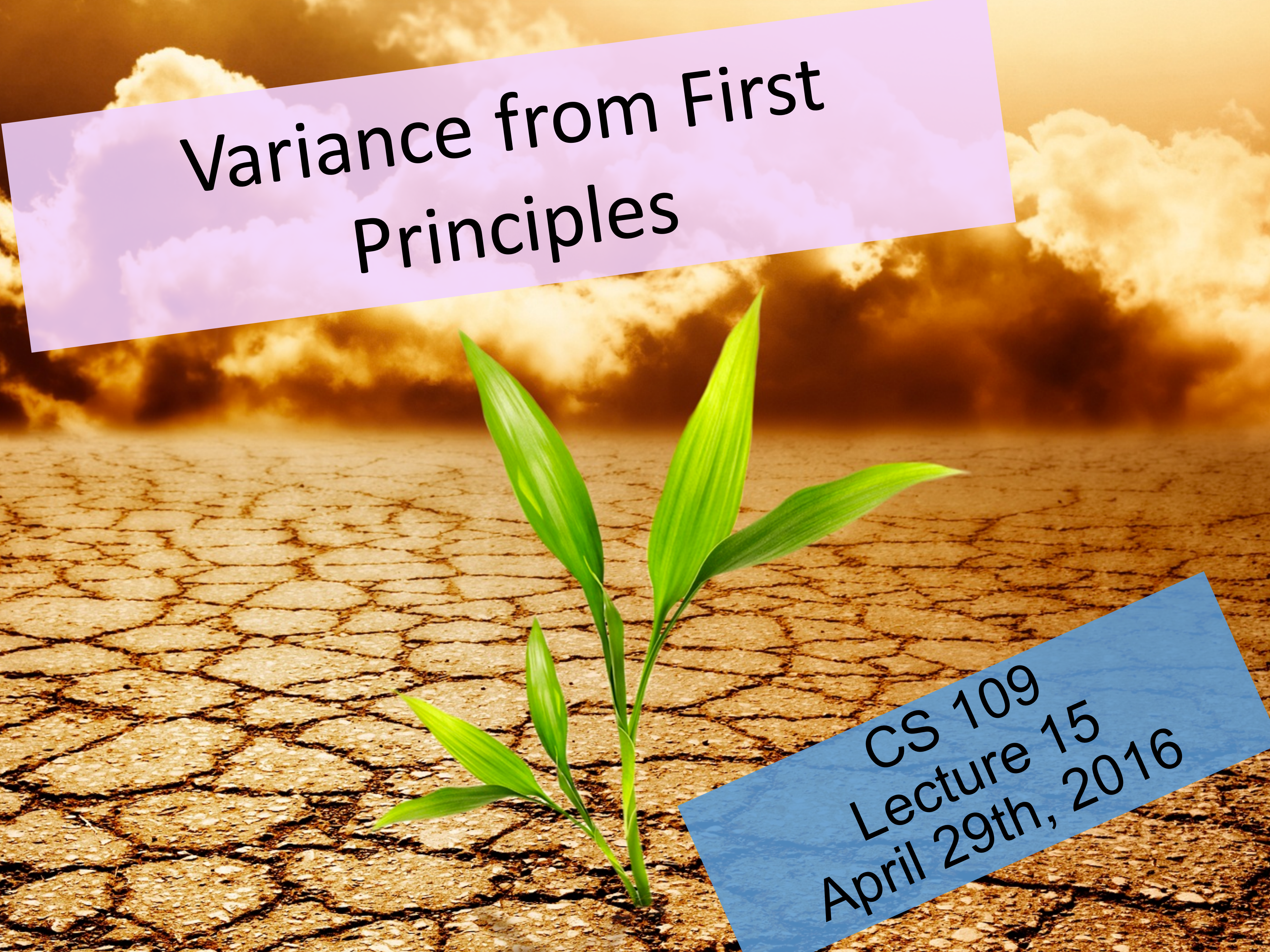
Variance from First Principles



CS 109
Lecture 15
April 29th, 2016

Course Mean

$$E[\mathrm{CS109}]$$

This is actual midpoint of course
(Just wanted you to know)

Review

Bayes Theorem

- Ross's form:

$$P(E) = P(EF) + P(EF^c)$$
$$= P(E \mid F)\,P(F) + P(E \mid F^c)\,P(F^c)$$

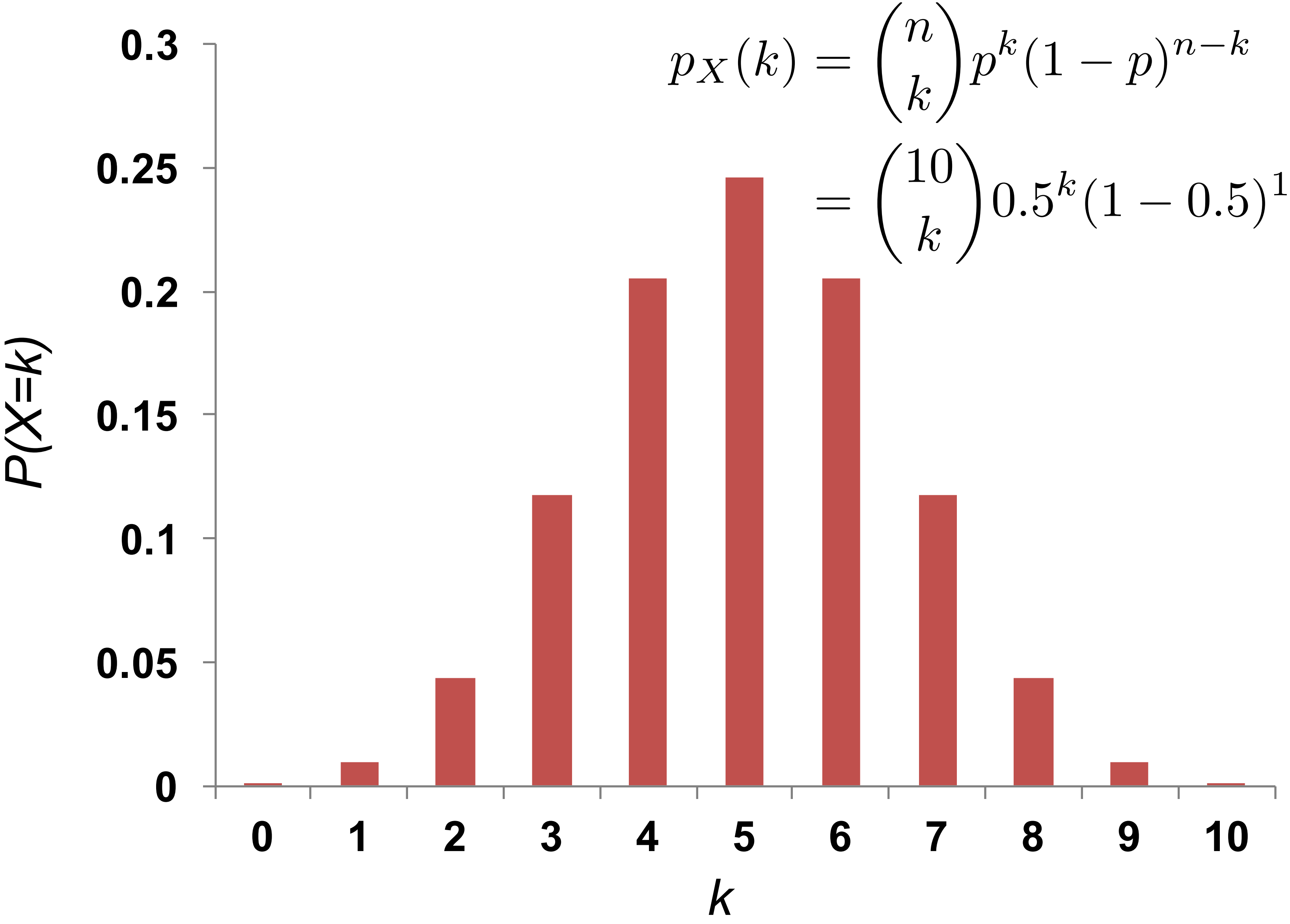
- Most common form:

$$P(F \mid E) = P(EF) \,/\, P(E)$$
$$= \frac{P(E \mid F)\,P(F)}{P(E)}$$

- Expanded form:

$$P(F \mid E) = \frac{P(E \mid F)\,P(F)}{P(E \mid F)\,P(F) + P(E \mid F^c)\,P(F^c)}$$

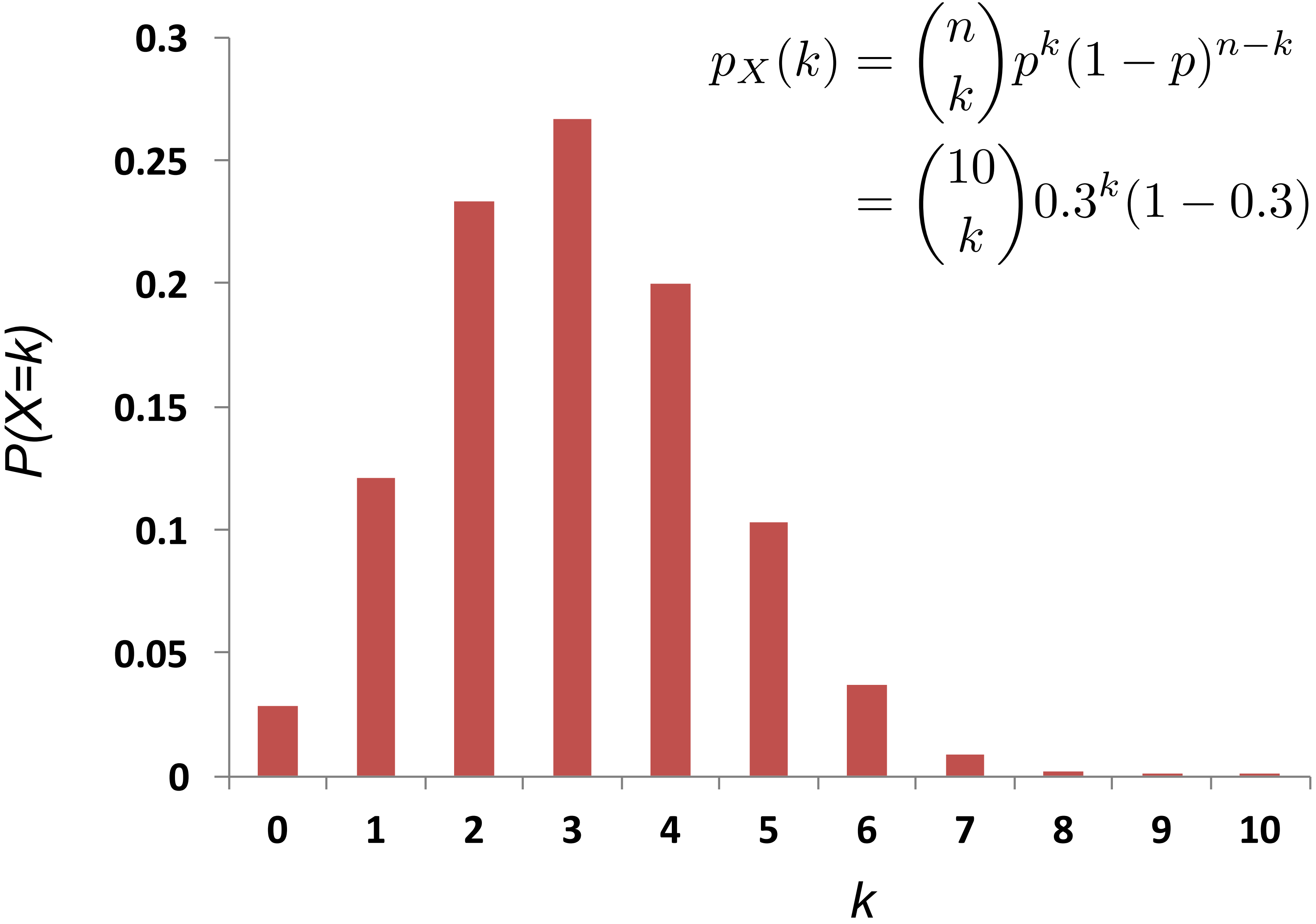
PMF for X ~ Bin(10, 0.5)



