CS2040S
Data Structures and Algorithms

Welcome!

Puzzle of the day:

You start with 100$ in your account and you can choose to participate in a bet (as many times as you like) where a fair coin is flipped.

W.p. ½ you lose 1% of your account.

W.p. ½ you gain 1% of your account.

Let n be the number of times you do this. As n tends to infinity, do you expect your account value to go: A) Up B) Stay about the same C) Go down

Last Time: Sorting

QuickSort:

- Divide-and-Conquer
- Partitioning
- Duplicates
- Bad Choices for pivots
- Paranoid Quicksort

Today: Randomized Analysis!

Paranoid QuickSort:

- Randomized Analysis

Ordered Statistics:

- Quickselect
- Randomized Analysis

QuickSort

Key Idea:

- Choose the pivot at random.

Randomized Algorithms:

- Algorithm makes decision based on random coin flips.
- Can “fool” the adversary (who provides bad input)
- Running time is a *random variable*.

Randomization

What is the difference between:

- Randomized algorithms
- Average-case analysis

Randomization

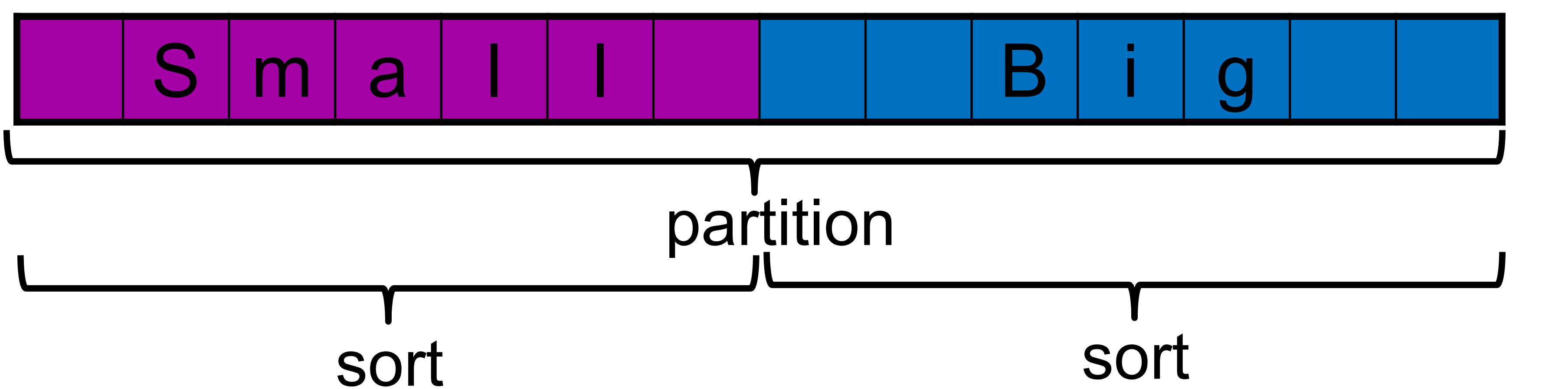
Randomized algorithm:

- Algorithm makes random choices
- For every input, there is a good probability of success.

Average-case analysis:

- Algorithm (may be) deterministic
- “Environment” chooses random input
- Some inputs are good, some inputs are bad
- For most inputs, the algorithm succeeds

```
QuickSort(A[1..n], n)
    if (n == 1) then return;
    else
      pIndex = random(1, n)
      p = 3WayPartition(A[1..n], n, pindex)
      x = QuickSort(A[1..p-1], p-1)
      y = QuickSort(A[p+1..n], n-p)
```



Paranoid QuickSort

```
ParanoidQuickSort(A[1..n], n)
    if (n == 1) then return;
    else
        repeat
            pIndex = random(1, n)
            p = partition(A[1..n], n, pIndex)
        until p > (1/10)n and p < (9/10)n

        x = QuickSort(A[1..p–1], p–1)
        y = QuickSort(A[p+1..n], n–p)
```

Paranoid QuickSort

Easier to analyze:

- Every time we recurse, we reduce the problem size by at least (1/10).
- We have already analyzed that recurrence!

Note: non-paranoid QuickSort works too

- Analysis is a little trickier (but not much).

Paranoid QuickSort

```
ParanoidQuickSort(A[1..n], n)
    if (n == 1) then return;
    else
        repeat
            pIndex = random(1, n)
            p = partition(A[1..n], n, pIndex)
        until p > (1/10)n and p < (9/10)

        x = QuickSort(A[1..p–1], p–1)
        y = QuickSort(A[p+1..n], n–p)
```

Paranoid QuickSort

T(n) → ParanoidQuickSort(A[1..n], n)

1 → **if** (n == 1) **then** return;

else

?? → **repeat**

pIndex = **random**(1,

p = partition(A[1..n],

until p > (1/10)n **and** p < (

T(k) → x = QuickSort(A[1..p–1], p

T(n-k) → y = QuickSort(A[p+1..n], n

9n/10 > k > n/10

Paranoid QuickSort

T(n)

1

O(n)
(in expectation)

T(k)
T(n-k)

9n/10 > k > n/10

ParanoidQuickSort(A[1..n], n)
if (n == 1) **then** return;
else
repeat
pIndex = **random**(1,
p = partition(A[1..n],
until p > (1/10)n **and** p < (

x = QuickSort(A[1..p–1], p
y = QuickSort(A[p+1..n], n

Paranoid QuickSort

Key claim:

- We only execute the **repeat** loop O(1) times (in expectation).

