Design & Analysis of Algorithms

Randomized Algorithms

Prof. Wilson Rivera
Spring 2018

Example of a better randomized algorithm: QuickSort

- Runs in expected time $O(n\log(n))$.
- Worst-case runtime $O(n^2)$.
- Easier to implement than MergeSort, and the constant factors inside the O() are very small.
- In practice often more desirable.

Quicksort

We want to sort this array.

First, pick a "pivot." Do it at random.

7	6	3	5	1	2	4

random pivot!

Next, partition the array into "bigger than 5" or "less than 5"

This PARTITION step takes time O(n). (Notice that we don't sort each half). [same as in SELECT]

Arrange them like so:

L = array with things smaller than A[pivot]

R = array with things larger than A[pivot]

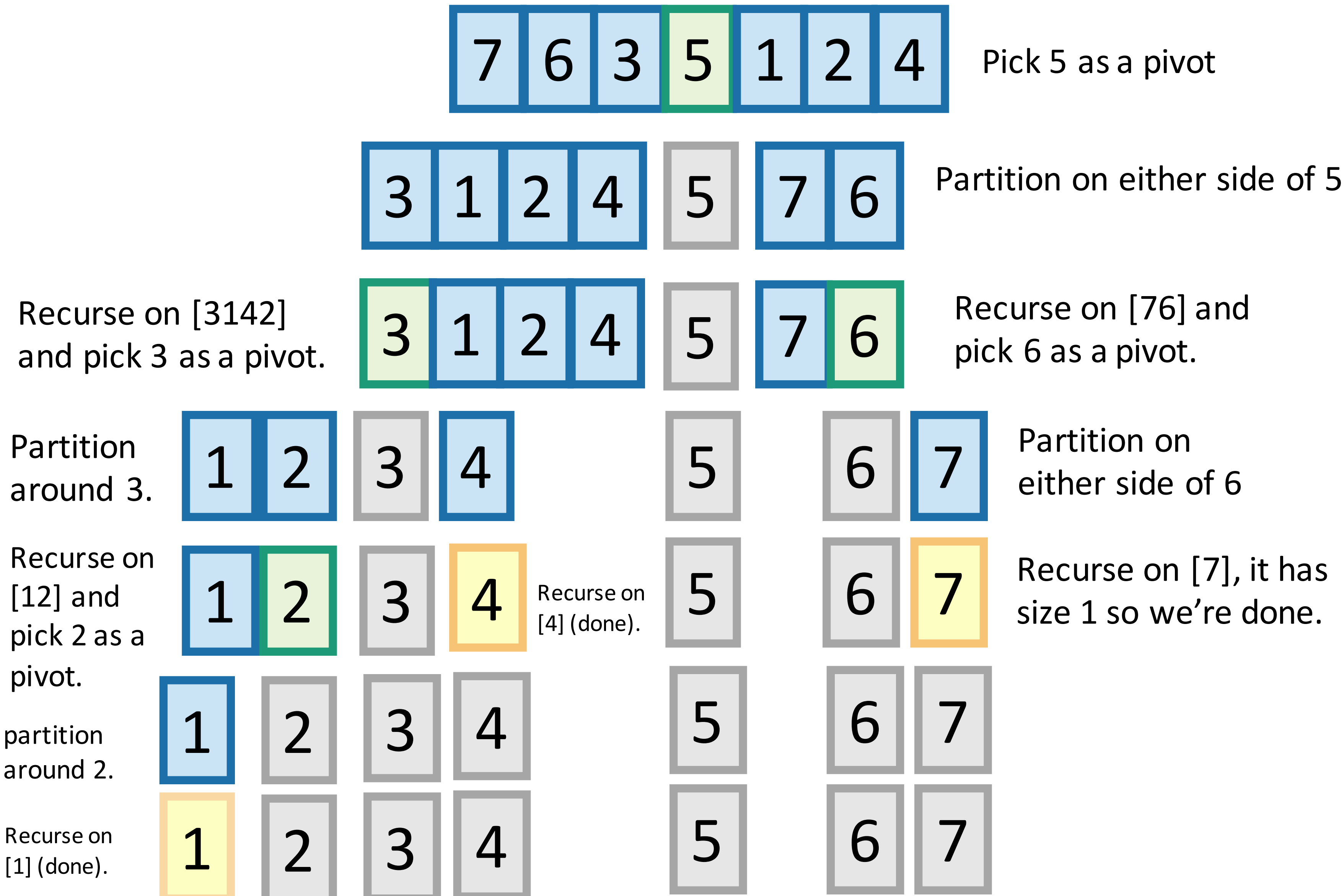
Recurse on L and R:

