Announcements

- will do review later today
- will take questions at end
- please make sure to submit/plan hw
  - semester is going to end in 2 weeks from today
Outline

- Sorting – quick sort
- Disjoint DS
- Review for midterm

- Reading: Chapter 7.7-7.8, 8-8.3

Quick sort

- fastest currently known sort
  - Average N log N
  - Worst: N^2
Quicksort

- if one element return
- else
  - pick a pivot from the list
  - split the list around the pivot
  - return quicksort(left) + pivot + quicksort(right)

- Lets do an example

issues

- How does worst case happen?
- how to pick the pivot?
Pivot #1

- use the first element of the list

- pro/cons ?

- sorted list will always be $N^2$
Pivot #2

- choose random element for pivot

- pro/cons?

- great performance

- expensive to generate random number
Pivot #3

- Choose median value from the list

- pro/cons ?

- hmmm don’t you need a sorted list to get median?

- actually there is a linear algorithm for this 😊 will be doing it on homework
Pivot #4

- Median of 3

- since #3 isn’t cheap, can grab 3 elements and take median
  - can even use random if you don’t mind

Coding

- ok so enough theory, how do you code all this??
- arrays are much cheaper than linked lists
- lots of tricks to keep things cheap
```cpp
/**
 * Quicksort algorithm (driver).
 */

// Template
template<typename Comparable>
void quicksort(vector<Comparable> & a)
{
    quicksort(a, 0, a.size() - 1);
}

// Median
const Comparable & median3(vector<Comparable> & a, int left, int right)
{
    int center = (left + right) / 2;
    if (a[center] < a[left])
        swap(a[left], a[center]);
    if (a[right] < a[left])
        swap(a[left], a[right]);
    if (a[right] < a[center])
        swap(a[center], a[right]);

    // Place pivot at position right - 1
    swap(a[center], a[right - 1]);
    return a[right - 1];
}
```
understanding

- Ok, any idea of how to maximize cutoff point ?

- how to analyze quicksort ??
Analysis

- so how to analyze quick sort
- think how we did mergesort analysis

Quick sort

- $i =$ size of left partition
- $C_1 =$ time to choose pivot
- $C_2 =$ partitioning the set

- what do you get?
\[ T(N) = C_1 + C_2 N + T(i) + T(N - i - 1) \]

- what would be the worst case runtime?

- can you solve this with the methods from Monday?

---

**Telescope**

\[
\begin{align*}
i &= 0 \\
T(N) &= cN + T(0) + T(N - 1) \\
T(N) &= T(N - 1) + cN \\
T(N - 1) &= T(N - 2) + c(n - 2) \\
&\text{......} \\
&\text{add}
\end{align*}
\]
what is the big O runtime ??
this was pretty clear before the analysis 😊
its slow-sort 😊
what is the best case

\[ T(N) = T(1) + c \left( \frac{N(N-1)}{2} \right) \]

- i = N/2
- T(N) = 2T(N/2) + cN
For average case if you were to analyze every possible input, when left small, right large

\[ T(i) = \frac{1}{N} \sum_{j=0}^{N-1} T(j) \]
Telescope This!

\[ T(N) = cN + 2 \left[ \frac{1}{N} \sum_{j=0}^{N-1} T(j) \right] \]

telescope

multiple \(-N\)

\[ N \cdot T(N) = 2 \sum_{j=0}^{N-1} T(j) + cN^2 \]

telecope

\[ (N-1)T(N-1) = 2 \sum_{j=0}^{N-2} T(j) + (N-1)^2 \]

subtract

\[ N \cdot T(N) - (N-1)T(N-1) = 2T(N-1) + cN^2 - c(N-1)^2 \]

...
Bottom line

\[ \frac{T(N)}{N+1} = \frac{T(1)}{2} + 2C \sum_{i=0}^{N+1} \frac{1}{i} \]

simplifies

\[ \frac{T(N)}{N+1} = O(\log N) \]

\[ T(N) = N \log N \]

Wrapping up sorting

- consider three elements
  - a, b, c

- If we want to sort these three, what possible ordering can there be?
comparisons

- if we compare a & b
  - left if a < b
  - right if b < a

- what are subtrees?

Decision Tree

- decision trees portrays the comparisons made by some algorithm

- can imagine a tree for quick sort

- we can use it represent any comparison based sorting algorithm
so let's discuss some tree ideas, and apply them back to sorting routines

Lemma 1

- A binary tree of depth $d$ has at most $2^d$ leaves

- can you prove this??
Lemma 2

- A binary tree with $L$ leaves must have a depth of at least $\log(L)$

- proof should be obvious

Lemma 3

- Any sorting algorithm that uses only comparisons requires at least $\log(N!)$ comparisons in the worst case

- since $N!$ ordering possibilities
Theorem

- any sorting algorithm that only uses comparisons requires $\Theta(N \log N)$ comparisons

- can you show this?

