CSC263 Week 6

Announcements

PS2 marks out today.

Class average 85% !

Midterm tomorrow evening, 8-9pm EX100

Don't forget to bring your ID!

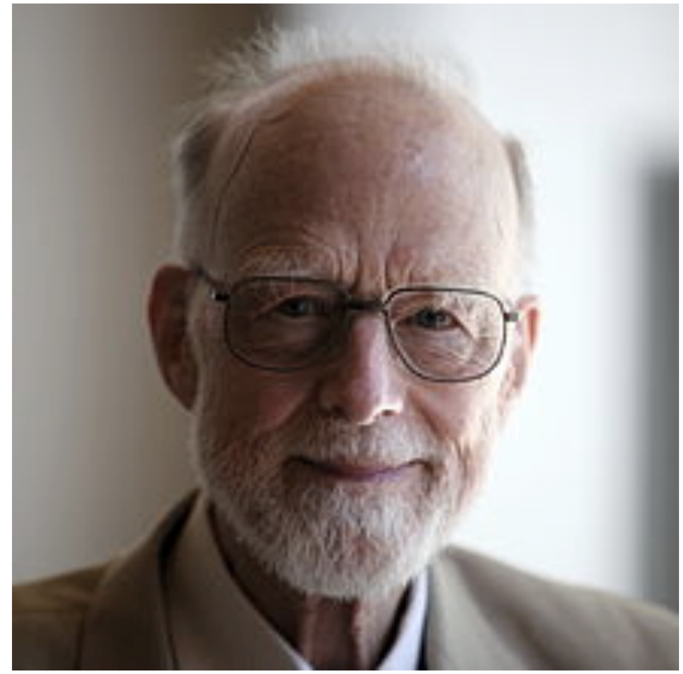
This week

- ➔QuickSort and analysis
- ➔Randomized QuickSort
- ➔Randomized algorithms in general