$$p_X(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

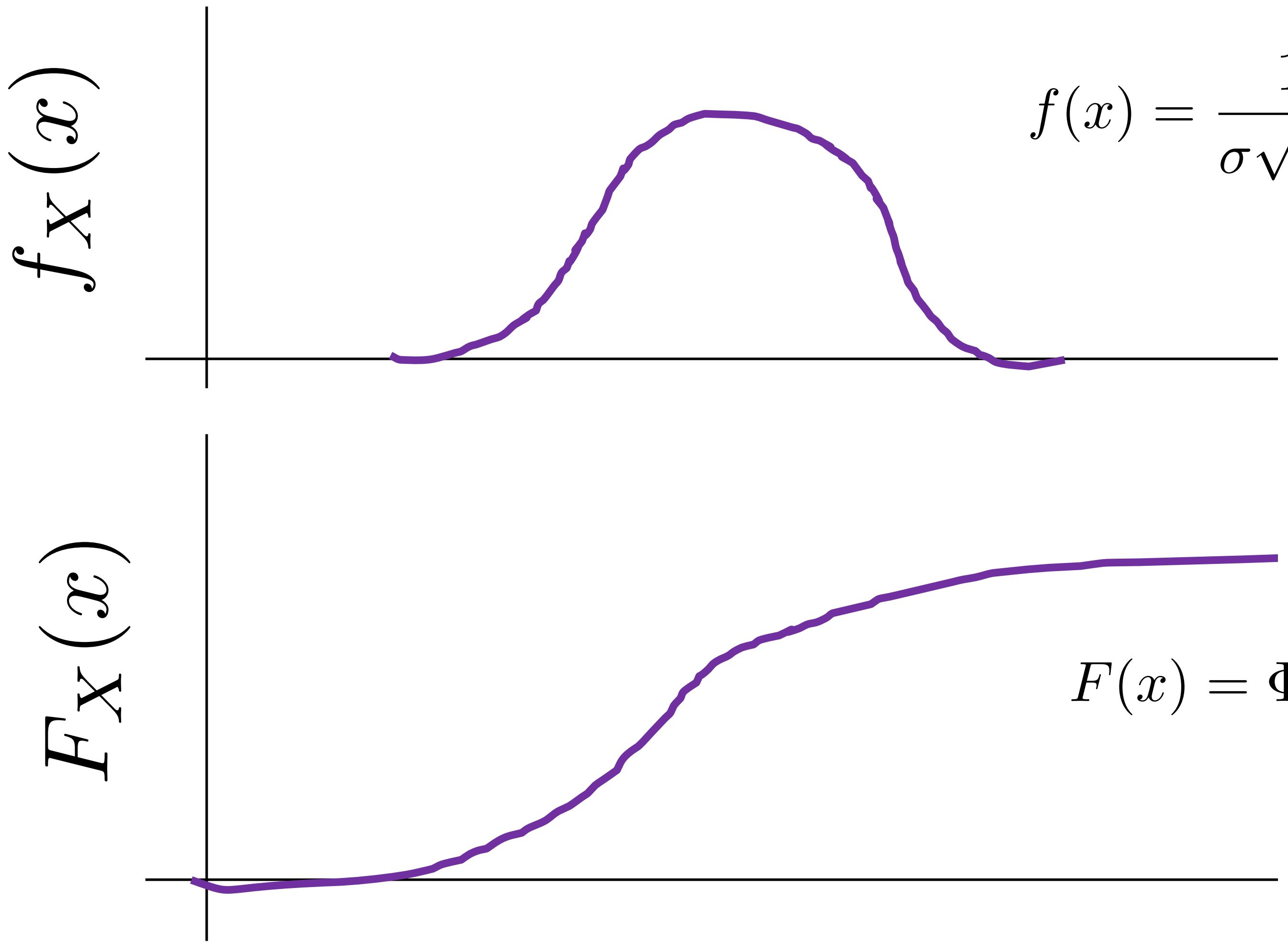
$$= \binom{10}{k} 0.5^k (1-0.5)^{10-k}$$

PMF for X ~ Bin(10, 0.3)



$$
\begin{aligned}
p_X(k) &= \binom{n}{k} p^k (1-p)^{n-k} \\
&= \binom{10}{k} 0.3^k (1-0.3)^{10-k}
\end{aligned}
$$

PDF and CDF of X ~ N(4, 9)

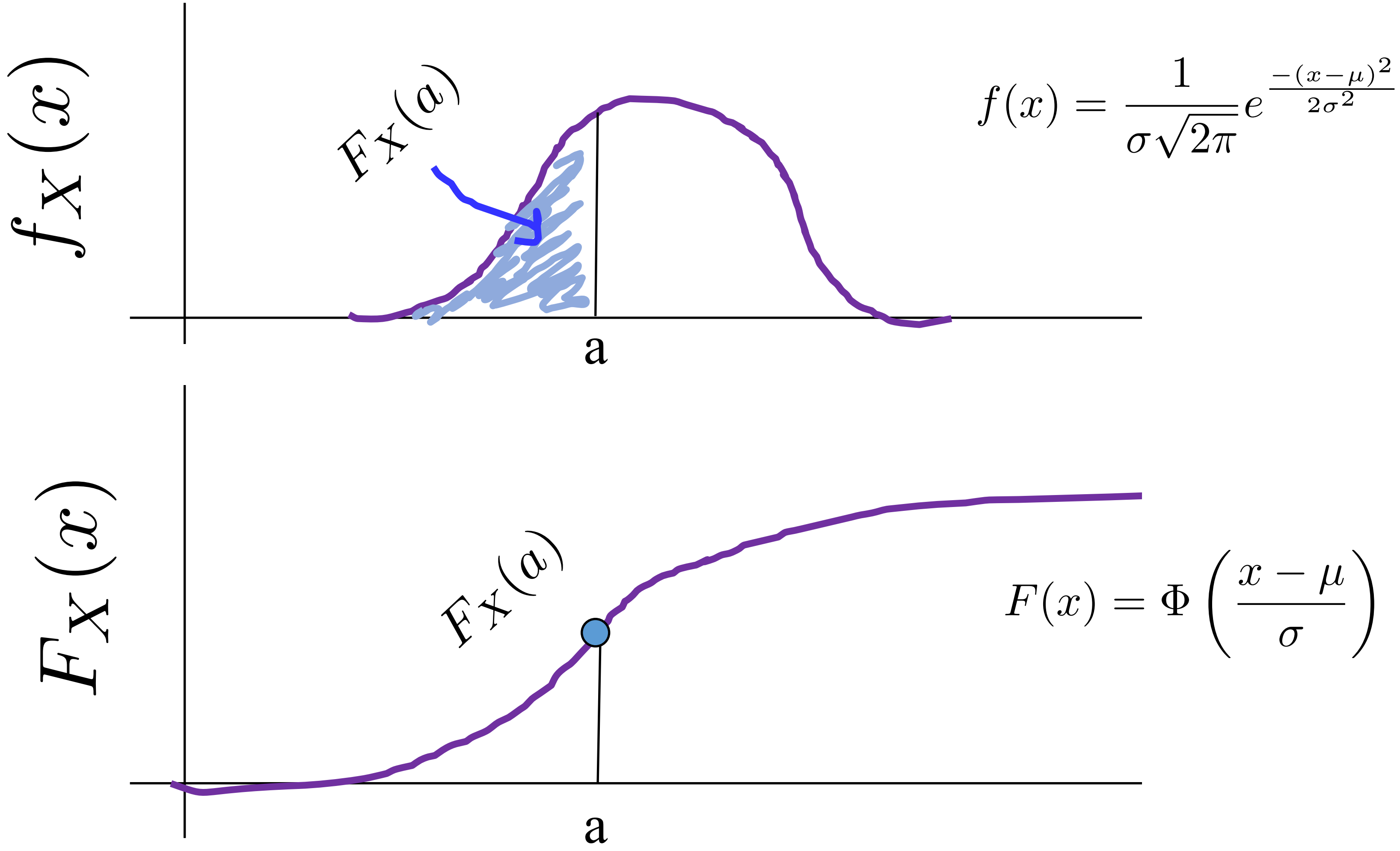


$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

A CDF is the integral from –infinity to x of the PDF

PDF and CDF of X ~ N(4, 9)