1	2	3	4	5	6	7

QuickSort

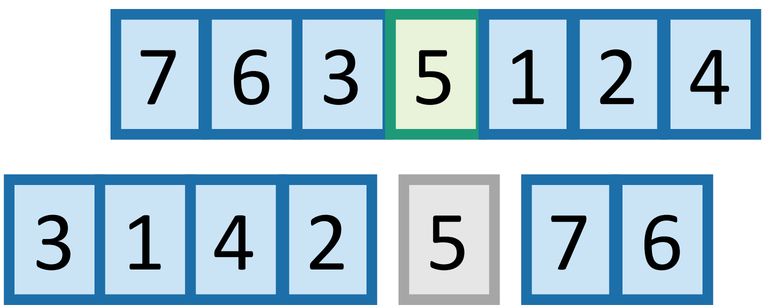
- QuickSort(A):
  - **If** len(A) <= 1:
    - **return**
  - Pick some x = A[i] at random. Call this the pivot.
  - PARTITION the rest of A into:
    - L (less than x) and
    - R (greater than x)
  - Replace A with [L, x, R] (that is, rearrange A in this order)
  - QuickSort(L)
  - QuickSort(R)

Example of recursive calls

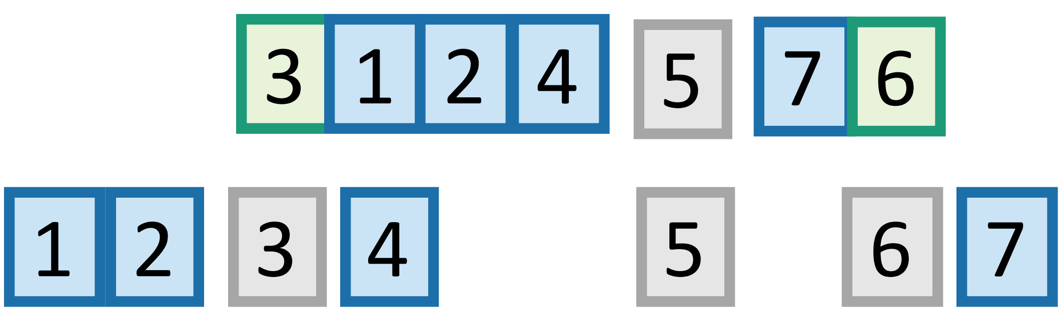


How long does this take to run?

- We will count the number of comparisons that the algorithm does.
  - This turns out to give us a good idea of the runtime. (Not obvious).
- How many times are any two items compared?



In the example before, everything was compared to 5 once in the first step….and never again.



But not everything was compared to 3.
5 was, and so were 1,2 and 4.
But not 6 or 7.

Each pair of items is compared either 0 or 1 times. Which is it?

7	6	3	5	1	2	4

Let's assume that the numbers in the array are actually the numbers 1,…,n

- Whether or not a,b are compared is a random variable, that depends on the choice of pivots. Let's say

$$X_{a,b} = \begin{cases} 1 & \text{if } a \text{ and } b \text{ are ever compared} \\ 0 & \text{if } a \text{ and } b \text{ are never compared} \end{cases}$$

- In the previous example $X_{1,5} = 1$, because item 1 and item 5 were compared.
- But $X_{3,6} = 0$, because item 3 and item 6 were NOT compared.
- Both of these depended on our random choice of pivot!

Counting comparisons

- The number of comparisons total during the algorithm is

$$\sum_{a=1}^{n} \sum_{b=a+1}^{n} X_{a,b}$$

- The expected number of comparisons is

$$E\left[\sum_{a=1}^{n} \sum_{b=a+1}^{n} X_{a,b}\right] = \sum_{a=1}^{n} \sum_{b=a+1}^{n} E[X_{a,b}]$$

using linearity of expectations.

