation
2. Pre-order, in-order and post-order tree (BST) traversal (can be combined with recursion)
3. Different sorting algorithms, detailed steps, variations and complexity, especially heapsort
4. Tree recursion problem
5. Real-world applications with different data structures (with all what you have learned combined)
    - Homework 5 Problem 4 and Project 4 as an example.
    - How items are stored? Are the items sorted / ordered?
    - Complexity of operations (insertion, deletion and search)
    - If possible, how can the hash table be applied in the problem with efficient implementation and less collision?

Thank you and good luck!

UCLA Samueli Computer Science

The final exam is so easy!



Yep! Yep!

UCLA Samueli Computer Science

Break Time! (5 minutes)

Q & A

Group Exercises: Worksheet

UCLA Samueli Computer Science

- Final practice problems props to the LAs :)
- If you didn't go to the LA Final Review/ Practice, get the slides here -> tinyurl.com/CS32-LAFinal

Thank you!

Q & A