- $\log(N!) = \log (N \times N-1 \times N-2 \times N-3 \ldots)$
  - log of product = log of sums
- $\log(N!) = \log(N) + \log(N-1) + \log(N-2) \ldots$
- (drop $n/2$ terms) $\geq \log(N) + \ldots + \log(N/2)$
- $\geq n/2 \log(N/2) = \Theta N \log N$
so how to get past the N log N barrier ??

more information

- if you have extra information you can break the n log n barrier
- knowing the range, we can use other sorts
  - bucket sort
External Sorts

- say you can only handle 100 items at a time
- need to sort 1000

- general case: many times the number of instances to sort will not fit into memory

- Strategy:
  - any ideas?

switch gears

- let's switch gears for a second

- Question:
  - I want to be able to give you a bunch of items and you should say which set the items belong to (given this information)
    - should be able to add items to sets
    - should be able to lookup an item's set
  - I want to be able to do it quickly

  - Example: search results: want to be able to tell you what broad categories a search results can be divided by
any implementation ideas?

runtime?

equivalence

A relationship R is defined on a set, if for every pair (a,b) we can answer aRb as true or false.

Equivalence relationship:
1. reflexive
   1. aRa for any A
2. symmetric
   1. if aRb then bRa
3. transitive
   1. if aRb and bRc then ... aRc
- one implementation idea:
  - use large matrix and mark if relationship exist

- let's do a quick example of a bunch of join's using a matrix
### Equivalence class

- an equivalence class of an element \( a \) in \( S \) is the subset of \( S \) that contains all elements related to \( a \)

- what are the equivalence classes in the previous example?

### online vs offline

- some DS operations can be online some offline

- Offline:
  - get all information, and then can process

- Online:
  - need to deal with the information before continuing

- Example:
  - paper exam vs oral exam
Equivalence Class ADT

- **Union(a,b)**
  - merges 2 equivalence classes

- **Find(a)**
  - retrieves equivalence class containing a

Analyzing Equivalence classes

- when doing the analysis here we will be interested in series of M operations

- lets go to implementation
  - any ideas?
Arrays

- use an array and store name of class at each position

- example of union

- what is the running time
  - find?
  - union?

running times

- so find we can do in $O(1)$

- merge, worst could be $O(N)$
  - for $M$ merges on $N$ items
    - $O(NM)$
    - $M$ can’t exceed $N$
    - so $O(N^2)$
linked lists

- each member will be in its own list
- merge?
- find?

- will come back to this next week

Review time

- Lets start a general review
- make sure you are familiar with everything covered so far
- address some of the questions seen
runtimes

- Runtimes are a rough way of judging algorithms
- what is Big-O
- why classes of functions?

Recursion

- understanding recursion
- tail recursion
- when to use
- when not to use
what is a DS?

what are ADT?

**LIST**

what is the list ADT?
- operations?
- runtimes?
Lists
- arrays
- linked lists

queues
- what is a the Queue ADT?
  - operations?
  - runtimes?

- Priority queue ADT?
  - operations?
  - runtimes?
how do the number of links affect the linked list class?
- single vs double linked

- what if we added a mid point link
- what is we added a bunch of others

Trees

what is the Tree ADT?
- operations
- runtimes?
Trees

- complete trees
- binary trees
- BST
- Balanced BST
  - AVL
  - Red-black
- lazy deletion
- tree traversal algorithms
- expression trees
- B+ trees
- huffman trees

heaps

- what are heaps?
- what DS being implemented?
hashtable

- what is the hash table ADT?
  - operations?
  - runtimes?

- what issues need to be dealt with during operations?
  - why not issue with Lists?
- Extendible hashing

sorting

- basic sorts
  - bubble
  - insert
  - selection
  - random
- better ones
  - heap sort
- Better yet
  - mergesort
  - quicksort
Sample Exam

- let's do some sample questions together

Sample 1

- An algorithm takes 0.5 ms for input of size 100. How long will it take for input of size 500 if the runtime is of the following (assume low order terms are negligible). Show work.
  a) Linear
  b) $O(N \log N)$
  c) Quadratic
  d) Cubic
Sample 2

- Programs A and B are analyzed and found to have worst-case running times no greater than $150N \log N$ and $N^2$, respectively. Answer the following if possible:
  
  a) Which program has the better guarantee on the running time for large values of $N$ ($N > 10,000$)?
  
  b) Which program has the better guarantee on the running time for small values of $N$ ($N < 100$)

Sample 3

- Suppose we want to add an operation FINDKth to our toolbox of binary tree operations.
- This operation returns the kth smallest item in the tree. Assume all items are distinct (no repeats)
- Explain how you would modify the binary tree structure to support FINDKth.
- This operation also must run in $O(\log N)$ average time, without sacrificing the time bound on any other operation currently in the tree.
Sample 4

- Show the result of inserting the keys 1011-1101, 0000-0010, 1001-1011, 1011-1110, 0111-1111, 0101-0001, 1001-0110, 0000-1011, 1100-1111, 1001-1110, 1101-1011, 0010-1011, 0110-0001, 1111-0000, 01110-1111 into an initially empty B-tree structure with M=3, L=3. These are 8 character keys, the dash makes them easier to read.

- any other questions ??
Next

- do all the reading
- the exam is open brain/notes/book closed
general internet
- will post when test is ready

- please email/aim but will only answer up to test time

- Good luck