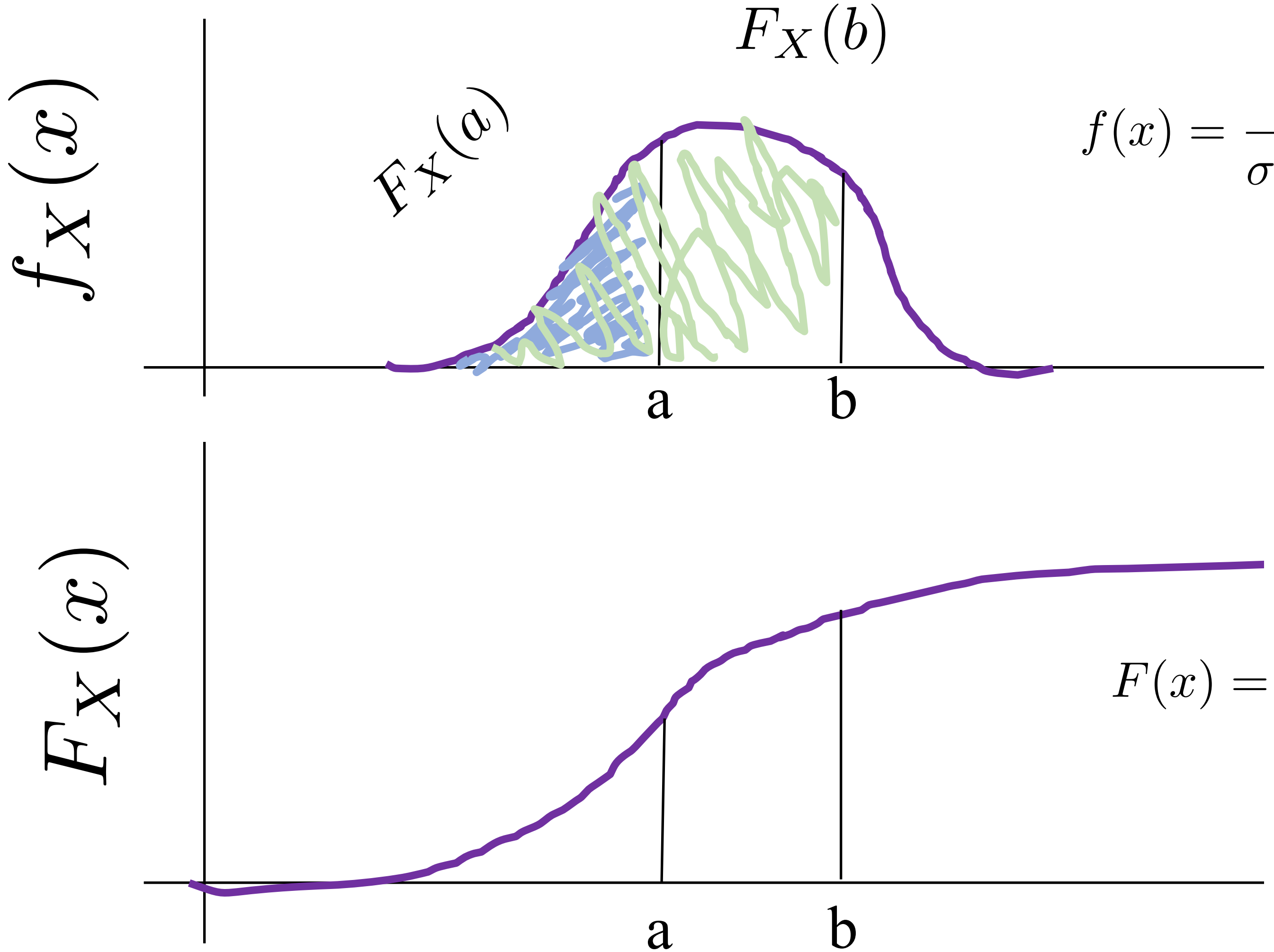


$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

A CDF is the integral from –infinity to x of the PDF

PDF and CDF of X ~ N(4, 9)

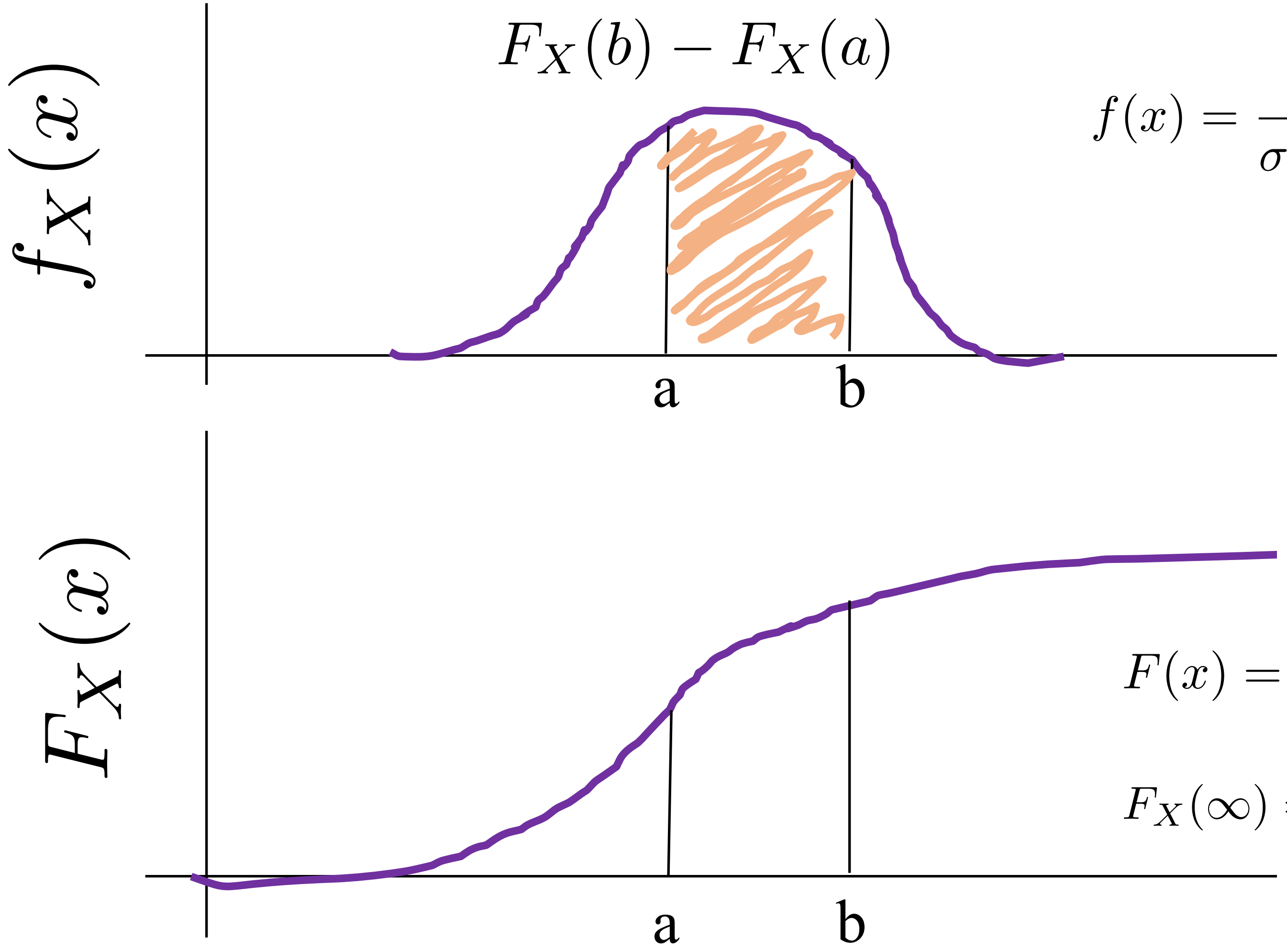


$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

A CDF is the integral from –infinity to x of the PDF

PDF and CDF of X ~ N(4, 9)



$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

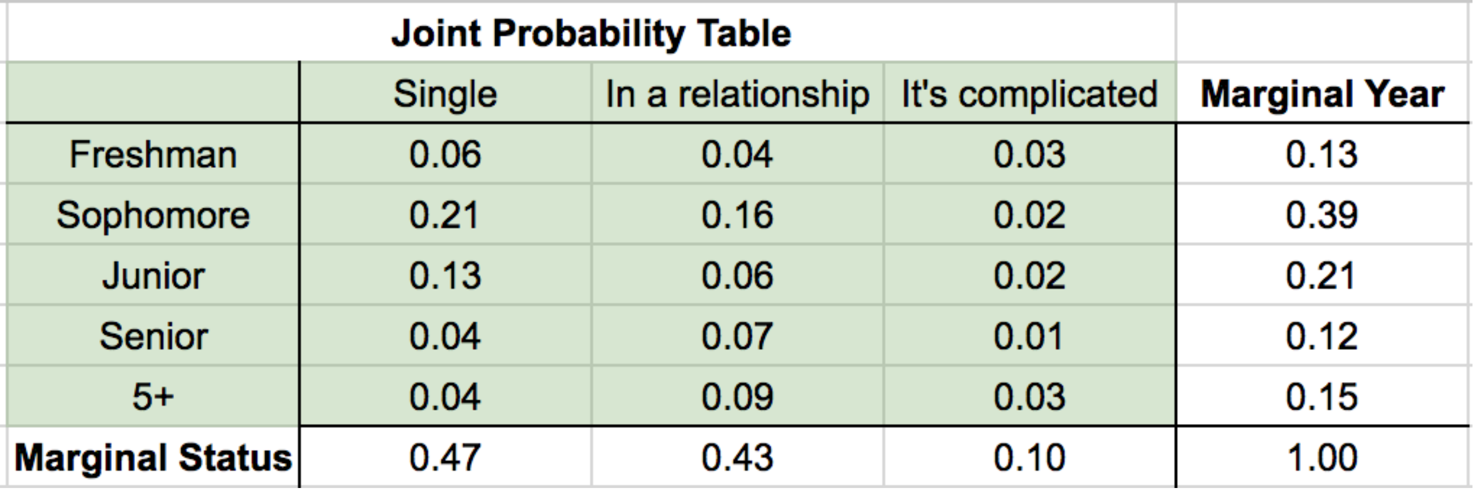
$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

$$F_X(\infty) = 1$$

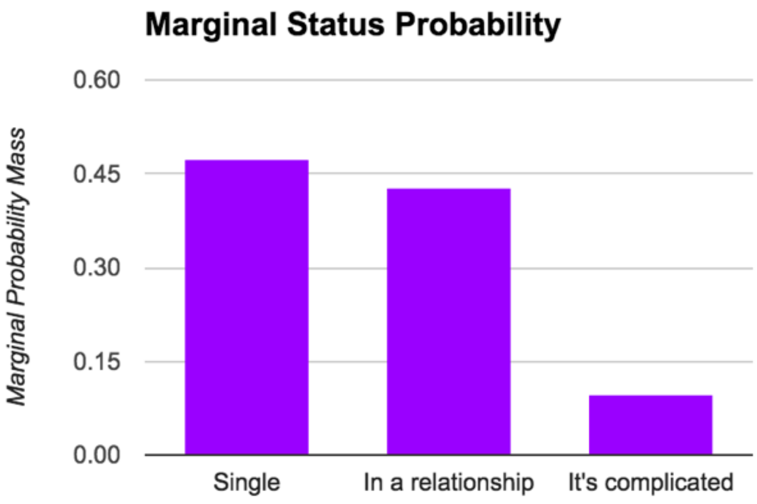
A CDF is the integral from –infinity to x of the PDF

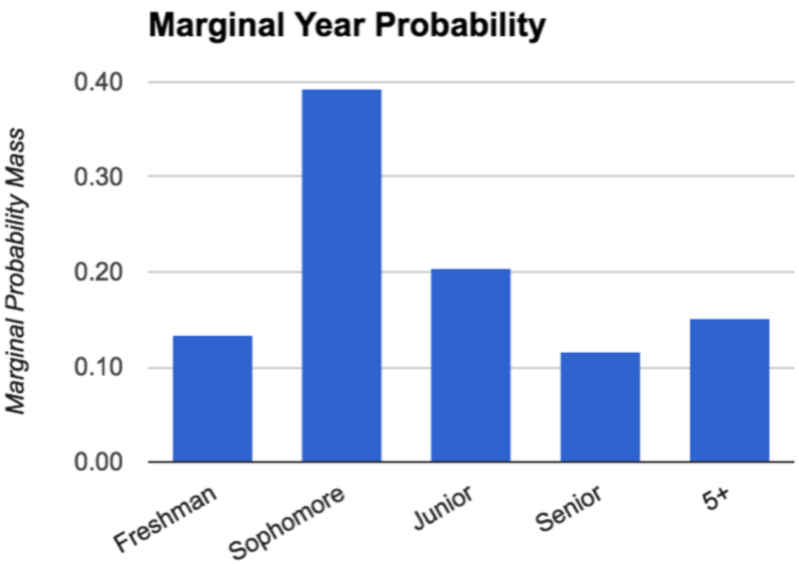
F(-infinity)

