

CSC 301: Analysis of Algorithms

Lecture 16: Quick Sort and Sorting Basics

William Rebelsky
[rebelskyw]

Grinnell College

Monday 26th February, 2024

Reminders

- Homework 4 is due Wednesday
- Quiz 4 is due Tonight

Schedule Until Spring Break

- This week (26 Feb-01 Mar): Sorting in Linear Time (will be included on midterm)
 - Homework 5 will be released Wed 28 Feb. I'd recommend doing it before the midterm, but it will not be due until 13 Mar
 - Quiz 5 will still be due on Monday Mar 4
- The week after (04 Mar-08 Mar): Monday will be review. Wednesday I will release the Midterm, Wednesday and Friday I will be in the classroom to answer any questions you have about the Midterm.
 - The midterm will be closed Internet (no using google, chatgpt, etc)
 - The midterm will be individual work (no collaborating)
 - It will be due Wednesday Mar 13, but I believe you should be able to finish it by Friday the 8th
- The following week (11 Mar-15 Mar): Bias in Algorithms. Please do the readings (see course website).
 - The Essay Assignment will be released and due after break

Table of Contents

1 Sorting Basics

2 Quick Sort

3 Performance

4 Practice

Sorting Basics

Given an Array of size n , return the same elements in sorted order (generally increasing). A comparison sorting algorithm cannot perform better than $\Theta(n \log n)$ on average.

Theorem 8.1

Any comparison sort algorithm requires $\Omega(n \lg n)$ in the worst case

- It suffices to determine the height of a decision tree where each permutation appears as a reachable leaf.
- Let the decision tree have height h , l reachable leaves, corresponding to comparison sort on n elements
- $n!$ possible permutations appears on at least one leaf, so $n! \leq l$
- Binary tree has no more than 2^h leaves
- $n! \leq l \leq 2^h \rightarrow h \geq \lg(n!) = \Omega(n \lg n)$

Standard Algorithms

Algorithm	Best Case	Expected	Worst Case	Method
Merge sort	$n \log n$	$n \log n$	$n \log n$	Merging
Insertion sort	n	n^2	n^2	Insertion
Bubble sort	n	n^2	n^2	Exchanging

Each algorithm has advantages and disadvantages (stability, memory requirements, etc). Today we will be looking at Quick Sort

Table of Contents

1 Sorting Basics

2 Quick Sort

3 Performance

4 Practice

Quick Sort

- Worst case run time: $\Theta(n^2)$
- Expected run time: $\Theta(n \log n)$, with smaller constants than merge sort
- Sort in place
- Divide and Conquer Method

Quick Sort

- **Divide** by partitioning A into two subarrays around a pivot q such that each element in the left subarray is less than or equal to q and each element in the right subarray is greater than or equal to q .
- **Conquer** recursively call on sub arrays (note that subarrays do not include the pivot)
- **Combine** do nothing, it was sorted in place.

Quick Sort Pseudo Code

Algorithm 1: Quicksort(A, p, r)

Input: Array A , left index p , right index r **Output:** Sorted Array A

```
1 if  $p < r$  then
2   |    $q = \text{Partition}(A, p, r)$ 
3   |   Quicksort( $A, p, q-1$ )
4   |   Quicksort( $A, p, q+1$ )
```

Partition

Algorithm 2: Partition(A, p, r)

Input: Array A , left index p , right index r **Output:** new index of pivot

```
1  $x = A[r]$ 
2  $i = p - 1$ 
3 for  $j = p$  to  $r - 1$  do
4   if  $A[j] \leq x$  then
5      $i = i + 1$ 
6     exchange  $A[i]$  with  $A[j]$ 
7 exchange  $A[i + 1]$  with  $A[r]$ 
8 return  $i + 1$ 
```

Proof of correctness can be found in the text.

Partition Example

$$A = \{2, 8, 7, 1, 3, 5, 6, 4\}$$

Table of Contents

1 Sorting Basics

2 Quick Sort

3 Performance

4 Practice

Worst Case Performance of QuickSort

- Worst case: partitioning produces one subproblem of size $n-1$ and one of size 0
- Partitioning costs $\Theta(n)$
- Run time:

$$\begin{aligned}T(n) &= T(n-1) + T(0) + \Theta(n) \\&= T(n-1) + \Theta(n) \\&= \Theta(n^2)\end{aligned}\tag{1}$$

Best Case Performance of QuickSort

- Best case: partitioning produces two subproblems of size no more than $n/2$.
- Partitioning costs $\Theta(n)$
- Run time:

$$\begin{aligned}T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n)\end{aligned}\tag{2}$$

Balanced Performance of QuickSort

- Assume the partition produces a consistent split in each step (e.g. 9:1).
- Partitioning costs $\Theta(n)$
- Using the tree, height is $\lg(n)$
- We still have $O(n \lg n)$

Expected Performance of QuickSort

- Some splits will be good, some will be bad
- In the average case, the good and bad splits are distributed randomly.
- If we assume they alternate, can easily show that it looks like the balanced case (with a larger constant hidden in the O notation)
- We still have $O(n \lg n)$

Randomized QuickSort

- Instead of always choosing the last element, start by selecting a random element and swapping it with the last element
- On average expect a reasonable split
- Same expected runtime, narrower distribution

Table of Contents

1 Sorting Basics

2 Quick Sort

3 Performance

4 Practice

QuickSort Practice

- Show the entire computation of (non-randomized) QuickSort applied to the array $A = \{2, 8, 7, 1, 3, 5, 6, 4\}$. (Keep track of p, r, i, j in each step)
- What is the run time of Quicksort when A contains distinct elements sorted in decreasing order?
- Why do I claim partitioning is $\Theta(n)$
- Why does it not matter what the split is as long as it is consistent when doing runtime analysis of quicksort?