CS 361, Lecture 17

Jared Saia
University of New Mexico

Outline

- Randomized Quicksort
- Analysis

Midterm

- Mean was 68.7/100 (better than I expected)
- Distribution:
- 90-100 4
- 80-89 8
- 70-79 10
- 60-69 5
- 50-59 3
- 40-49 4
- 0-40 3

Problem Areas

- Loop invariant problem
- Asymptotic Analysis (second part of problem 1)

Midterm

- Grades (Roughly):
- 80-100 A
- 70-80 B
- 60-70 C
- 50-60 D
- 0-50 F

Midterm

- These grades are approximate
- But, if you got 55 or below on the midterm, come see me in my office hours
- If you do much better on the final than on the midterm, I will weight the final more heavily

Questions?

- Any Questions?

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[p..r] is the array to be partitioned, p>=1 and r <= size
//     of A, A[r] is the pivot element
//POST: Let A' be the array A after the function is run.  Then
//      A'[p..r] contains the same elements as A[p..r].  Further,
//      all elements in A'[p..res-1] are <= A[r], A'[res] = A[r],
//      and all elements in A'[res+1..r] are > A[r]
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?

Average Case Intuition

- Even if the recurrence tree is somewhat unbalanced, Quicksort does well
- Imagine we always have a 9-to-1 split
- Then we get the recurrence $T(n) \leq T(9n/10) + T(n/10) + cn$
- Solving this recurrence (with annihilators or recursion tree) gives $T(n) = \Theta(n \log n)$

Randomized Quick-Sort

- We'd like to ensure that we get reasonably good splits reasonably quickly
- Q: How do we ensure that we "usually" get good splits? How can we ensure this even for worst case inputs?
- A: We use randomization.

R-Partition

```
//PRE: A[p..r] is the array to be partitioned, p>=1 and r <= size
//     of A
//POST: Let A' be the array A after the function is run.  Then
//      A'[p..r] contains the same elements as A[p..r].  Further,
//      all elements in A'[p..res-1] are <= A[i], A'[res] = A[i],
//      and all elements in A'[res+1..r] are > A[i], where i is
//      a random number between $p$ and $r$.
R-Partition (A,p,r){
  i = Random(p,r);
  exchange A[r] and A[i];
  return Partition(A,p,r);
}
```

Randomized Quicksort

```
//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
R-Quicksort (A,p,r){
  if (p<r){
    q = R-Partition (A,p,r);
    R-Quicksort (A,p,q-1);
    R-Quicksort (A,q+1,r);
}
```


Todo

- Finish Chapter 7