Joint Probability Table

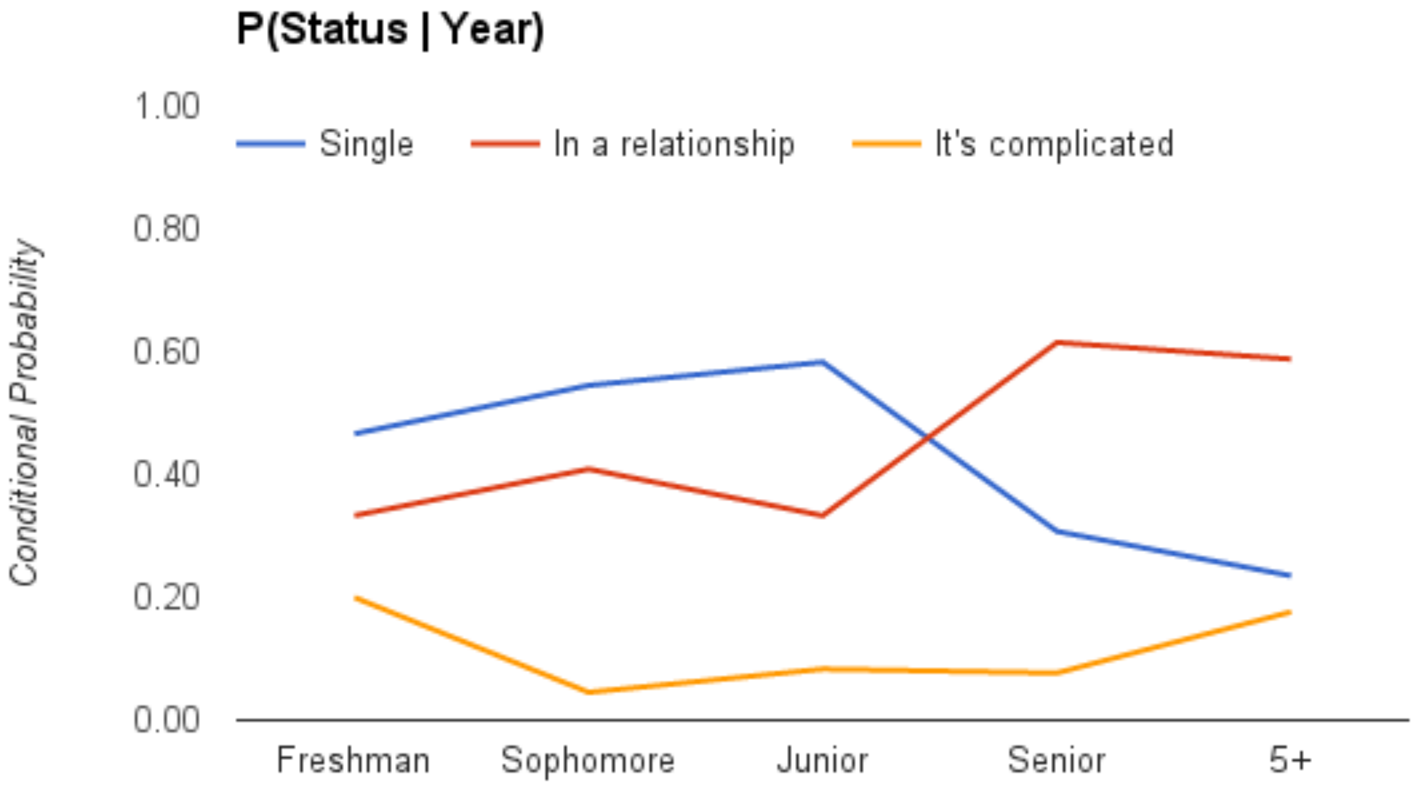


Joint Probability Table				
	Single	In a relationship	It's complicated	**Marginal Year**
Freshman	0.06	0.04	0.03	0.13
Sophomore	0.21	0.16	0.02	0.39
Junior	0.13	0.06	0.02	0.21
Senior	0.04	0.07	0.01	0.12
5+	0.04	0.09	0.03	0.15
Marginal Status	0.47	0.43	0.10	1.00





Conditional Probability



Joint Probability Density

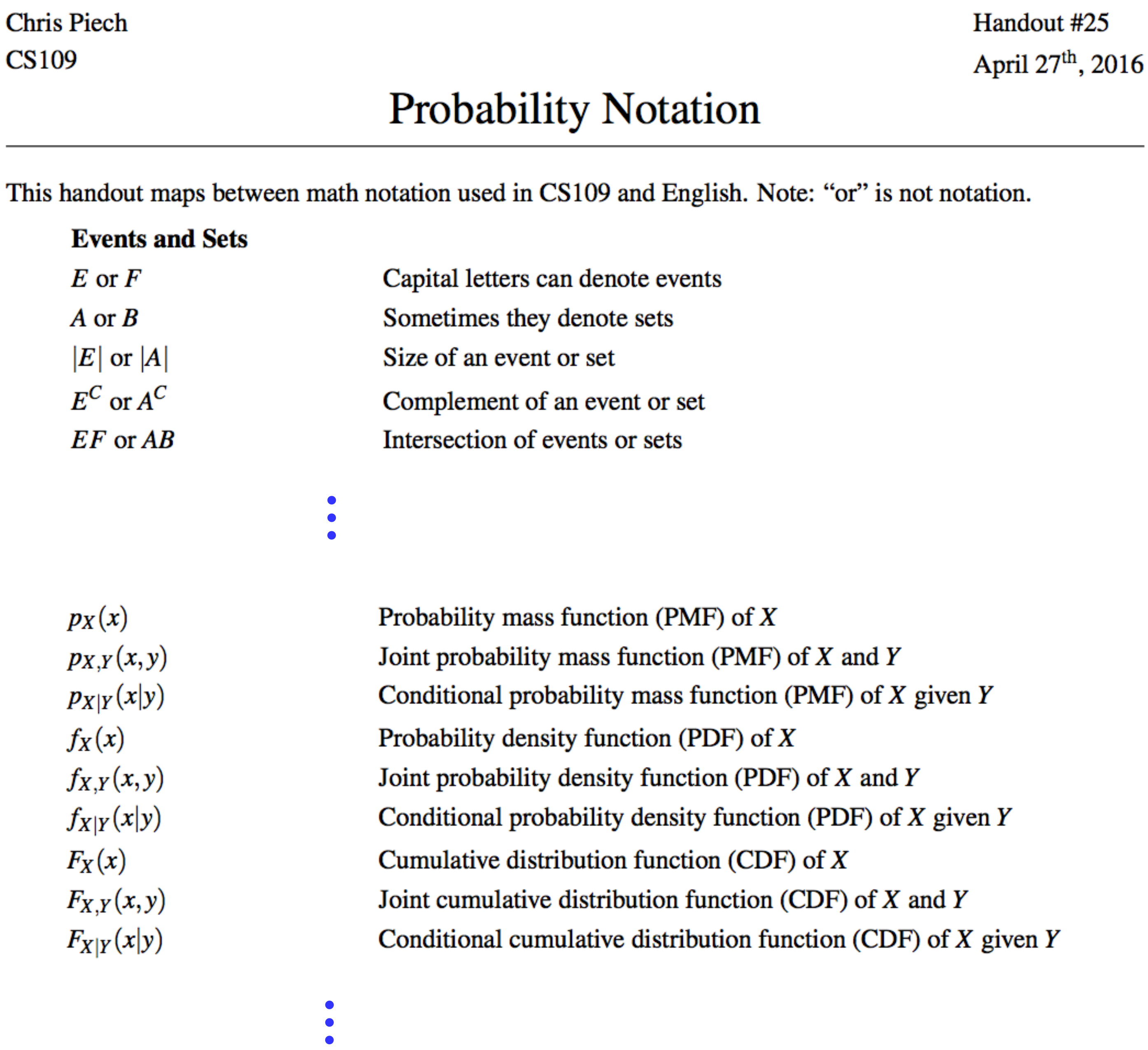
- X and Y are continuous RVs with Joint PDF:

$$f_{X,Y}(x,y) = \begin{cases} \frac{12}{5}x(2-x-y) & \text{where } 0 < x, y < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$f_{X|Y}(x \mid y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}$$

$$f_{X|N}(x \mid n) = \frac{p_{N|X}(n \mid x) f_X(x)}{p_N(n)}$$

Notation explosion!



Chris Piech
CS109

Handout #25
April 27th, 2016

Probability Notation

This handout maps between math notation used in CS109 and English. Note: "or" is not notation.

Events and Sets	
E or F	Capital letters can denote events
A or B	Sometimes they denote sets
$\|E\|$ or $\|A\|$	Size of an event or set
E^C or A^C	Complement of an event or set
EF or AB	Intersection of events or sets

⋮

$p_X(x)$	Probability mass function (PMF) of X
$p_{X,Y}(x,y)$	Joint probability mass function (PMF) of X and Y
$p_{X\|Y}(x\|y)$	Conditional probability mass function (PMF) of X given Y
$f_X(x)$	Probability density function (PDF) of X
$f_{X,Y}(x,y)$	Joint probability density function (PDF) of X and Y
$f_{X\|Y}(x\|y)$	Conditional probability density function (PDF) of X given Y
$F_X(x)$	Cumulative distribution function (CDF) of X
$F_{X,Y}(x,y)$	Joint cumulative distribution function (CDF) of X and Y
$F_{X\|Y}(x\|y)$	Conditional cumulative distribution function (CDF) of X given Y

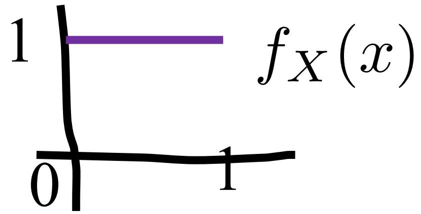
⋮

Flip a Coin With Unknown Probability

- Flip a coin (n + m) times, comes up with n heads
  - We don't know probability X that coin comes up heads
  - Our belief before flipping coins is that: X ~ Uni(0, 1)
  - Let N = number of heads
  - Given X = x, coin flips independent: (N | X) ~ Bin(n + m, x)

$$f_{X|N}(x|n) = \frac{\boxed{P(N = n|X = x)}\,\boxed{f_X(x)}}{P(N = n)} \quad 1$$

Binomial



$$= \frac{\binom{n+m}{n} x^n (1-x)^m}{P(N = n)}$$

Move terms around

$$= \boxed{\frac{\binom{n+m}{n}}{P(N = n)}} x^n (1-x)^m$$

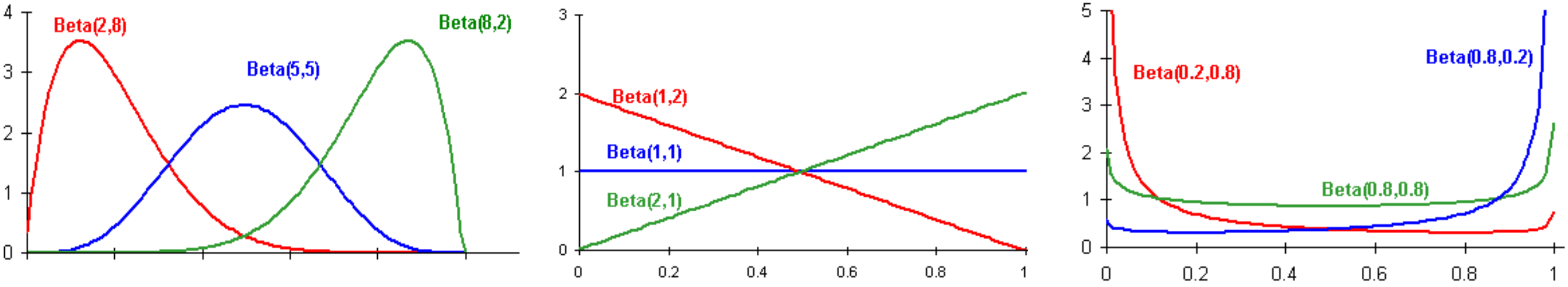
Constant

$$= \frac{1}{c} \cdot x^n (1-x)^m \qquad \text{where } c = \int_0^1 x^n (1-x)^m dx$$

