

# CSCB63 – Design and Analysis of Data Structures

Anya Tafliovich<sup>1</sup>

---

<sup>1</sup>with huge thanks to Anna Bretscher and Albert Lai

## introduction

Today we begin studying a different way to look at complexity of algorithms.

So far we saw

- worst case analysis of individual operations
- amortised analysis over a sequence of operations

Today we add

- expected running time analysis
- need to review our probabilities!

## quicksort: idea

A divide-and-conquer algorithm.

1. if  $|A| < 2$ : stop
2. pick an element  $x$ , called “pivot”
3. partition  $A$  into:
  - $L$ : array of elements  $\leq x$
  - $G$ : array of elements  $> x$

( $\Theta(|A|)$  time;  $|A| - 1$  comparisons against pivot)
4. recurse: sort  $L$
5. recurse: sort  $G$
6. concatenate sorted  $L$ , sorted  $G$

We write  $L$ ,  $G$  as new arrays to show the idea.

In reality, “partition” is done in-place in  $A$ , so  $L$  occupies the left side,  $G$  occupies the right side, and there is no “concatenate” step.

## quicksort: algorithm

```
quicksort(A, p, r):
```

```
0. if p < r:
```

```
1.   q = partition(A, p, r)
```

```
2.   quicksort(A, p, q-1)
```

```
3.   quicksort(A, q+1, r)
```

```
partition(A, p, r):
```

```
0. x := A[r]    // pivot
```

```
1. i := p-1
```

```
2. for j in p, ..., r-1:
```

```
3.   if A[j] <= x:
```

```
4.     i := i + 1
```

```
5.     exchange A[i] with A[j]
```

```
6. exchange A[i+1] with A[r]
```

```
7. return i + 1
```

## quicksort: example

Trace quicksort on the array

$$A = [2, 8, 7, 1, 3, 5, 6, 4]$$

## quicksort: correctness

The loop invariants of partition:

- $A[k] \leq x$  for  $p \leq k \leq i$
- $A[k] > x$  for  $i + 1 \leq k \leq j - 1$
- $A[k] = x$  for  $k = r$

Proof: exercise.

## quicksort: complexity

- running time of partition is

$$\Theta(n) \text{ where } n = r - p + 1$$

- running time depends on partitioning
  - balanced partition produces better time
- worst case when:
  - $n - 1$  elements in one partition and 1 element in the other
- best case when:
  - $n/2$  elements in one partition and  $n/2$  elements in the other
- average case:
  - is much closer to best case than to the worst case!

## randomised quicksort: idea

- remove “bias” in the input that may cause too many unbalanced partitions
- by selecting pivots at random
- we expect the split to be reasonably well balanced on average
- technique called random sampling



## randomised quicksort: algorithm

r-quicksort(A, p, r):

0. if  $p < r$ :
1.     $q = \text{r-partition}(A, p, r)$
2.     $\text{r-quicksort}(A, p, q-1)$
3.     $\text{r-quicksort}(A, q+1, r)$

r-partition(A, p, r):

0. exchange  $A[r]$  with  $A[\text{random}(p, r)]$
1.  $x := A[r]$             // pivot is now random from  $A[p..r]$
2.  $i := p-1$
3. for  $j$  in  $p, \dots, r-1$ :
4.    if  $A[j] \leq x$ :
5.         $i := i + 1$
6.        exchange  $A[i]$  with  $A[j]$
7. exchange  $A[i+1]$  with  $A[r]$
8. return  $i + 1$

## randomised quicksort: complexity

Idea:

- partition selects a pivot
- pivot never included in any future recursive calls
- at most  $n$  calls to partition over entire execution of quicksort
- partition is:  $\mathcal{O}(1)$  plus for-loop
- focus on line 4 in for-loop
- can count number of times line 4 runs
- will give us bound on time spent in the for-loop over entire execution of quicksort

Formally:

- quicksort is  $\mathcal{O}(n + X)$  where
- $n$  is the length of array  $A$  and
- $X$  is the total number of comparisons on line 4 over entire execution

## randomised quicksort: complexity

Notice:

- each pair of elements is compared at most once:
- elements are compared only to the pivot
- pivot is never used after call to partition

Let:

- elements of  $A$ :  $\{z_1, z_2, \dots, z_n\}$  where  $z_i$  is the  $i^{th}$  smallest element of  $A$
- $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$  be set of elements between  $z_i$  and  $z_j$
- $X_{ij} = I\{z_i \text{ is compared to } z_j\}$  be indicator random variable

## stats flashback: indicator variable

An indicator random variable  $X$  is

$$X_i = \begin{cases} 1 & \text{if an event } i \text{ occurs} \\ 0 & \text{if it does not} \end{cases}$$

## randomised quicksort: expected time

Total number of comparisons:

$$\begin{aligned} E(X) &= E \left( \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij} \right) \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n E(X_{ij}) \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \Pr(z_i \text{ is compared to } z_j) \end{aligned}$$

## randomised quicksort: expected time

When is  $z_i$  compared to  $z_j$ ?

In  $z_i < \dots < x < \dots < z_j$ ,

- if  $x$  becomes pivot first,  $z_i$  is not compared to  $z_j$   
( $z_i$  moved to left partition,  $z_j$  moved to right partition)
- if  $z_i$  becomes pivot first,  $z_i$  is compared to  $z_j$
- if  $z_j$  becomes pivot first,  $z_i$  is compared to  $z_j$

## randomised quicksort: expected time

$$\begin{aligned}& Pr(z_i \text{ is compared to } z_j) \\&= Pr(z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}) \\&= Pr(z_i \text{ is the first pivot chosen from } Z_{ij}) \\&+ Pr(z_j \text{ is the first pivot chosen from } Z_{ij}) \\&= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\&= \frac{2}{j-i+1}\end{aligned}$$

## randomised quicksort: expected time

Total number of comparisons:

$$\begin{aligned} E(X) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \Pr(z_i \text{ is compared to } z_j) \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\ &< 2 * \sum_{i=1}^{n-1} \sum_{k=1}^n \frac{1}{k} \\ &< 2 * \sum_{i=1}^{n-1} \ln n \\ &= 2(n-1) \ln n \\ &\in \mathcal{O}(n \log n) \end{aligned}$$