QuickSort

Background

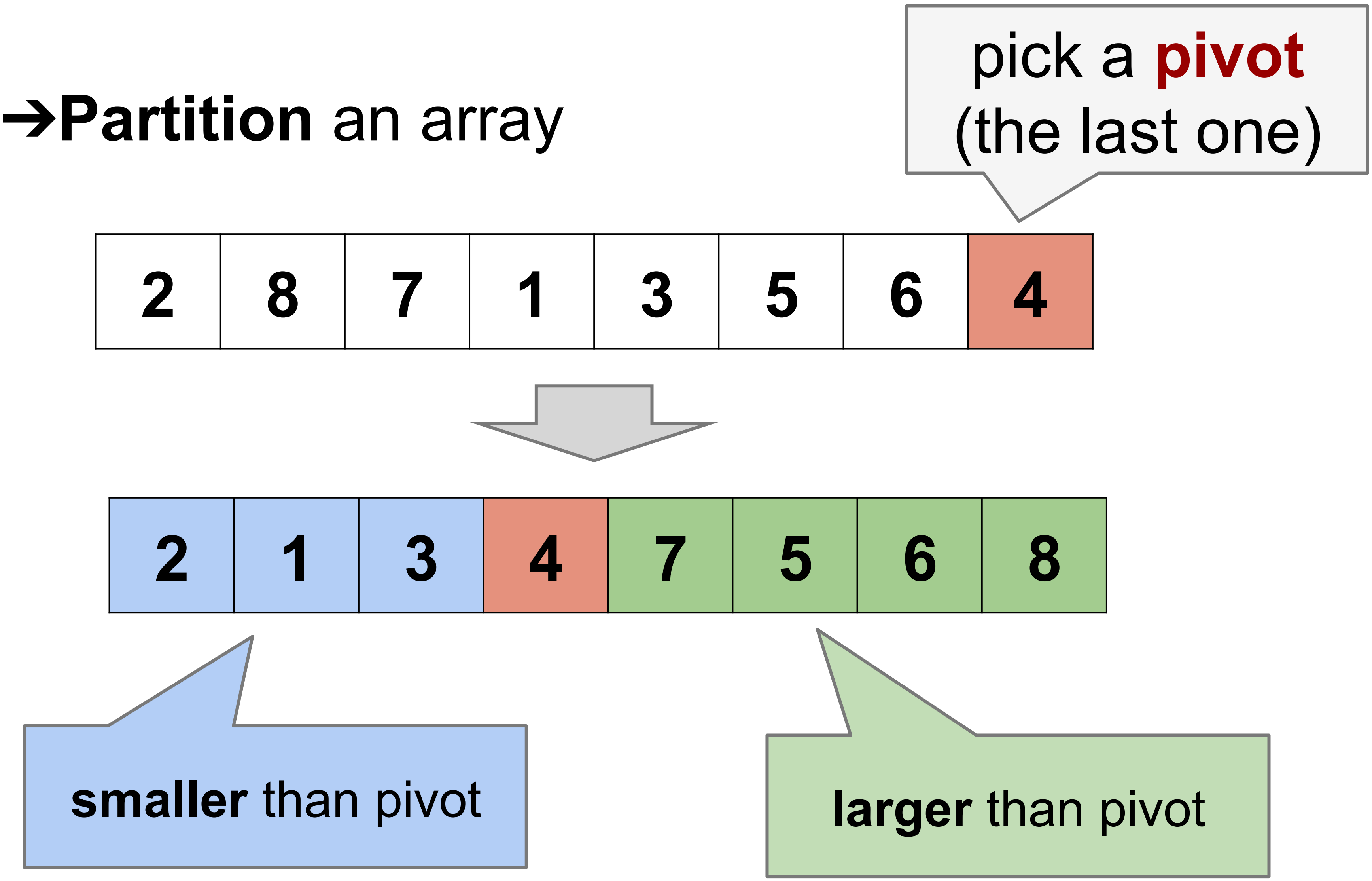
Invented by **Tony Hoare** in 1960

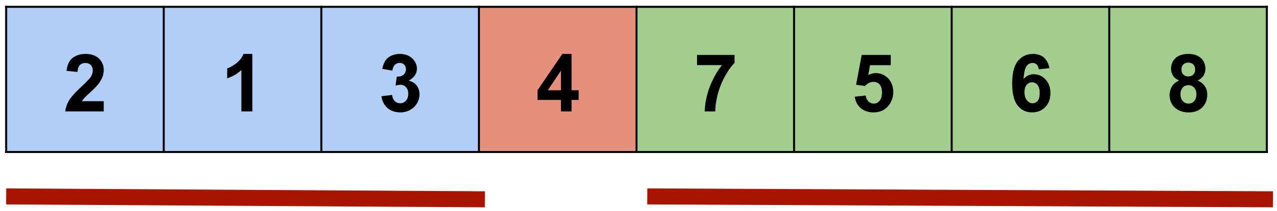


Very commonly used sorting algorithm. When **implemented well**, can be about 2-3 times faster than **merge sort** and **heapsort.**

QuickSort: the idea

➔**Partition** an array





Recursively partition the sub-arrays **before** and **after** the pivot.

Base case:

1 **sorted**

Read textbook Chapter 7 for details of the Partition operation

Worst-case Analysis of QuickSort

T(n): the total number of **comparisons** made

For simplicity, assume all elements are distinct

A								
	2	1	3	4	7	5	6	8

Claim 1. Each element in **A** can be chosen as **pivot** **at most once**.

A pivot never goes into a sub-array on which a recursive call is made.

Claim 2. Elements are **only** compared to **pivots**.

That’s what partition is all about -- comparing with pivot.

A	**2**	**1**	**3**	**4**	**7**	**5**	**6**	**8**

Claim 3. Every **pair** (a, b) in A are compared with each other **at most once**.

The only possible one happens when **a or b** is chosen as a **pivot** and the other is compared to it; after being the pivot, the pivot one will be out of the market and never compare with anyone anymore.

So, the total number of **comparisons** is **no more than** the **total number of pairs**.