Beta Random Variable

- X is a **Beta Random Variable**: X ~ Beta(*a*, *b*)
  - Probability Density Function (PDF): (where *a*, *b* > 0)

$$f(x) = \begin{cases} \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1} & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{where} \quad B(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}\,dx$$

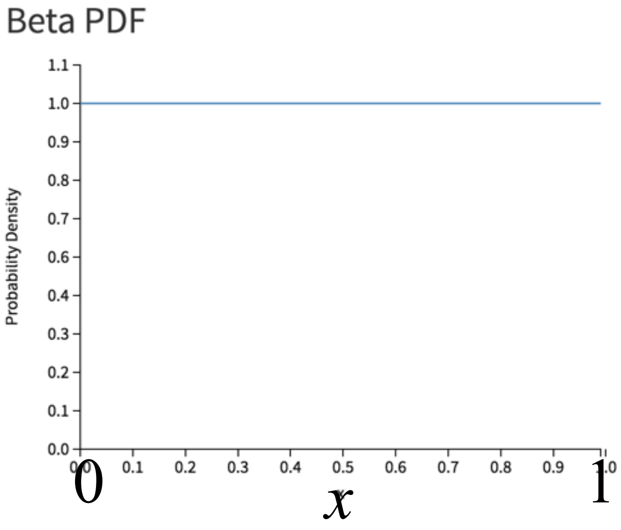


  - Symmetric when *a* = *b*
  - $E[X] = \frac{a}{a+b}$ $\qquad Var(X) = \frac{ab}{(a+b)^2(a+b+1)}$

Random Variable for *p*

No flips

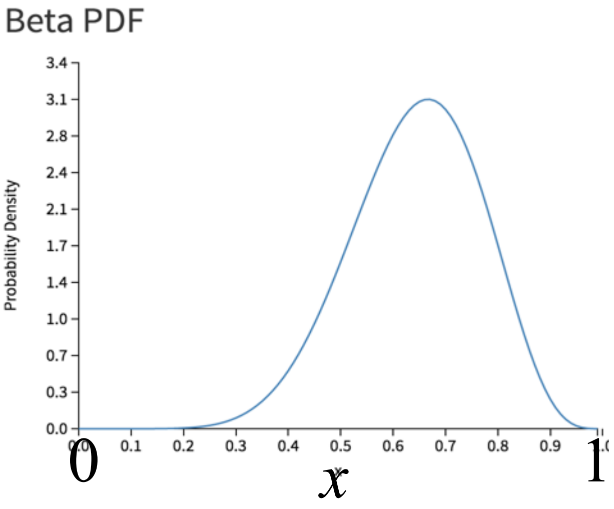
$f_{X|(0 \text{ heads, } 0 \text{ tails})}$



X ~ Beta(1,1)

12 flips, 8 heads

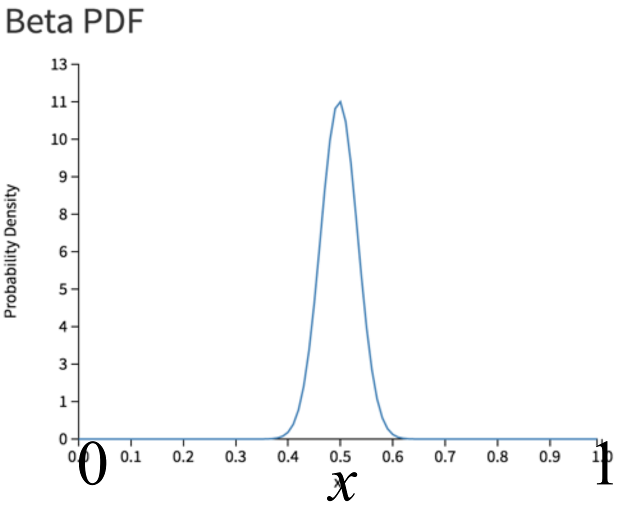
$f_{X|(8 \text{ heads, } 4 \text{ tails})}$



X ~ Beta(9,5)

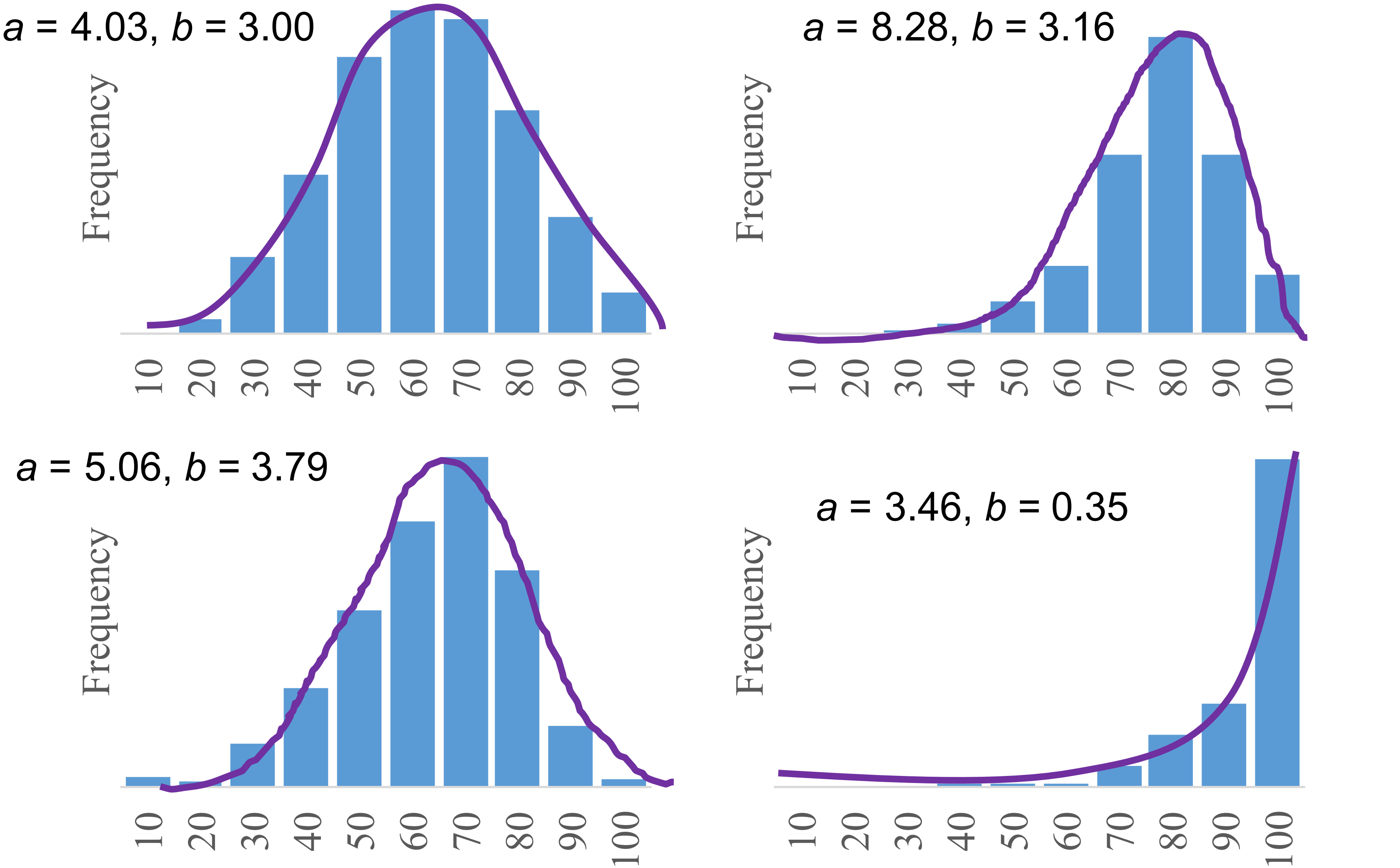
199 flips, 99 heads

$f_{X|(100 \text{ heads, } 101 \text{ tails})}$



X ~ Beta(101,102)

Assignment Grades


a = 4.03, *b* = 3.00

a = 8.28, *b* = 3.16

a = 5.06, *b* = 3.79

a = 3.46, *b* = 0.35

We have 2055 assignment distributions from grade scope

End Review

Expectation -> Covariance

Expected Values of Sums

Big deal lemma: first stated without proof

$$E[X + Y] = E[X] + E[Y]$$

Generalized: $$E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

Holds regardless of dependency between X_i's

Bool Was Cool

- Let E_1, E_2, ... E_n be events with indicator RVs X_i
  - If event E_i occurs, then X_i = 1, else X_i = 0
  - Recall $E[X_i] = P(E_i)$
  - Why?

$$E[X_i] = 0 \cdot (1 - P(E_i)) + 1 \cdot P(E_i)$$

Bernoulli aka Indicator Random Variables were studied extensively by George Bool

Bool died of being too cool

Expectation of Binomial

- Let Y ~ Bin(n, p)
  - *n* independent trials
  - Let X_i = 1 if *i*-th trial is "success", 0 otherwise
  - X_i ~ Ber(p) $E[X_i] = p$

$$Y = X_1 + X_2 + \cdots + X_n = \sum_{i=1}^{n} X_i$$

$$E[Y] = E[\sum_{i=1}^{n} X_i]$$

$$= \sum_{i=1}^{n} E[X_i]$$

$$= E[X_1] + E[X_2] + \ldots E[X_n]$$

$$= np$$

Expectation of Negative Binomial

- Let Y ~ NegBin(r, p)
  - Recall Y is number of trials until r "successes"
  - Let X_i = # of trials to get success after (*i* – 1)st success
  - X_i ~ Geo(p) (i.e., Geometric RV) $E[X_i] = \frac{1}{p}$

$$Y = X_1 + X_2 + \cdots + X_r = \sum_{i=1}^{r} X_i$$

$$\begin{aligned} E[Y] &= E[\sum_{i=1}^{r} X_i] \\ &= \sum_{i=1}^{r} E[X_i] \\ &= E[X_1] + E[X_2] + \ldots E[X_r] \\ &= \frac{r}{p} \end{aligned}$$

Hash Tables (aka Toy Collecting)