Then we know:

$$T(n) <= T(n/10) + T(9n/10) + n(\text{\# iterations of } \textbf{repeat})$$
$$= O(n \log n)$$

Paranoid QuickSort

T(n)

1

O(n)

T(k)

T(n-k)

9n/10 > k > n/10

```
ParanoidQuickSort(A[1..n], n)
    if (n == 1) then return;
    else
        repeat
                pIndex = random(1,
                p = partition(A[1..n],
        until p > (1/10)n and p < (

        x = QuickSort(A[1..p–1], p
        y = QuickSort(A[p+1..n], n
```

Probability Theory

Probability Theory

Flipping a coin:

- Pr(heads) = ½
- Pr(tails) = ½

Coin flips are independent:

- Pr(heads , heads) = ½ * ½ = ¼
- Pr(heads , tails , heads) = ½ * ½ * ½ = 1/8

Probability Theory

Flipping a coin:

- Pr(heads) = ½
- Pr(tails) = ½

Set of uniform events (e_1, e_2, e_3, …, e_k):

- Pr(e_1) = 1/k
- Pr(e_2) = 1/k
- …
- Pr(e_k) = 1/k

Probability Theory

Events **A**, **B**:

- Pr(**A**), Pr(**B**)
- **A** and **B** are independent
  (e.g., unrelated random coin flips)

Then:

- Pr(**A** and **B**) = Pr(**A**)Pr(**B**)

How many times do you have to flip a coin before it comes up heads?

How many times do you have to flip a coin before it comes up heads?

Poorly defined question…

How many times do you have to flip a coin before it comes up heads?

How many times do we <u>expect</u> to flip a coin before it comes up heads?

How many times do you have to flip a coin before it comes up heads?

How many times do we expect to flip a coin before it comes up heads?

Let T be the random variable denoting the number of flips before a coin comes up heads.

How many times do you have to flip a coin before it comes up heads?

How many times do we <u>expect</u> to flip a coin before it comes up heads?

Let T be the random variable denoting the number of flips before a coin comes up heads.

Then we wish to find: E[T]

How many times do you have to flip a coin before it comes up heads?

How many times do we <u>expect</u> to flip a coin before it comes up heads?

Let T be the random variable denoting the number of flips before a coin comes up heads.

CS1231S

Then we wish to find: E[T]

Probability Theory

Expected value:

- Weighted average

Example: event **A** has two outcomes:

- Pr(**A** = 12) = ¼
- Pr(**A** = 60) = ¾

Expected value of A:

$$E[A] = (\tfrac{1}{4})12 + (\tfrac{3}{4})60 = 48$$

Probability Theory

Set of outcomes for $X = (e_1, e_2, e_3, \ldots, e_k)$:

- $Pr(e_1) = p_1$
- $Pr(e_2) = p_2$
- …
- $Pr(e_k) = p_k$

Expected outcome:

$$E[X] = e_1p_1 + e_2p_2 + \ldots + e_kp_k$$

Probability Theory

Flipping a coin:

- $\Pr(\text{heads}) = \frac{1}{2}$
- $\Pr(\text{tails}) = \frac{1}{2}$

In two coin flips: I <u>expect</u> one heads.

Probability Theory

Define event **A**:

- **A** = number of heads in two coin flips

In two coin flips: I <u>expect</u> one heads.

- Pr(heads, heads) = ¼
- Pr(heads, tails) = ¼
- Pr(tails, heads) = ¼
- Pr(tails, tails) = ¼

2 * ¼	=	½
1 * ¼	=	¼
1 * ¼	=	¼
0 * ¼	=	0
		1

Probability Theory

Flipping a coin:

- Pr(heads) = ½
- Pr(tails) = ½

In two coin flips: I <u>expect</u> one heads.

- If you repeated the experiment many times, on average after two coin flips, you will have one heads.

Goal: calculate <u>expected</u> time of QuickSort

Probability Theory

Linearity of Expectation:

– $E[A + B] = E[A] + E[B]$

Example:

– A = # heads in 2 coin flips: $E[A] = 1$
– B = # heads in 2 coin flips: $E[B] = 1$
– $A + B$ = # heads in 4 coin flips

$$E[A+B] = E[A] + E[B] = 1 + 1 = 2$$

Probability Theory

Flipping an (unfair) coin:

- Pr(heads) = p
- Pr(tails) = (1 – p)

How many flips to get at least one head?

E[X]= expected number of flips to get one head

Example: X = 7

T T T T T T H

Probability Theory

Flipping an (unfair) coin:

- Pr(heads) = p
- Pr(tails) = (1 – p)

How many flips to get at least one head?

E[X]= Pr(heads after 1 flip)*1 +

Pr(heads after 2 flips)*2 +

Pr(heads after 3 flips)*3 +

Pr(heads after 4 flips)*4 +

…

Probability Theory

Flipping an (unfair) coin:

- Pr(heads) = p
- Pr(tails) = (1 – p)

How many flips to get at least one head?

E[X]= Pr(H)*1 +

Pr(T H)*2 +

Pr(T T H)*3 +

Pr(T T T H)*4 +

…

Probability Theory

Flipping an (unfair) coin:

- Pr(heads) = p
- Pr(tails) = $(1 - p)$

How many flips to get at least one head?

$$
\begin{aligned}
\mathbf{E}[X] = {} & p(1) + \\
& (1-p)(p)(2) + \\
& (1-p)(1-p)(p)(3) + \\
& (1-p)(1-p)(1-p)(p)(4) + \\
& \ldots
\end{aligned}
$$

Probability Theory

Flipping an (unfair) coin:

- Pr(heads) = p
- Pr(tails) = $(1 - p)$

How many flips to get at least one head?

$$\mathbf{E}[X] = (p)(1) + (1 - p)(1 + \mathbf{E}[X])$$

How many more flips to get a head?