So, the total number of **comparisons** is **no more than** the **total number of pairs**.

$$T(n) \leq \binom{n}{2} = \frac{n(n-1)}{2}$$

$$T(n) \in \mathcal{O}(n^2)$$

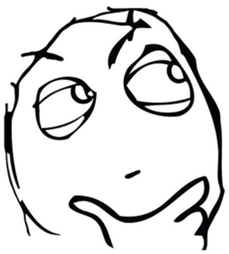
$$\text{Next, show } T(n) \in \Omega(n^2)$$

Show $T(n) \in \Omega(n^2)$

i.e., the **worst-case** running time is **lower-bounded** by some cn^2

Just find **one input** for which the running time is at least cn^2

so, just find **one input** for which the running time is some **cn²**



i.e., find one input that results in **awful** partitions (everything on one side).

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Remember that we always pick the last one as pivot.

IRONY:

The worst input for QuickSort is an already sorted array.

Calculate the number of comparisons

1	2	3	4	5	6	7	8

Choose pivot **A[n]**, then **n-1** comparisons

Recurse to subarray, pivot **A[n-1]**, then **n-2** comps

Recursive to subarray, pivot **A[n-2]**, then **n-3** comps

. . .

Total # of comps:

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$$

So, the worst-case runtime

$$T(n) \geq \frac{n(n-1)}{2}$$

$$T(n) \in \Omega(n^2)$$

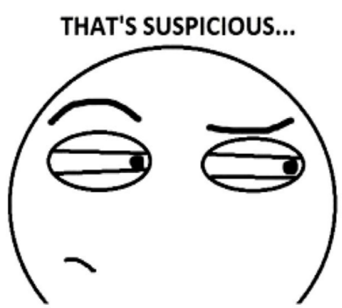
already shown $T(n) \in \mathcal{O}(n^2)$

so, $T(n) \in \Theta(n^2)$

$$T(n) \in \Theta(n^2)$$

What other sorting algorithms have **n²** worst-case running time?

(The stupidest) Bubble Sort!



Is QuickSort really "quick" ?

Yes, in **average-case.**

Average-case Analysis of QuickSort

O(n log n)

Average over what?

Sample space and input distribution

All **permutations** of array **[1, 2, …, n]**, and each permutation appears **equally likely**.

Not the only choice of sample space, but it is a representative one.

What to compute?

Let **X** be the random variable representing the **number of comparisons** performed on a sample array drawn from the sample space.

We want to compute **E[X]**.

An indicator random variable!

array is a permutation of [1, 2, …, n]

$$X_{ij} = \begin{cases} 1 & \text{if the values } i \text{ and } j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

So the total number of comparisons:

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

sum over all possible pairs

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

$$E[X] = E\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}]$$

because IRV

Just need to figure this out!

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \boxed{\Pr(i \text{ and } j \text{ are compared})}$$

$$\Pr(i \text{ and } j \text{ are compared})$$

Note: i < j

Think about the sorted sub-sequence

$$Z_{ij} : i, i+1, \ldots, j$$

A Clever Claim: ***i*** and ***j*** are compared **if and only if**, among all elements in ***Z_{ij}***, the first element to be picked as a **pivot** is **either *i* or *j***.

$$Z_{ij} : i, i+1, \ldots, j$$

Claim: ***i*** and ***j*** are compared **if and only if**, among all elements in ***Z*****_ij_**, the first element to be picked as a **pivot** is **either *i* or *j***.

Proof:

The "**only if**": suppose the first one picked as pivot as some k that is between i and j,...
then i and j will be separated into **different partitions** and will never meet each other.

The "**if**": if ***i*** is chosen as pivot (the **first one** among ***Z*****_ij_**), then ***j*** will be compared to pivot ***i*** for sure, because nobody could have possibly separated them yet!

Similar argument for first choosing j

$$Z_{ij} : i, i+1, \ldots, j$$

Claim: ***i*** and ***j*** are compared **if and only if**, among all elements in ***Z_{ij}***, the first element to be picked as a **pivot** is **either *i* or *j***.

$$\Pr(i \text{ and } j \text{ are compared})$$

$$= \Pr(i \text{ or } j \text{ is the first among } Z_{ij} \text{ chosen as pivot})$$

$$= \frac{2}{j-i+1}$$

There are ***j-i+1*** numbers in Z_{ij}, and each of them is **equally likely** to be chosen as the first pivot.

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

$$E[X] = E\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}]$$

We have figured this out!