- Consider a hash table with *n* buckets
  - Each string equally likely to get hashed into any bucket
  - Let X = # strings to hash until each bucket ≥ 1 string
  - What is E[X]?
  - Let X_i = # of trials to get success after *i*-th success
    - where “success” is hashing string to previously empty bucket
    - After *i* buckets have ≥ 1 string, probability of hashing a string to an empty bucket is *p* = (*n* – *i*) / *n*
    - $P(X_i = k) = \frac{n-i}{n}\left(\frac{i}{n}\right)^{k-1}$ equivalently: $X_i \sim \text{Geo}((n-i)/n)$
    - $E[X_i] = 1/p = n/(n-i)$
  - $X = X_0 + X_1 + \ldots + X_{n-1} \Rightarrow E[X] = E[X_0] + E[X_1] + \ldots + E[X_{n-1}]$

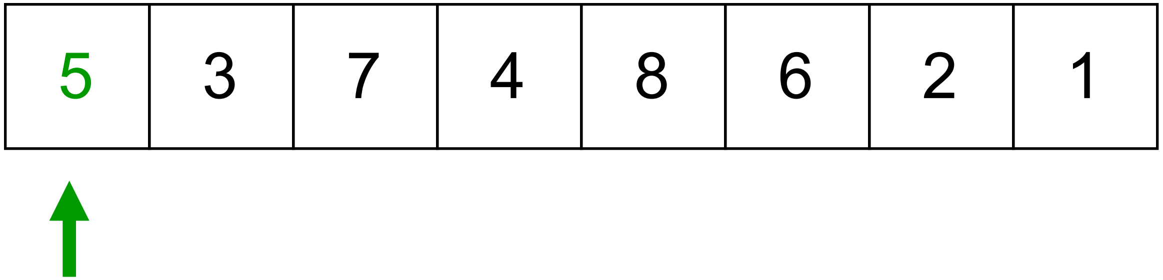
$$E[X] = \frac{n}{n} + \frac{n}{n-1} + \frac{n}{n-2} + \ldots + \frac{n}{1} = n\left[\frac{1}{n} + \frac{1}{n-1} + \ldots + 1\right] = O(n \log n)$$

This is your final answer

Let’s Do Some Sorting!

5	3	7	4	8	6	2	1

QuickSort



5	3	7	4	8	6	2	1

select
“pivot”

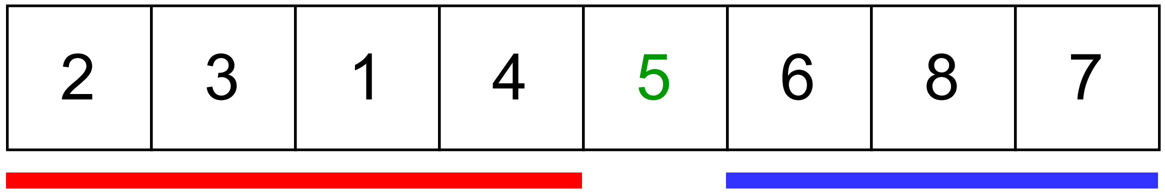
Recursive Insight

5	3	7	4	8	6	2	1

Partition array so:

- everything smaller than pivot is on left
- everything greater than or equal to pivot is on right
- pivot is in-between

Recursive Insight

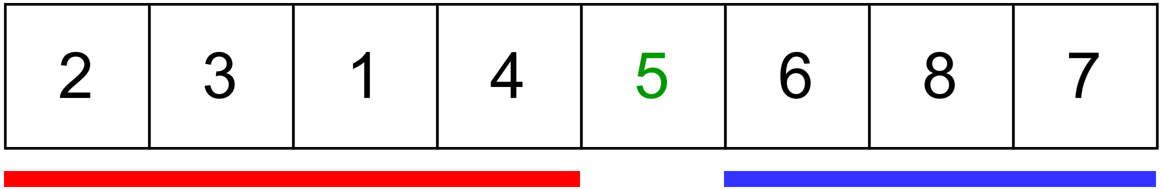


2	3	1	4	5	6	8	7

Partition array so:

- everything smaller than pivot is on left
- everything greater than or equal to pivot is on right
- pivot is in-between

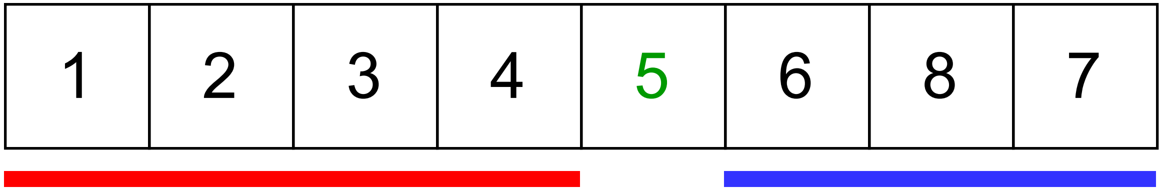
Recursive Insight



2	3	1	4	5	6	8	7

Now recursive sort “red” sub-array

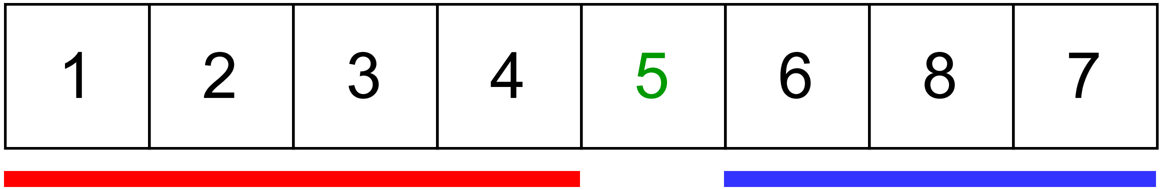
Recursive Insight



1	2	3	4	5	6	8	7

Now recursive sort “red” sub-array

Recursive Insight

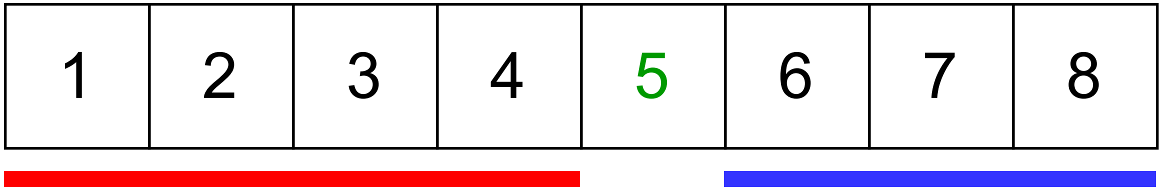


1	2	3	4	5	6	8	7

Now recursive sort “red” sub-array

Then, recursive sort “blue” sub-array

Recursive Insight



Now recursive sort “red” sub-array

Then, recursive sort “blue” sub-array

Recursive Insight

1	2	3	4	5	6	7	8

Everything is sorted!

```
void Quicksort(int arr[], int n)
{
   if (n < 2) return;

   int boundary = Partition(arr, n);

   // Sort subarray up to pivot
   Quicksort(arr, boundary);

   // Sort subarray after pivot to end
   Quicksort(arr + boundary + 1, n – boundary - 1);
}
```

“boundary” is the index of the pivot

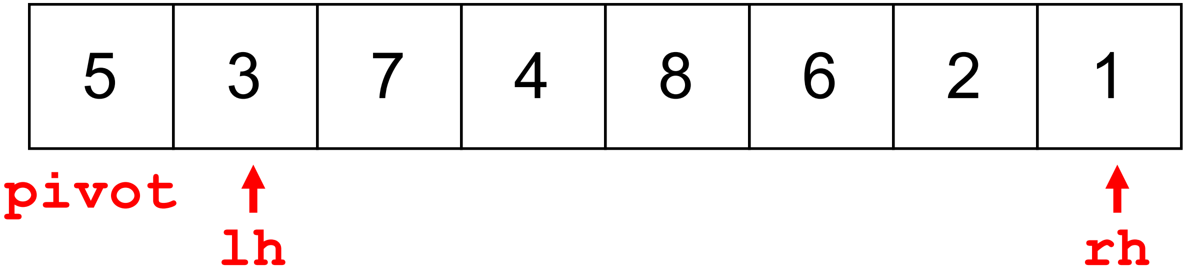
This is equal to the number of elements before pivot

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```

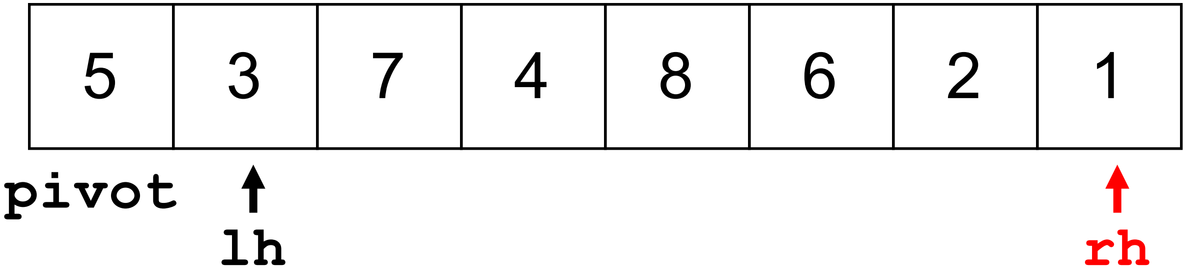
5	3	7	4	8	6	2	1

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



5	3	7	4	8	6	2	1

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



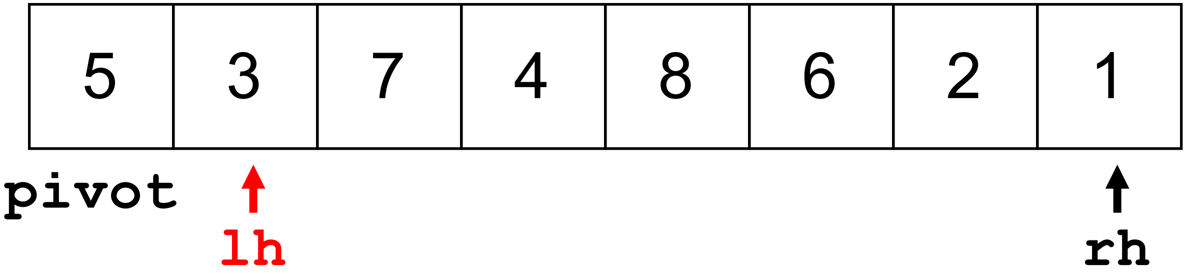
5	3	7	4	8	6	2	1

pivot

lh

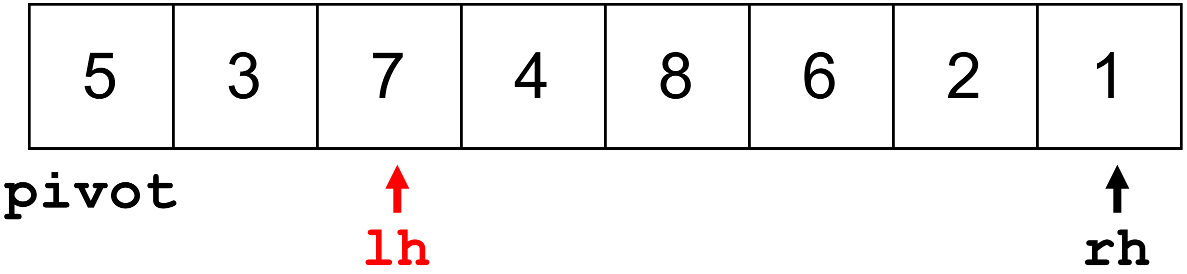
rh

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



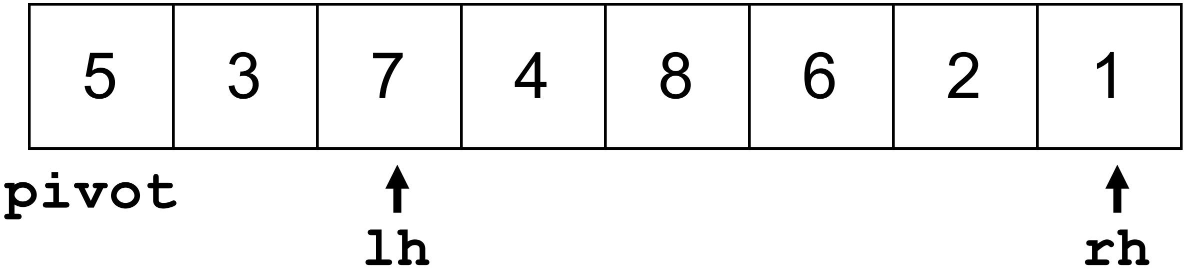
5	3	7	4	8	6	2	1

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
      while (lh < rh && arr[rh] >= pivot) rh--;
      while (lh < rh && arr[lh] < pivot) lh++;
      if (lh == rh) break;
      Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