Idea: If I flip "tails," the expected number of additional flips to get a "heads" is still $\mathbf{E}[X]$!!

Probability Theory

Flipping an (unfair) coin:

- Pr(heads) = p
- Pr(tails) = $(1 - p)$

How many flips to get at least one head?

$$\mathbf{E}[X] = (p)(1) + (1 - p)\,(1 + \mathbf{E}[X])$$

$$= p + 1 - p + 1\mathbf{E}[X] - p\mathbf{E}[X]$$

Probability Theory

Flipping an (unfair) coin:

- $\Pr(\text{heads}) = p$
- $\Pr(\text{tails}) = (1 - p)$

How many flips to get at least one head?

$$\mathbf{E}[X] = (p)(1) + (1 - p)\,(1 + \mathbf{E}[X])$$

$$= p + 1 - p + 1\mathbf{E}[X] - p\mathbf{E}[X]$$

$$\mathbf{E}[X] - \mathbf{E}[X] + p\mathbf{E}[X] = 1$$

Probability Theory

Flipping an (unfair) coin:

– $\Pr(\text{heads}) = p$

– $\Pr(\text{tails}) = (1 - p)$

How many flips to get at least one head?

$$\mathbf{E}[X] = (p)(1) + (1 - p)\,(1 + \mathbf{E}[X])$$

$$= p + 1 - p + 1\mathbf{E}[X] - p\mathbf{E}[X]$$

$$p\mathbf{E}[X] = 1$$

$$\mathbf{E}[X] = 1/p$$

Probability Theory

Flipping an (unfair) coin:

- Pr(heads) = p
- Pr(tails) = $(1 - p)$

How many flips to get at least one head?

If $p = ½$, the expected number of flips to get one head equals:

$$\mathbf{E}[X] = 1/p = 1/½ = 2$$

Paranoid QuickSort

```
ParanoidQuickSort(A[1..n], n)
    if (n == 1) then return;
    else
        repeat
            pIndex = random(1, n)
            p = partition(A[1..n], n, pIndex)
        until p > (1/10)n and p < (9/10)

        x = QuickSort(A[1..p–1], p–1)
        y = QuickSort(A[p+1..n], n–p)
```

How many times do we repeat?

QuickSort Partition

Remember:

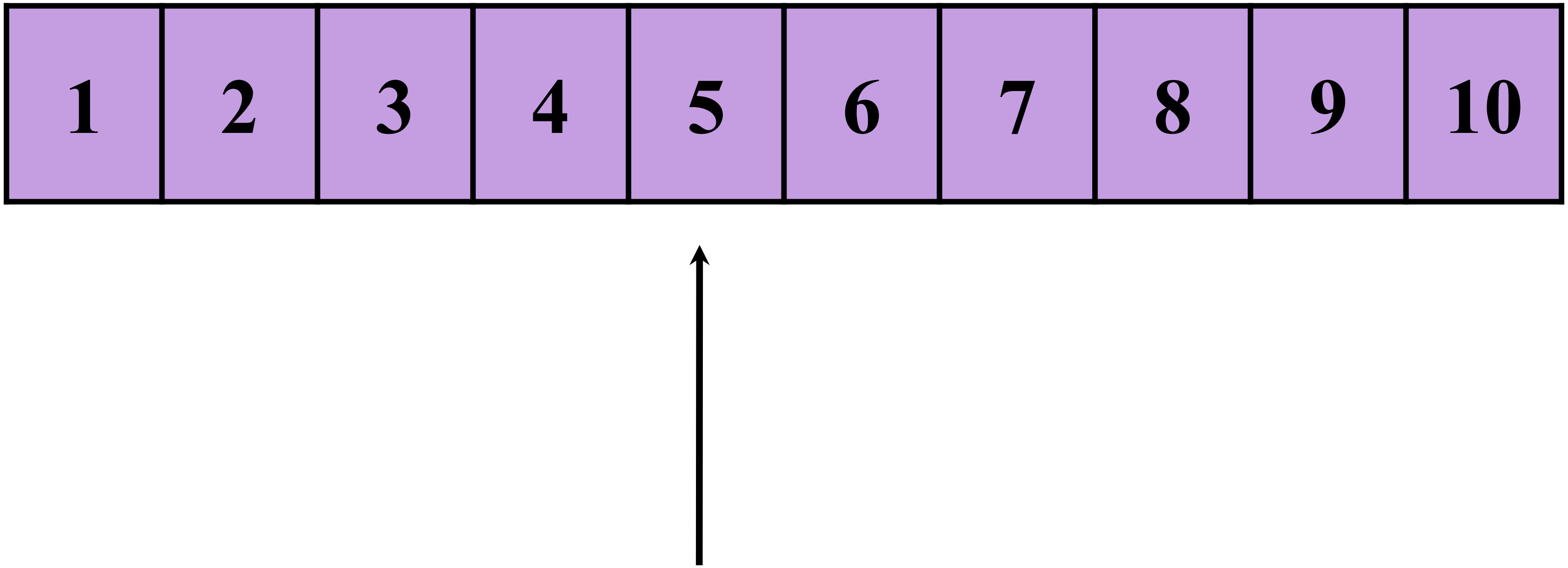
A *pivot* is **good** if it divides the array into two pieces, each of which is size at least $n/10$.

If we choose a pivot at random, what is the probability that it is <u>good</u>?

1. 1/10
2. 2/10
3. 8/10
4. $1/\log(n)$
5. $1/n$
6. I have no idea.