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \boxed{\Pr(i \text{ and } j \text{ are compared})}$$

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr(i \text{ and } j \text{ are compared})$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

$$\in \mathcal{O}(n \log n)$$

Analysis Over!

Something close to

$$n \sum_{k=1}^{n} \frac{1}{k}$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

$\leq$ 2n (1 + 1/2 + 1/3 + 1/4 + 1/5 + …. + 1/n)

$$\in \mathcal{O}(n \log n)$$

Why is (1 + 1/2 + 1/3 + 1/4 + .1/5 + …. + 1/n) ≤ log n ?

Divide sum into (log n) groups:

S1 = 1

S2 = 1/2 + 1/3

S3 = 1/4 + 1/5 + 1/6 + 1/7

S4 = 1/8 + 1/9 + 1/10 + 1/11 + 1/12 + 1/13 + 1/14 + 1/15

Each group sums to a number ≤ 1, so total sum of all groups is ≤ log n !

Summary

The worst-case runtime of Quicksort is **$\Theta(n^2)$**.

The average-case runtime is **$O(n \log n)$**.

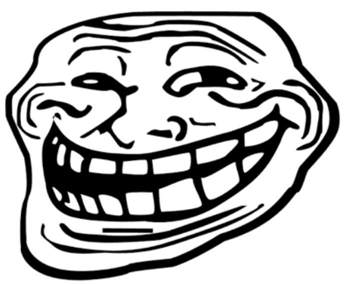
(over all permutations of [1,..,n])

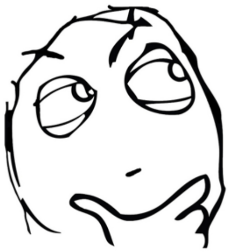
However, in real life...

Average case analysis tells us that for most inputs the runtime is O(n log n), but this is a small consolation if our input is one of the bad ones!

QuickSort(A)

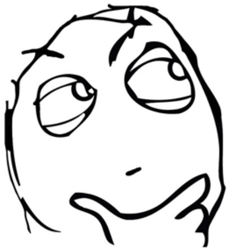
The theoretical O(nlog n) performance is in no way guaranteed in real life.





Let’s try to get around this problem by adding randomization into the algorithm itself:

```
Randomize-QuickSort(A):
    run QuickSort(A) as above
    but each time picking a random
    element in the array as a pivot
```



Let's try to get around this problem by adding randomization into the algorithm itself:

```
Randomize-QuickSort(A):
    run QuickSort(A) as above
    but each time picking a random
    element in the array as a pivot
```

- We will prove that for **any** input array of n elements, the expected time is O(n log n)
- This is called a **worst-case expected time bound**
- We no longer assume any special properties of the input

Worst-case Expected Runtime of Randomized QuickSort

O(n log n)

What to compute?

Let **X** be the random variable representing the **number of comparisons** performed on a sample array drawn from the sample space.

We want to compute **E[X]**.

Now the expectation is over the random choices for the pivot, and the input is fixed.

An indicator random variable!

array is a permutation of [1, 2, …, n]

$$X_{ij} = \begin{cases} 1 & \text{if the values } i \text{ and } j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

So the total number of comparisons:

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

sum over all possible pairs

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

$$E[X] = E\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}]$$

because IRV

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \boxed{\Pr(i \text{ and } j \text{ are compared})}$$

Just need to figure this out!

$$Z_{ij} : i, i+1, \ldots, j$$

Claim: ***i*** and ***j*** are compared **if and only if**, among all elements in ***Z***

$$\Pr(i \text{ and } j \text{ are compared})$$

$$= \Pr(i \text{ or } j \text{ is the first among } Z_{ij} \text{ chosen as pivot})$$

$$= \frac{2}{j-i+1}$$

There are ***j-i+1*** numbers in Z_{ij}, and each of them is **equally likely** to be chosen as the first pivot.

A Different Analysis (less clever)

T(n) is expected time to sort n elements. First pivot chooses i^{th} smallest element, all equally likely. Then:

$$T(n) \quad = \quad (n-1) + \frac{1}{n}\sum_{i=0}^{n-1}(T(i) + T(n-i-1)).$$

$$T(n) \quad = \quad (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} T(i)$$

Solving this recurrence gives T(n) ≤ O(n log n)

Randomized Algorithms

Use randomization to guarantee expected performance

We do it everyday.