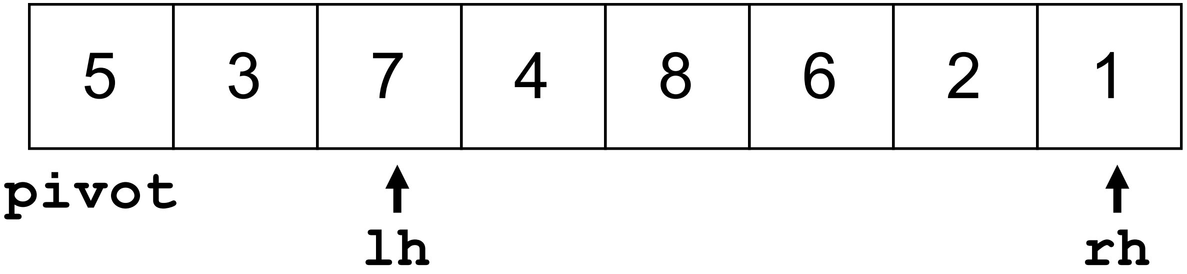
5	3	7	4	8	6	2	1
pivot		lh					rh

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
      while (lh < rh && arr[rh] >= pivot) rh--;
      while (lh < rh && arr[lh] < pivot) lh++;
      if (lh == rh) break;
      Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



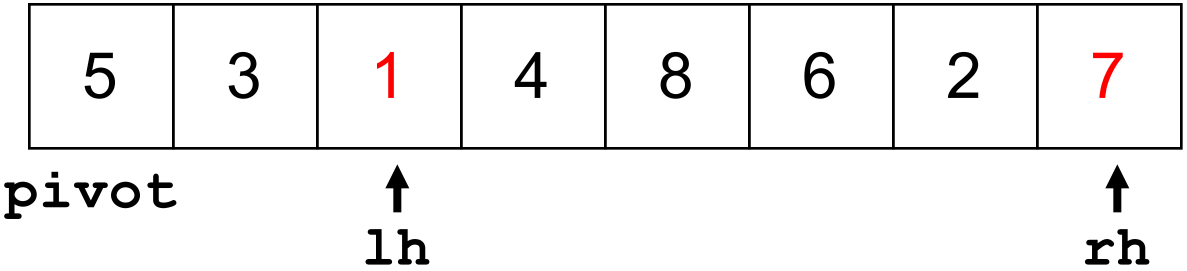
5	3	7	4	8	6	2	1

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



5	3	7	4	8	6	2	1
pivot		↑ lh					↑ rh

```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    int pivot = arr[0];
    while (true) {
      while (lh < rh && arr[rh] >= pivot) rh--;
      while (lh < rh && arr[lh] < pivot) lh++;
      if (lh == rh) break;
      Swap(arr[lh], arr[rh]);
    }
    if (arr[lh] >= pivot) return 0;
    Swap(arr[0], arr[lh]);
    return lh;
}
```



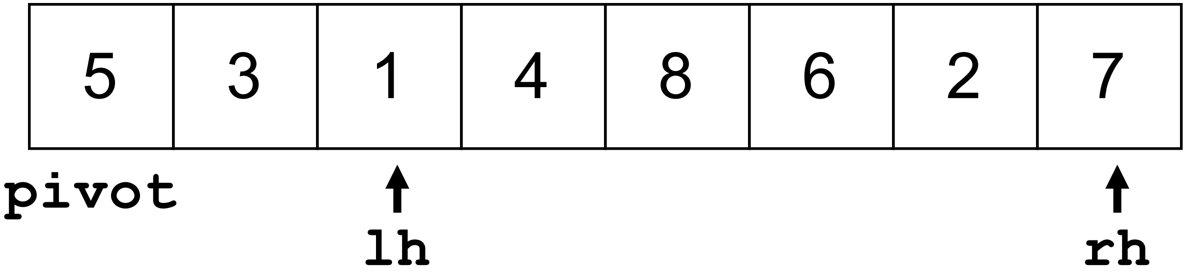
5	3	1	4	8	6	2	7

pivot

lh

rh

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



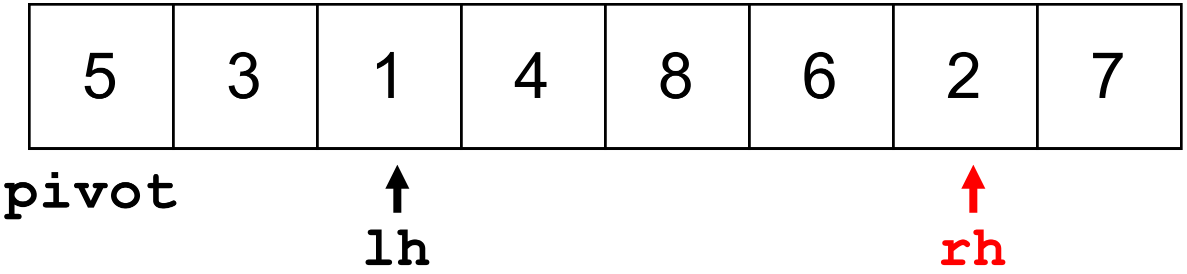
5	3	1	4	8	6	2	7

pivot

lh

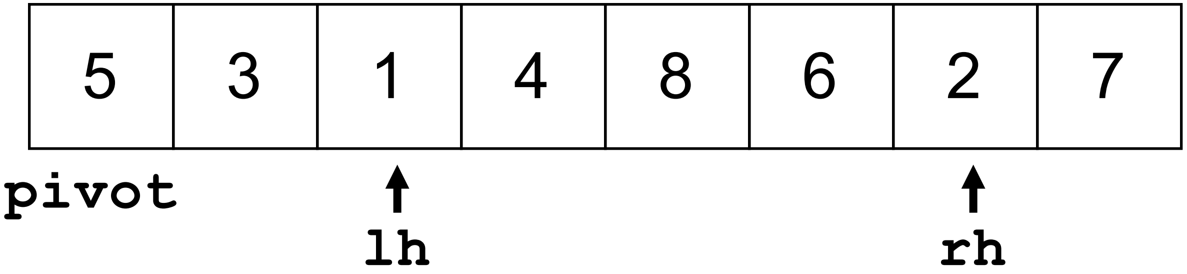
rh

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
      while (lh < rh && arr[rh] >= pivot) rh--;
      while (lh < rh && arr[lh] < pivot) lh++;
      if (lh == rh) break;
      Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



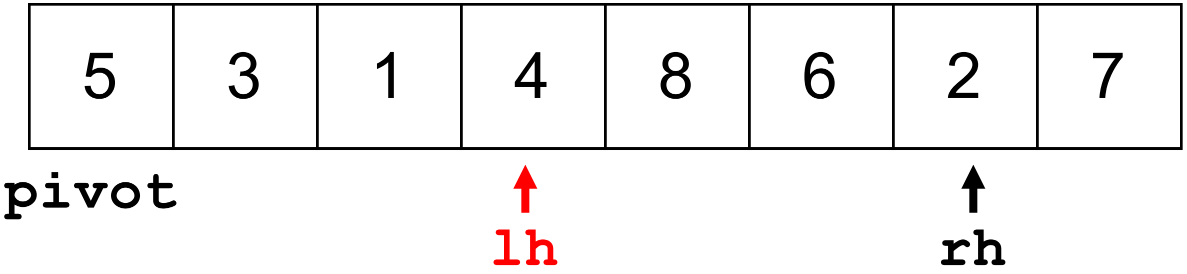
5	3	1	4	8	6	2	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



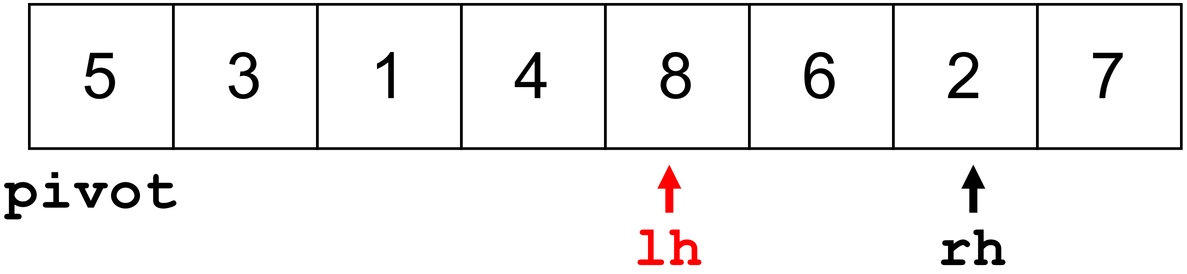
5	3	1	4	8	6	2	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



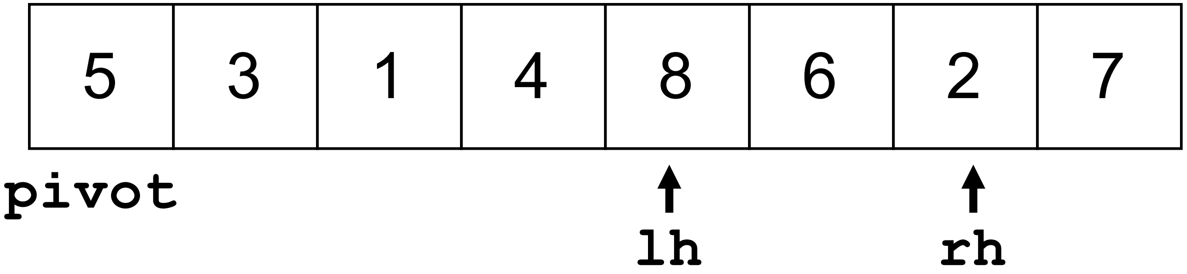
5	3	1	4	8	6	2	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