Choosing a Good Pivot

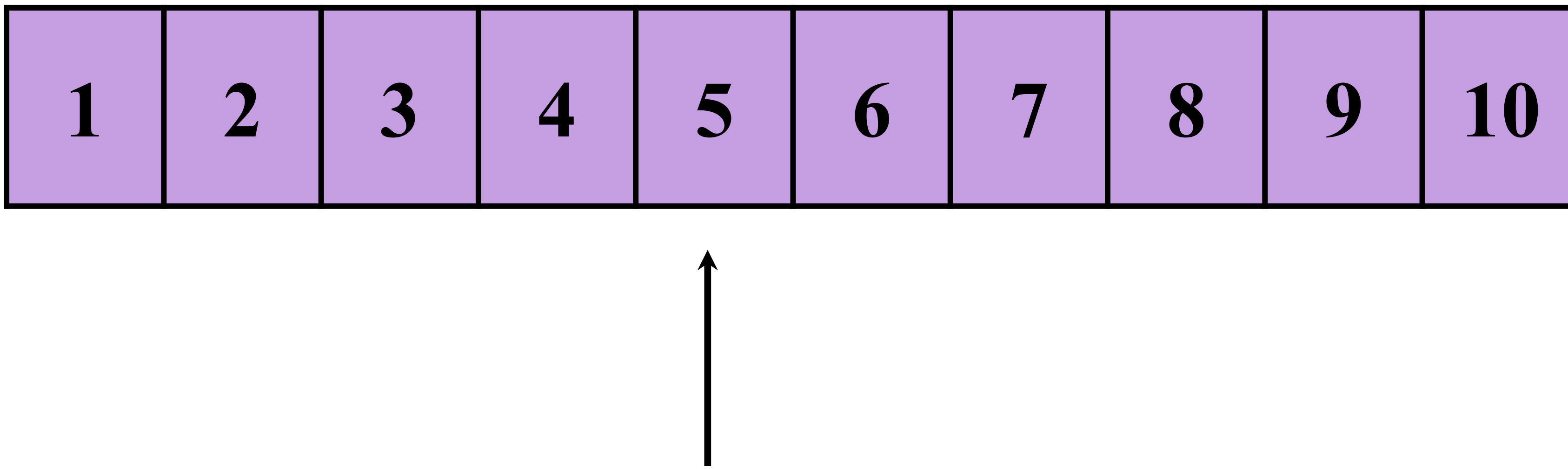
Imagine the array divided into 10 pieces:



Choose a random point at which to partition.

Choosing a Good Pivot

Imagine the array divided into 10 pieces:

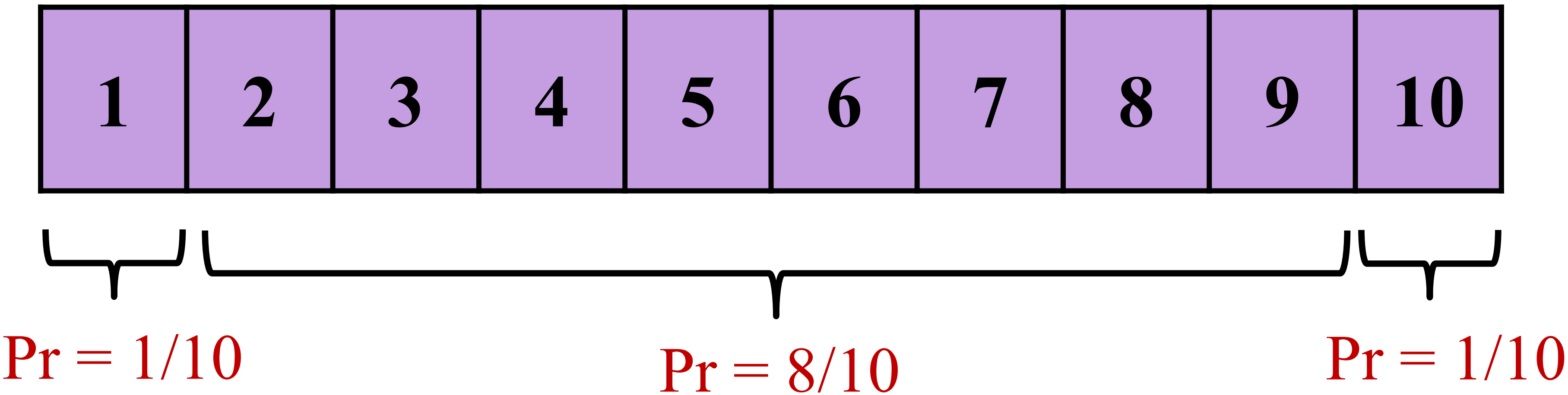


Choose a random point at which to partition.

- 10 possible events
- each occurs with probability 1/10

Choosing a Good Pivot

Imagine the array divided into 10 pieces:

