CS 361, Lecture 15

Jared Saia
University of New Mexico

Outline

- Midterm
- Quicksort

Midterm

- 5 questions, 20 points each
- Hard but fair
- There will be some time pressure, so make sure you can e.g. solve recurrences both quickly and correctly.
- I expect a class mean of between 50 :(and 65 :) points

Question 1

Collection of true/false questions and short answer on:

- sorting algorithms (mergesort, heapsort, bubblesort)
- heaps (heights, number of nodes, heap algorithms, where is the max?, where is the min?)
- theta notation (i give you a bunch of functions and ask you to give me the simplest possible theta notation for each)

Question 2

- A question on annihilators and recurrence trees (like problems 1-3 of hw)
- You'll need to know the formula for sum of an infinite convergent series

Question 3

- A question on using annihilators to solve a recurrence with both homogeneous and non-homogeneous parts

Question 4

- A question on writing recurrences for both the result of a function and the time cost of the function, and solving both of these recurrences using annihilators

Question 5

- A question asking you to prove the correctness of an algorithm using loop invariants
- I'll give you the loop invariant and ask you to prove initialization, maintenance and termination
- Will be for an algorithm on heaps

Questions

- Any questions?

Review Session

There will be a review session Today at 1pm

Other Review Session Options:

- Today at 5pm
- Today at 7pm
- Tomorrow at 3pm
- Tomorrow at 5pm

In-Class Exercise

- Imagine you have a min-heap with the following operations defined and taking $O(\log n)$:
  - (key,data) Heap-Extract-Min (A)
  - Heap-Insert (A,key,data)
- Now assume you're given k sorted lists, each of length n/k
- Use this min-heap to give a $O(n \log k)$ algorithm for merging these k lists into one sorted list of size n.

In-Class Exercise

- Q1: What is the high level idea for solving this problem?
- Q2: What is the pseudocode for solving the problem?
- Q3: What is the runtime analysis?
- Q4: What would be an appropriate loop invariant for proving correctness of the algorithm?

In-Class Exercise

```
KMerge (int arrList[][], int n, int k)\{
int arrI[] = new int[k];
int arrRes[] = new int[n];
for (i=1;i<= k;i++){
  Heap-Insert (A,arrList[i][1],i);
  arrI[i] = 1;
}
for (i=1;i<=n;i++){
  (key,listNum) = Heap-Extract-Min (A);

   arrRes[i] = key;
   arrI[listNum]++;
   if (arrI[lisNum] <= n/k){
     Heap-Insert (A,arrList[listNum][arrI[listNum]],
                  arrI[listNum]);
}
```


Takeaway

- Can use heaps to merge k lists in $O(n \log k)$ time
- Heaps are a simple but very handy data structure for solving lots of problems

Quicksort

- Based on divide and conquer strategy
- Worst case is $\Theta(n^2)$
- Expected running time is $\Theta(n \log n)$
- An In-place sorting algorithm
- Almost always the fastest sorting algorithm

Quicksort

"To conquer the enemy without resorting to war is the most desirable. The highest form of generalship is to conquer the enemy by strategy" - Sun Tzu, The Art of War

- **Divide:** Pick some element A[q] of the array A and partition A into two arrays A_1 and A_2 such that every element in A_1 is $\leq$ A[q], and every element in A_2 is $>$ A[p]
- **Conquer:** Recursively sort A_1 and A_2
- **Combine:** A_1 concatenated with $A[q]$ concatenated with A_2 is now the sorted version of A

The Algorithm

```
//PRE: A is the array to be sorted, p>=1,  and r is <= the size of A
//POST: A[p..r] is in sorted order
Quicksort (A,p,r){
  if (p<r){
    q = Partition (A,p,r);
    Quicksort (A,p,q-1);
    Quicksort (A,q+1,r);
}
```


Partition

```
//PRE: A is the array to be partitioned, p>=1 and r <= size of A
//POST: A[]
Partition (A,p,r){
  x = A[r];
  i = p-1;
  for (j=p;j<=r-1;j++){
    if (A[j]<=x){
      i++;
      exchange A[i] and A[j];
  }
  exchange A[i+1] and A[r];
  return i+1;
}
```

Correctness

Basic idea: The array is partitioned into four regions, x is the pivot

- Region 1: Region that is less than or equal to x
- Region 2: Region that is greater than x
- Region 3: Unprocessed region
- Region 4: Region that contains x only

Region 1 and 2 are growing and Region 3 is shrinking

Correctness

Basic idea: The array is partitioned into four regions, x is the pivot

- Region 1: Region that is less than or equal to x (between p and i)
- Region 2: Region that is greater than x (between $i+1$ and $j-1$)
- Region 3: Unprocessed region (between j and $r-1$)
- Region 4: Region that contains x only (r)

Region 1 and 2 are growing and Region 3 is shrinking

Example

- Consider the array (2 6 4 1 5 3)

Loop Invariant

At the beginning of each iteration of the for loop, for any index k:

1. If $p \leq k \leq i$ then $A[k] \leq x$
2. If $i+1 \leq k \leq j-1$ then $A[k] > x$
3. If $k = r$ then $A[k] = x$

In Class Exercise

- Show Initialization for this loop invariant
- Show Termination for this loop invariant
- Show Maintenance for this loop invariant:
  - Show Maintenance when $A[j] > x$
  - Show Maintenance when $A[j] \leq x$

Scratch Space

Scratch Space

Analysis

- The function Partition takes $O(n)$ time. Why?
- Q: What is the runtime of Quicksort?
- A: It depends on the size of the two lists in the recursive calls

Best Case

- In the best case, the partition always splits the original list into two lists of half the size
- Then we have the recurrence $T(n) = 2T(n/2) + \Theta(n)$
- This is the same recurrence as for mergesort and its solution is $T(n) = O(n \log n)$

Worst Case

- In the worst case, the partition always splits the original list into a singleton element and the remaining list
- Then we have the recurrence $T(n) = T(n-1) + T(1) + \Theta(n)$, which is the same as $T(n) = T(n-1) + \Theta(n)$
- The solution to this recurrence is $T(n) = O(n^2)$. Why?

Todo

- Read Chapter 7
- Finish HW
- Study for Midterm!