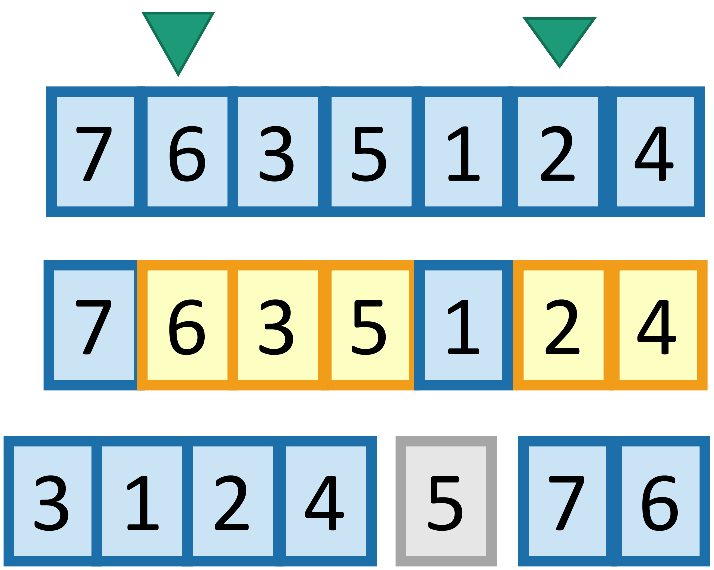
expected number of comparisons:

$$\sum_{a=1}^{n} \sum_{b=a+1}^{n} E[X_{a,b}]$$

Counting comparisons

- So we just need to figure out $E[\, X_{a,b} \,]$
- $E[\, X_{a,b} \,] = P(\, X_{a,b} = 1 \,)\cdot 1 + P(\, X_{a,b} = 0 \,) \cdot 0 = P(X_{a,b} = 1)$
  - (using definition of expectation)
- So we need to figure out

$P(X_{a,b} = 1)$ = the probability that a and b are ever compared.



Say that a = 2 and b = 6. What is the probability that 2 and 6 are ever compared?

This is exactly the probability that either 2 or 6 is first picked to be a pivot out of the highlighted entries.

If, say, 5 were picked first, then 2 and 6 would be separated and never see each other again.

Counting comparisons

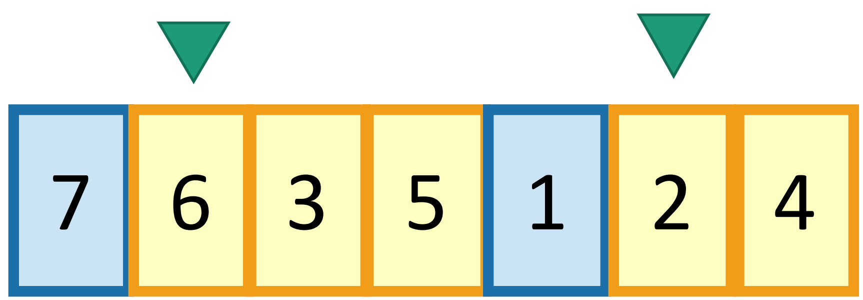
$P(\, X_{a,b} = 1 \,)$

= probability a,b are ever compared

= probability that one of a,b are picked first out of all of the b – a +1 numbers between them.

2 choices out of b-a+1…

$$= \frac{2}{b-a+1}$$



All together now…

Expected number of comparisons

- $E\left[\sum_{a=1}^{n}\sum_{b=a+1}^{n} X_{a,b}\right]$ This is the expected number of comparisons throughout the algorithm
- $= \sum_{a=1}^{n}\sum_{b=a+1}^{n} E[\,X_{a,b}]$ linearity of expectation
- $= \sum_{a=1}^{n}\sum_{b=a+1}^{n} P(\,X_{a,b} = 1)$ definition of expectation
- $= \sum_{a=1}^{n}\sum_{b=a+1}^{n} \frac{2}{b - a + 1}$ the reasoning we just did

- This is a big nasty sum, but we can do it.
- We get that this is less than 2n ln(n).

Worst-case running time for QuickSort (if time)

- Suppose that an adversary is choosing the random pivots for you.
- Then the running time might be $O(n^2)$
- In practice, this doesn't usually happen.

- **Aside**: We worked really hard last week to get a deterministic algorithm for SELECT, by picking the pivot very cleverly.
- What happens if you pick the pivot randomly?
- Turns out this is also usually a good idea.

Summary

- We can do SELECT and MEDIAN in time O(n).
- We already knew how to sort in time O(nlog(n)) with MergeSort.

- The randomized algorithm QuickSort also runs in expected time O(nlog(n)).
- In practice, QuickSort is often nicer.

- Skills of today:
  - substitution method
  - analysis of randomized algorithms.