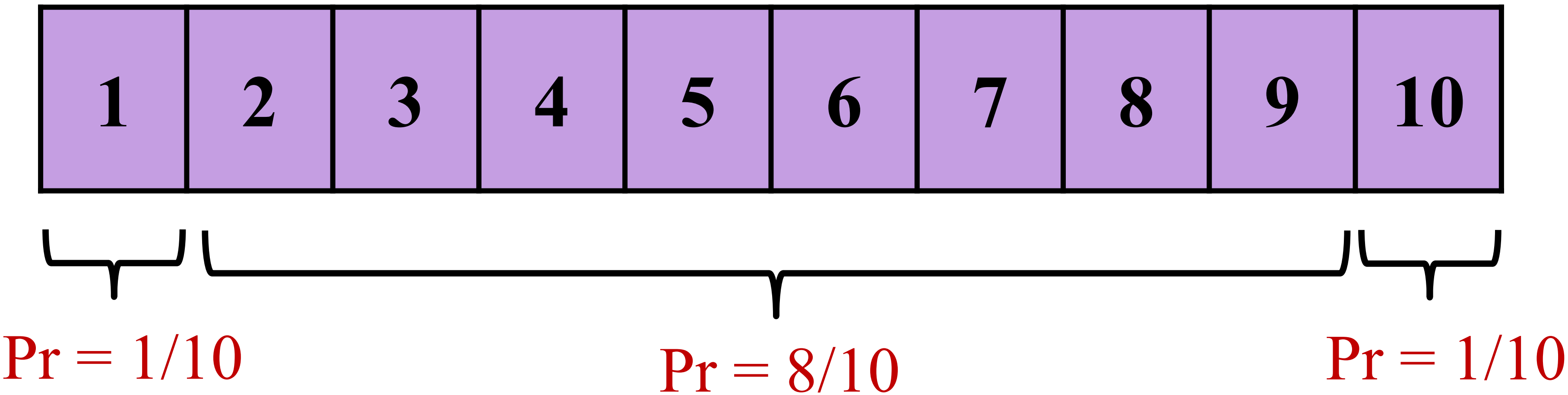


Choose a random point at which to partition.

- 10 possible events
- each occurs with probability 1/10

Choosing a Good Pivot

Imagine the array divided into 10 pieces:



Probability of a good pivot:

$p = 8/10$

$(1 - p) = 2/10$

Choosing a Good Pivot

Probability of a good pivot:

$$p = 8/10$$

$$(1 - p) = 2/10$$

Expected number of times to repeatedly choose a pivot to achieve a good pivot:

$$\mathbf{E}[\#\ \text{choices}] = 1/p = 10/8 < 2$$

Paranoid QuickSort

QuickSort(A[1..n], n)

if (n==1) **then** return;

else

repeat

Expect to run this only 2 times

$pIndex$ = **random**(1, n)

p = partition(A[1..n], n, $pIndex$)

until $p > n/10$ **and** $p < n(9/10)$

x = QuickSort(A[1..p–1], p–1)

y = QuickSort(A[p+1..n], n–p)

Paranoid QuickSort

Key claim:

We only execute the **repeat** loop < 2 times (in expectation).

Then we know:

$$\mathbf{E}[\mathrm{T}(n)] = \mathbf{E}[\mathrm{T}(k)] + \mathbf{E}[\mathrm{T}(n-k)] + \mathbf{E}[\#\text{ pivot choices}](n)$$

$$<= \mathbf{E}[\mathrm{T}(k)] + \mathrm{E}[\mathrm{T}(n-k)] + 2n$$

$$= \mathrm{O}(n \log n)$$

QuickSort Optimizations

Many, many optimizations and variants.

For small arrays, use InsertionSort.

- Stop recursion at arrays of size MinQuickSort.
- Do one InsertionSort on full array when done.

If array contains repeated keys, be careful!

- Use 3 way partitioning.

Order Statistics

Find k^{th} smallest element in an *unsorted* array:

$\mathbf{x_{10}}$	$\mathbf{x_2}$	$\mathbf{x_4}$	$\mathbf{x_1}$	$\mathbf{x_5}$	$\mathbf{x_3}$	$\mathbf{x_7}$	$\mathbf{x_8}$	$\mathbf{x_9}$	$\mathbf{x_6}$

E.g.: Find the median ($k = n/2$)

Find the 7th element ($k = 7$)

Order Statistics

Find k^{th} smallest element in an *unsorted* array:

1	**2**	**3**	**4**	**5**	**5**	**6**	**7**	**8**	**9**

How to count duplicate items?

Find the 5^{th} element ($k = 5$): 5

Find the 6^{th} element ($k = 6$): 5

Order Statistics

Find k^{th} smallest element in an *unsorted* array:

$\mathbf{x_{10}}$	$\mathbf{x_2}$	$\mathbf{x_4}$	$\mathbf{x_1}$	$\mathbf{x_5}$	$\mathbf{x_3}$	$\mathbf{x_7}$	$\mathbf{x_8}$	$\mathbf{x_9}$	$\mathbf{x_6}$

Option 1:

- Sort the array.
- Return element number k.

Order Statistics

Find k^{th} smallest element in an *unsorted* array:

$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{x_4}$	$\mathbf{x_5}$	$\mathbf{x_6}$	$\mathbf{x_7}$	$\mathbf{x_8}$	$\mathbf{x_9}$	$\mathbf{x_{10}}$

Option 1:

- Sort the array.
- Return element number k.

Running time?

Order Statistics

Find k^{th} smallest element in an *unsorted* array:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

Option 1:

- Sort the array.
- Return element number k.

Running time: O(n log n)

Order Statistics

Find k^{th} smallest element in an *unsorted* array:

x_{10}	x_2	x_4	x_1	x_5	x_3	x_7	x_8	x_9	x_6

Option 2:

- Only do the minimum amount of sorting necessary

Order Statistics

Key Idea: partition the array

x_2	x_4	x_1	x_3	x_5	x_7	x_8	x_9	x_6	x_{10}

Now continue searching in the correct half.

E.g.: Partition around x_5 and recursively search for x_3 in left half.

Order Statistics

Example: search for 5th element

9	**22**	**13**	**17**	**5**	**3**	**100**	**6**	**19**	**8**

Order Statistics

Example: search for 5^{th} element

9	**22**	**13**	**17**	**5**	**3**	**100**	**6**	**19**	**8**

Partition around random pivot: 17

9	**8**	**13**	**5**	**3**	**6**	**17**	**100**	**19**	**22**
1	2	3	4	5	6	**7**	**8**	**9**	**10**

Order Statistics

Example: search for 5th element

9	8	13	5	3	6	17	100	19	22
1	2	3	4	5	6	**7**	**8**	**9**	**10**

Search for 5th element in left half.