Two types of randomized algorithms

"**Las Vegas**" algorithm

➔Deterministic **answer**, random **runtime**

"**Monte Carlo**" algorithm

➔Deterministic **runtime**, random **answer**

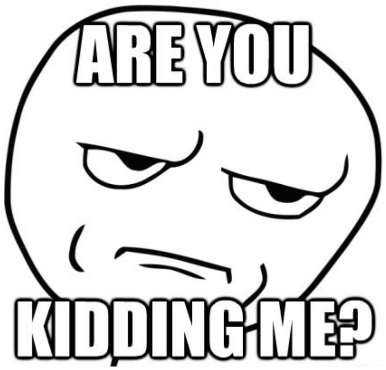
Randomized-QuickSort is a …
Las Vegas algorithm

An Example of
Monte Carlo Algorithm

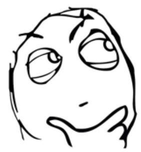
"Equality Testing"

The problem

Alice holds a binary number **x** and Bob holds **y**, decide whether **x = y**.



No kidding, what if the **size** of **x** and **y** are **10TB** each?
Alice and Bob would need to transmit $\sim 10^{14}$ bits.
Can we do better?



Why assuming x and y are of the same length?

Let n = len(x) = len(y) be the length of x and y.

Randomly choose a **prime number** $p \le n^2$,
then **$\mathrm{len}(p) \le \log_2(n^2) = 2\log_2(n)$**
then compare **(x mod p)** and **(y mod p)**
i.e., **return (x mod p) == (y mod p)**

Need to compare **at most 2log(n)** bits.

But, does it give the correct answer?

$\log_2(10^{14}) \approx 46.5$

Huge improvement on runtime!

Does it give the correct answer?

If **(x mod p) ≠ (y mod p)**, then…

Must be **x ≠ y**, our answer is correct **for sure**.

If **(x mod p) = (y mod p)**, then…

Could be **x = y or x ≠ y**, so our answer **might be** correct.

Correct with what probability?

What's the probability of a wrong answer?

Prime number theorem

In range **[1, m]**, there are roughly **m/ln(m)** prime numbers.
So in range **[1, n^2]**, there are
$n^2/\ln(n^2)$ = **$n^2/2\ln(n)$** prime numbers.

How many (**bad**) primes in **[1, n^2]** satisfy **(x mod p) = (y mod p)** even if **$x \neq y$** ?

At most n

(x mod p) = (y mod p) ⇔ $|x - y|$ is a multiple of p, i.e., p is a divisor of $|x - y|$.
$|x - y| < 2^n$ (n-bit binary #) so it has no more than n prime divisors (otherwise it will be larger than 2^n).

So...

Out of the **$n^2/2\ln(n)$** prime numbers we choose from, at most **n** of them are **bad**.

If we choose a **good** prime, the algorithm gives correct answer for sure.
If we choose a **bad** prime, the algorithm may give a wrong answer.
So the prob of wrong answer is less than

$$\frac{n}{n^2/(2\ln n)} = \frac{2\ln n}{n}$$

Error probability of our Monte Carlo algorithm

$$\Pr(\text{error}) \leq \frac{2 \ln n}{n}$$

When n = 10^{14} (10TB)
Pr(error) ≤ 0.00000000000644

Performance comparison (n = 10TB)

The **regular** algorithm **x == y**

➔ Perform 10^{14} comparisons

➔ Error probability: 0

The **Monte Carlo** algorithm **(x mod p) == (y mod p)**

➔ Perform < 100 comparisons

➔ Error probability: 0.000000000000644

If your boss says: “This error probability is too high!”

Run it **twice**: Perform < 200 comparisons

➔ Error prob squared: 0.000000000000000000000000215

Summary

Randomized algorithms

- ➔Guarantees worst-case expected performance
- ➔Make algorithm less vulnerable to malicious inputs

Monte Carlo algorithms

- ➔Gain time efficiency by sacrificing some correctness.

For more details:

Notes on Randomized Algorithms and Quicksort

posted on course webpage, lecture 6

- Also gives a good review of probability theory and computing expectations!