CS 380
ALGORITHM DESIGN
AND ANALYSIS

Lecture 5: Quicksort Introduction
Best, worst-case analysis
Text Reference: Chapter 7

Sorting

• What's the running time for:
  • Insertion Sort
  • Merge Sort
• Which of these algorithms sort in place?
Quicksort

- The Basic version of quicksort was invented by C. A. R. Hoare in 1960
- Divide and Conquer algorithm
- In practice, it is the fastest in-place sorting algorithm

https://en.wikipedia.org/wiki/Quicksort

Divide and Conquer

- **Divide**: Partition the array into two subarrays around a pivot \( x \) such that elements to the left are \( \leq x \) and elements to the right are \( \geq x \)

\[
\begin{array}{c}
\leq x \\
\hline
x \\
\hline
\geq x
\end{array}
\]

- **Conquer**: Recursively sort the two subarrays
- **Combine**: Trivial!

- To be effective, need good partitioning algorithm
Quicksort Pseudocode

QUICKSORT(A, p, r)

<table>
<thead>
<tr>
<th></th>
<th>Quicksort(A, p, r) // A:Array; p,r: integer indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if p &lt; r</td>
</tr>
<tr>
<td>2</td>
<td>q = Partition(A, p, r);</td>
</tr>
<tr>
<td>3</td>
<td>Quicksort(A, p, q-1);</td>
</tr>
<tr>
<td>4</td>
<td>Quicksort(A, q+1, r);</td>
</tr>
</tbody>
</table>

- What’s the call to sort the entire array?

Partitioning the Array

PARTITION(A, p, r)

<table>
<thead>
<tr>
<th></th>
<th>Partition(A,p,r) // A:Array; p,r: integer indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x = A[r]</td>
</tr>
<tr>
<td>2</td>
<td>i = p - 1</td>
</tr>
<tr>
<td>3</td>
<td>for j = p to r-1</td>
</tr>
<tr>
<td>4</td>
<td>if A[j] &lt;= x</td>
</tr>
<tr>
<td>5</td>
<td>i = i + 1</td>
</tr>
<tr>
<td>6</td>
<td>swap(A[i], A[j])</td>
</tr>
<tr>
<td>7</td>
<td>swap (A[i+1], A[r])</td>
</tr>
<tr>
<td>8</td>
<td>return i+1</td>
</tr>
</tbody>
</table>

What is significance of Partition returning the value i+1?
Correctness of Partition

- During the execution of PARTITION there are four distinct sections of the array:

  ![Diagram showing partition of array]

  - \( i \) = last value \( \leq x \)
  - \( j \) = first value that is unknown

  - \( j \) marches along
    - if \( a[j] \leq x \)
      - increment \( i \) (now \( i \) points to a value \( > x \))
      - swap \( a[i] \) and \( a[j] \)
    
  - finally: swap the first value \( > x \) with \( x \) itself and increase \( i \) (to include \( x \))

Example

- Array: 5 3 9 1 8 2 4 7
- Initial partition:
  - \( p \) = 1
  - \( i \) = 2
  - \( j \) = 3
  - \( r \) = 8

  ![Example of partition with values]

  - After partition:
    - \( p \) = 1
    - \( i \) = 2
    - \( j \) = 3
    - \( r \) = 8

  - Swap first value \( > x \) with \( x \) and increase \( i \): final partition
Example: Sequence of Function Calls

Original Array: 5 3 9 1 8 2 4 7
Calling Quicksort on A[1,8]  
Partition function returned: 6  
5 3 1 2 4 7 8 9
Calling Quicksort iteratively on A[1,5]  
Partition function returned: 4  
3 1 2 4 5 7 8 9
Calling Quicksort iteratively on A[1,3]  
Partition function returned: 2  
1 2 3 4 5 7 8 9
Calling Quicksort iteratively on A[1,1]  
Calling Quicksort iteratively on A[3,3]  
1 2 3 4 5 7 8 9
Calling Quicksort iteratively on A[5,5]  
1 2 3 4 5 7 8 9
Calling Quicksort iteratively on A[7,8]  
Partition function returned: 0  
1 2 3 4 5 7 8 9
Calling Quicksort iteratively on A[7,7]  
Calling Quicksort iteratively on A[9,8]  
1 2 3 4 5 7 8 9
Press any key to continue . . .
Quicksort: Calls, A[8], p = 1, r = 8

Q-S(1,8):

- P(1,8) = 6
- Q-S(1,5)
  - P(1,5) = 4
    - Q-S(1,3)
      - P(1,3) = 2
      - Q-S(3,3)
  - Q-S(5,5)
- Q-S(7,8)
  - P(7,8) = 8
    - Q-S(7,7)
    - Q-S(9,8)

Q-S(p,q) = Quicksort(A, p, q)
P(p,q) = Partition(A, p, q)

Note: Unlike Merge Sort, this tree is very dependent on initial array

Question for next time:
Why Q-S(9,8) and not Q(8,8)?

Exercise - Partition the Following

| 44 | 75 | 23 | 43 | 55 | 12 | 64 | 77 | 33 | 41 |

CS380: Algorithm Design and Analysis
Analysis of Partition

- What is the running time of PARTITION?

\[
\text{Partition}(A, p, r) \quad // \quad A:\text{Array}; \quad p, r: \text{integer indexes}
\]

1. \( x = A[r] \)
2. \( i = p - 1 \)
3. \( \text{for } j = p \text{ to } r-1 \)
4. \( \quad \text{if } A[j] \leq x \)
5. \( \quad i = i + 1 \)
6. \( \quad \text{swap}(A[i], A[j]) \)
7. \( \quad \text{swap}(A[i+1], A[r]) \)
8. \( \quad \text{return } i+1 \)

Exercise

- Sort the following array using quicksort, and produce the corresponding tree diagram:

\[
\begin{array}{ccccc}
3 & 4 & 2 & 5 & 1 \\
\end{array}
\]
Performance of Quicksort

- What does the performance of quicksort depend on?
- What would give us the best case?
- Are there better strategies for choosing the pivot?

Best Case of Quicksort

- The total partitioning on each level is \( O(n) \), and it takes \( \lg n \) levels of perfect partitions to get to single element subproblems. When we are down to single elements, the problems are sorted. Thus the total time in the best case is \( O(n \lg n) \).
Worst Case of Quick Sort

• Suppose our pivot element splits the array as unequally as possible. Thus instead of n/2 elements in the larger half, we get zero, meaning that the pivot element is the biggest element in the array. Now we have n-1 levels, instead of \( \lg n \).
• \( T(n) = T(n-1) + O(n), \) which has a solution of \( O(n^2) \)

Next time

• Average case analysis for Quicksort
• Alternative Partitioning Schemes
• Alternative Pivot Choices