9	8	13	5	3	6				
1	2	3	4	5	6				

Order Statistics

Example: search for 5th element

9	**8**	**13**	**5**	**3**	**6**				

Partition around random pivot: 8

6	**3**	**5**	**8**	**13**	**9**				
1	2	3	**4**	5	6				

Order Statistics

Example: search for 5th element

9	**8**	**13**	**5**	**3**	**6**				

Search for: 5 – 4 = 1 in right half

6	**3**	**5**	**8**	**13**	**9**				
1	2	3	**4**	5	6				

Order Statistics

Search for: 5 – 4 = 1 in right half

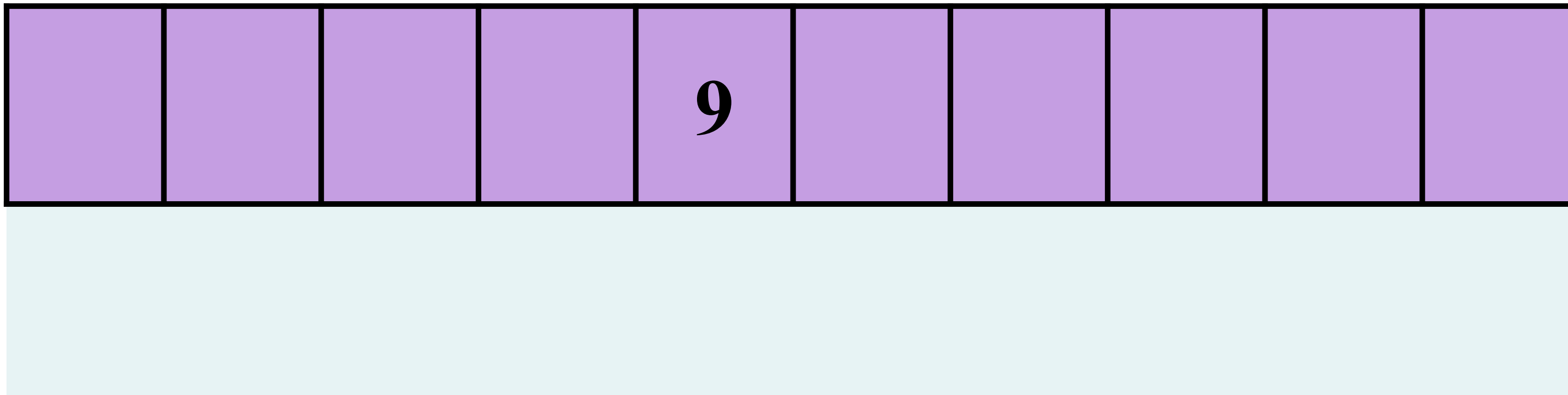
				13	**9**				

Partition around random pivot: 13

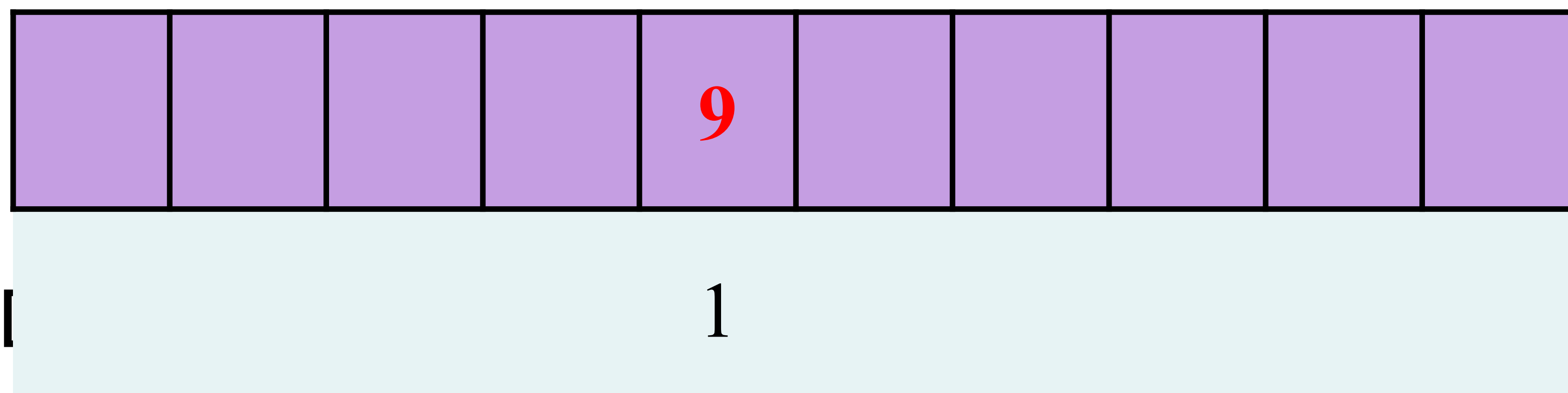
				9	**13**				
				1	2				

Order Statistics

Search for: 1 in left half

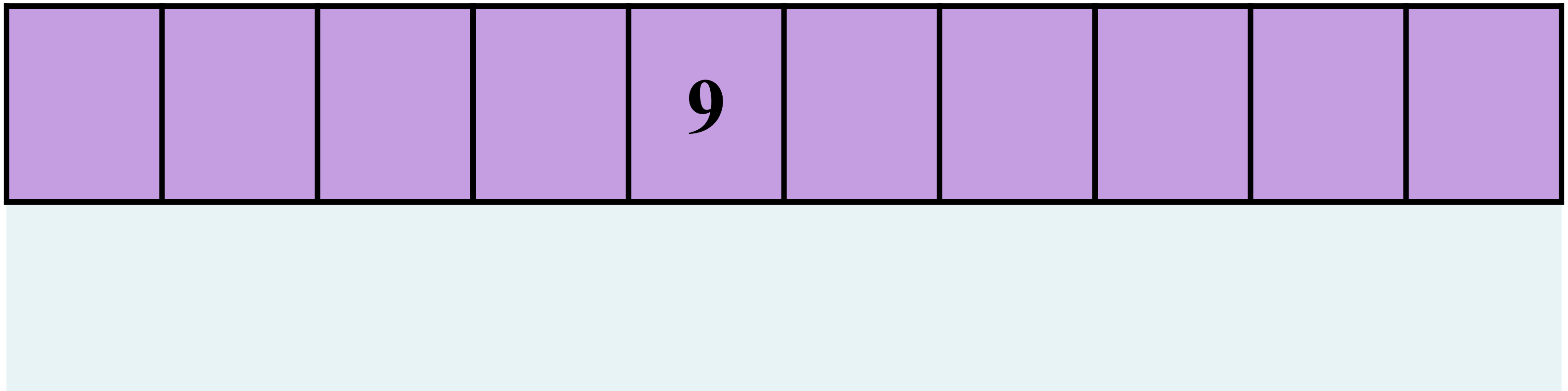


Partition around random pivot: 13

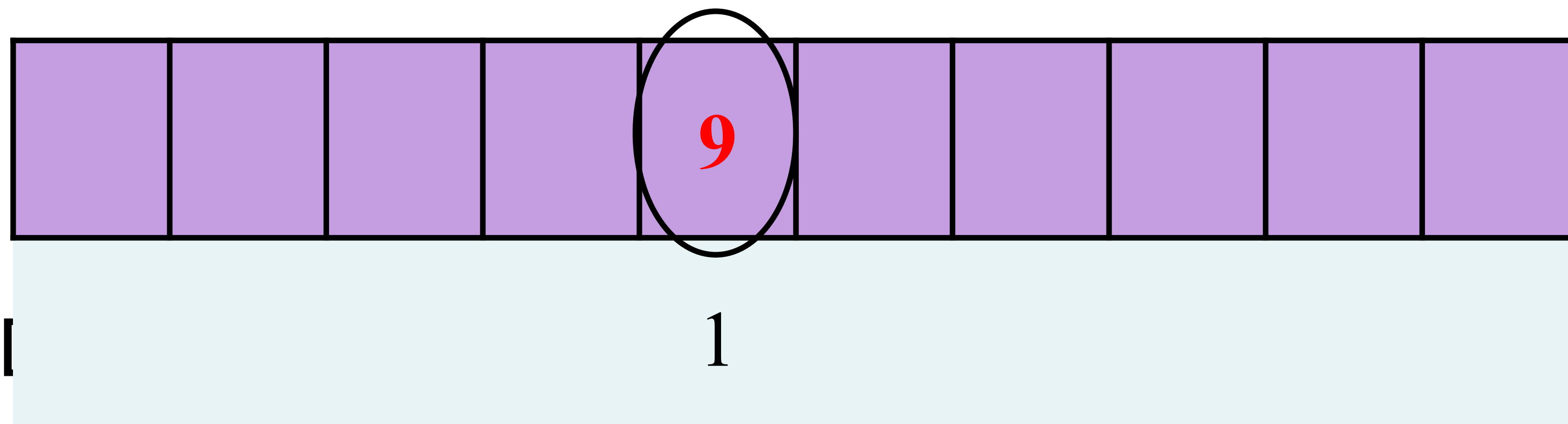


Order Statistics

Search for: 1 in left half



Partition around random pivot: 13



Finding the kth smallest element

```
Select(A[1..n], n, k)

    if (n == 1) then return A[1];
    else   Choose random pivot index pIndex.
           p = partition(A[1..n], n, pIndex)
           if (k == p) then return A[p];
           else if (k < p) then
                  return Select(A[1..p–1], k)
           else if (k > p) then
                  return Select(A[p+1], k – p)
```

Order Statistics

Recursing right and left are not exactly the same.

Example: search for 8^{th} element

9	22	13	17	5	3	100	6	19	8

Partition around random pivot: 8

5	6	3	8	17	13	100	22	19	9
1	2	3	4	5	6	**7**	**8**	**9**	**10**

Search for 4^{th} element on the right.

Order Statistics

Recursing right and left are not exactly the same.

Example: search for 4th element

9	22	13	17	5	3	100	6	19	8

Partition around random pivot: 8

5	6	3	8	17	13	100	22	19	9
1	2	3	4	5	6	**7**	**8**	**9**	**10**

Return 8.

Finding the k^{th} smallest element

```
Select(A[1..n], n, k)

    if (n == 1) then return A[1];
    else   Choose random pivot index pIndex.
           p = partition(A[1..n], n, pIndex)
           if (k == p) then return A[p];
           else if (k < p) then
                   return Select(A[1..p–1], k)
           else if (k > p) then
                   return Select(A[p+1], k – p)
```

Finding the k^{th} smallest element

Key point:

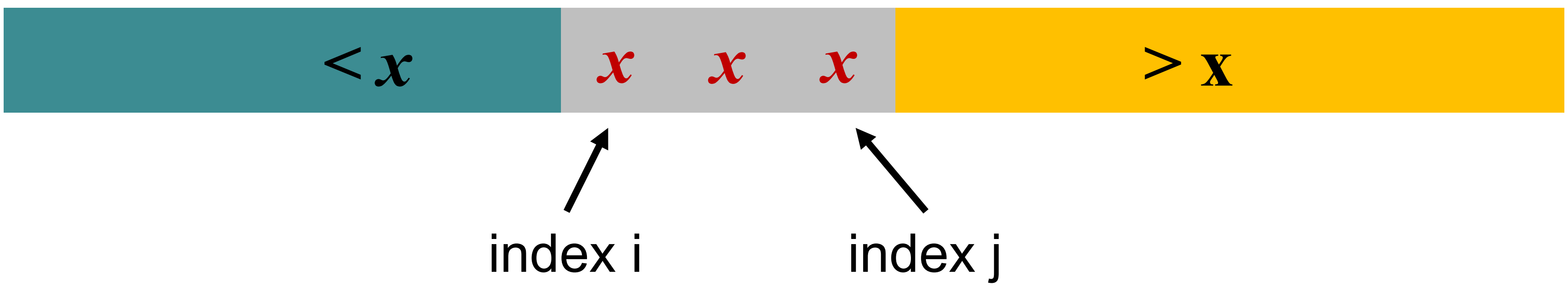
- Only recurse *once*!

- Why not recurse twice?
  - Does not help---the correct element is only on one side.
  - You do not need to sort both sides!
  - Makes it run a lot faster.
  - If you recurse on both sides, you are sorting!

Finding the k^{th} smallest element

What about duplicates?

3-way partitioning:



if (k < i): Select(A, begin, i-1, k)

if (k > j): Select(A, j+1, end, k-j)

if (i <= k <= j): return x

Analysis

Analysis

Paranoid-Select:

Repeatedly partition until at least $n/10$ in each half of the partition.

```
repeat
        p = partition(A[1..n], n, pIndex)
until (p > n/10) and (p < 9n/10)
```

Analysis

Paranoid-Select:

Repeatedly partition until at least $n/10$ in each half of the partition.

Recurrence:

$$\mathbf{E}[T(n)] \leq \mathbf{E}[T(9n/10)] + \mathbf{E}[\#\text{ partitions}](n)$$

cost of partitioning

Analysis

Paranoid-Select:

Repeatedly partition until at least $n/10$ in each half of the partition.

Recurrence:

$$\mathbf{E}[\mathrm{T}(\mathrm{n})] \le \mathbf{E}[\mathrm{T}(9\mathrm{n}/10)] + \mathbf{E}[\#\text{ partitions}](n)$$
$$\le \mathbf{E}[\mathrm{T}(9\mathrm{n}/10)] + 2n$$

Recurrence

What is the solution to the following recurrence?

$$T(n) \quad \leq \quad T(9n/10) + 2n$$

Recurrence

What is the solution to the following recurrence?

$$T(n) \leq T(9n/10) + 2n$$

$$\leq T(81n/100) + 2(9/10)n + 2n$$

Recurrence

What is the solution to the following recurrence?

$$T(n) \leq T(9n/10) + 2n$$

$$\leq T(81n/100) + 2(9/10)n + 2n$$

$$\leq T(729n/1000) + 2(81/100)n + 2(9/10)n + 2n$$

$$\ldots$$

Recurrence

What is the solution to the following recurrence?

$$T(n) \leq T(9n/10) + 2n$$

$$\leq T(81n/100) + 2(9/10)n + 2n$$

$$\leq T(729n/1000) + \underline{2(81/100)n + 2(9/10)n + 2n}$$

…

$$\begin{aligned} s_n &= ar^0 + ar^1 + \cdots + ar^{n-1} \\ &= \sum_{k=0}^{n-1} ar^k = \sum_{k=1}^{n} ar^{k-1} \\ &= \begin{cases} a\left(\frac{1-r^n}{1-r}\right), & \text{for } r \neq 1 \\ an, & \text{for } r = 1 \end{cases} \end{aligned}$$

Recurrence

What is the solution to the following recurrence?

$$T(n) \leq T(9n/10) + 2n$$
$$= O(n)$$

geometric series

$$2n(1 + (9/10) + (9/10)^2 + (9/10)^3 + (9/10)^4 + \ldots) <= O(n)$$

Analysis

Paranoid-Select:

Repeatedly partition until at least $n/10$ in each half of the partition.

Recurrence:

$$\mathbf{E}[\mathrm{T(n)}] \le \mathbf{E}[\mathrm{T(9n/10)}] + \mathbf{E}[\#\ \mathrm{partitions}](n)$$

$$\le \mathbf{E}[\mathrm{T(9n/10)}] + 2n$$

$$\le \mathrm{O}(n)$$

Question: If instead of 1/10 : 9/10, what happens if we did ⅓ : ⅔ ?

Analysis

Paranoid-Select:

Repeatedly partition until at least $n/10$ in each half of the partition.

Recurrence:

$$\mathbf{E}[\mathrm{T(n)}] \le \mathbf{E}[\mathrm{T(9n/10)}] + \mathbf{E}[\#\ \mathrm{partitions}](n)$$
$$\le \mathbf{E}[\mathrm{T(9n/10)}] + 2n$$
$$\le \mathrm{O}(n)$$

Recurrence: $T(n) = T(n/c) + O(n)$ for $c > 1$

Analysis

For you to think about:

1. For paranoid quicksort: What if we wanted ½ : ½ split for quicksort? Does this change the analysis?

1. What if we were less paranoid: E.g. As long as the pivot element is O(1) away from the ends, we recurse. Can we still prove O(n log n) in expectation?

Summary

QuickSort: O(n log n)

- How to partition efficiently
- Paranoid Quicksort
- Randomized analysis

Order Statistics: O(n)

- Finding the k^{th} smallest element in an array.
- Key idea: partition
- Paranoid Select

Next Week: Trees!