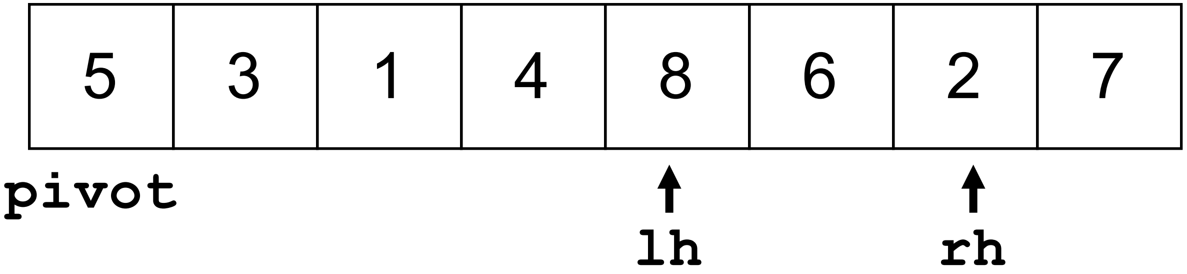
5	3	1	4	8	6	2	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



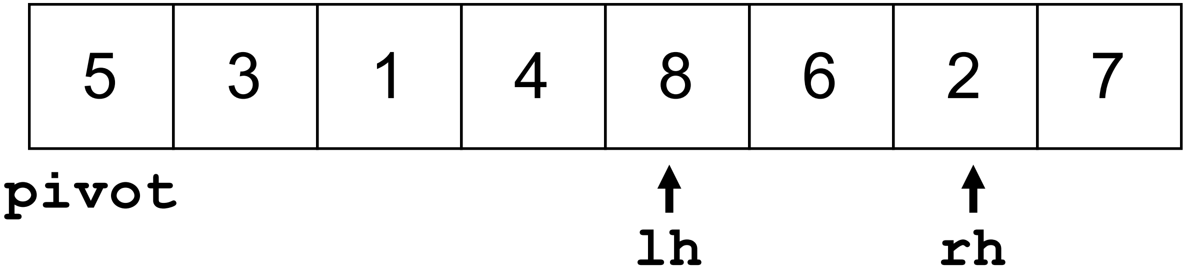
5	3	1	4	8	6	2	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



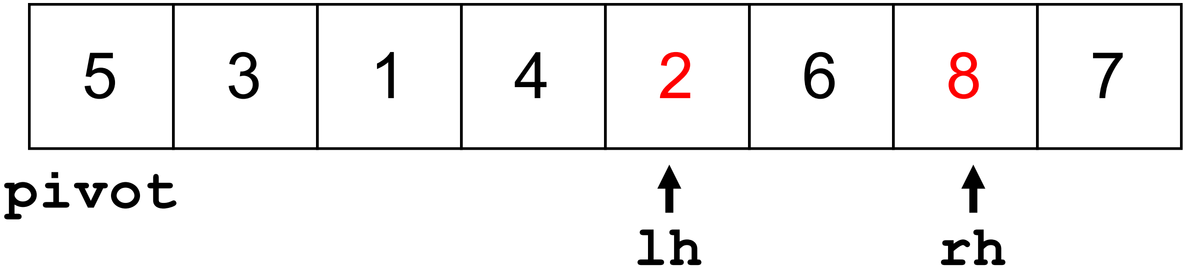
5	3	1	4	8	6	2	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



5	3	1	4	8	6	2	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



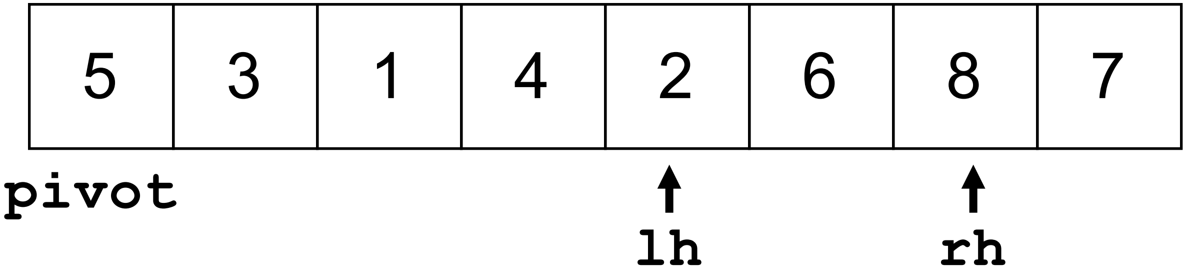
5	3	1	4	2	6	8	7

pivot

lh

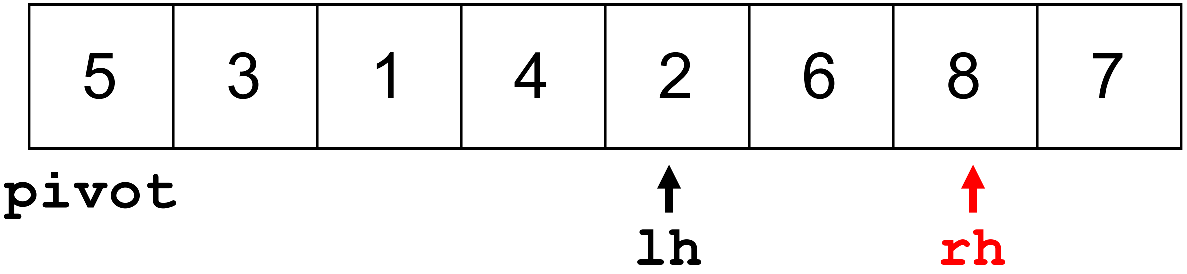
rh

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```

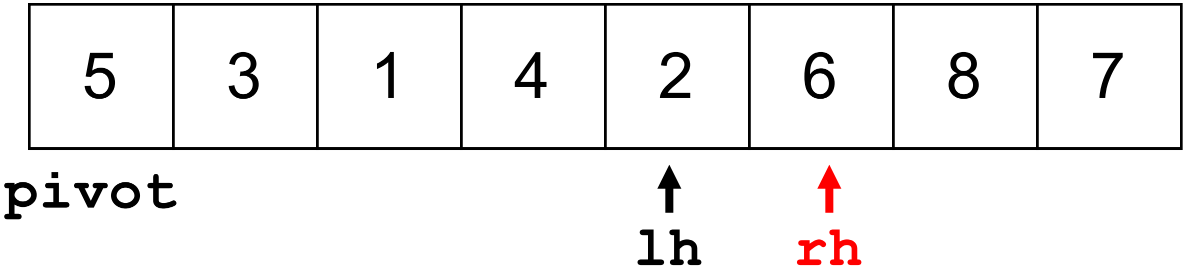


5	3	1	4	2	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```

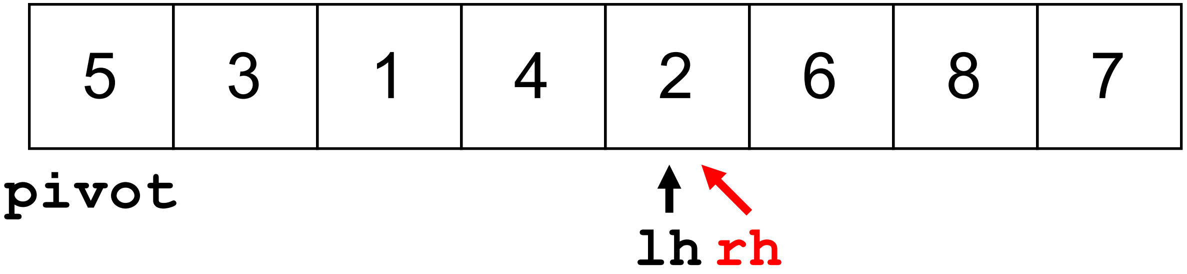


```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    int pivot = arr[0];
    while (true) {
      while (lh < rh && arr[rh] >= pivot) rh--;
      while (lh < rh && arr[lh] < pivot) lh++;
      if (lh == rh) break;
      Swap(arr[lh], arr[rh]);
    }
    if (arr[lh] >= pivot) return 0;
    Swap(arr[0], arr[lh]);
    return lh;
}
```



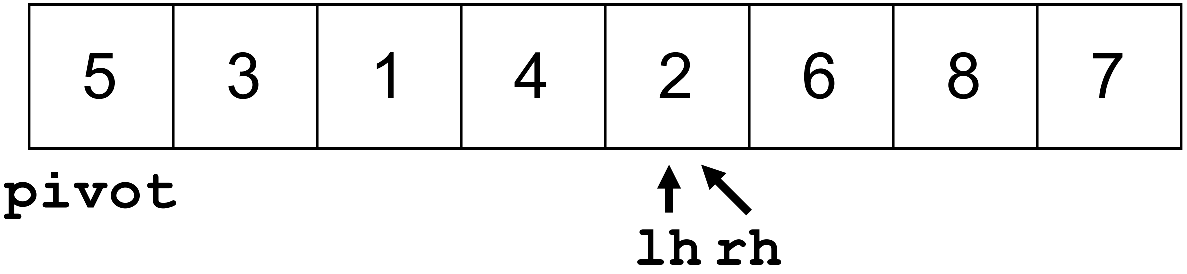
5	3	1	4	2	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



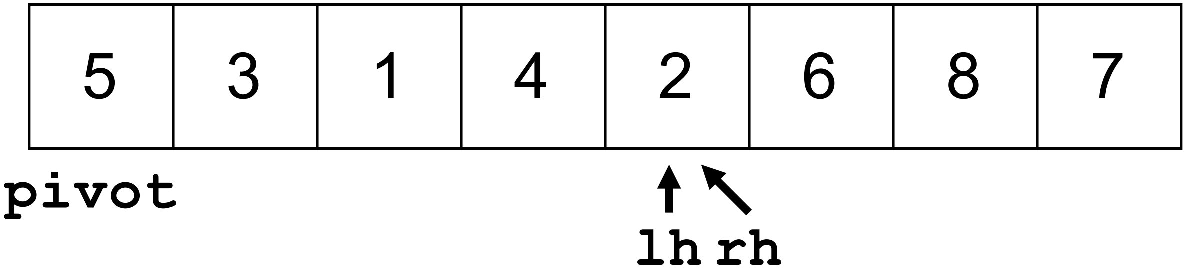
5	3	1	4	2	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



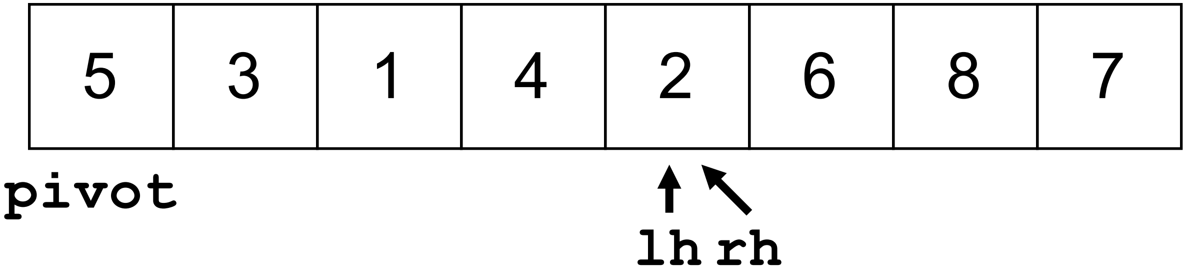
5	3	1	4	2	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
      while (lh < rh && arr[rh] >= pivot) rh--;
      while (lh < rh && arr[lh] < pivot) lh++;
      if (lh == rh) break;
      Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



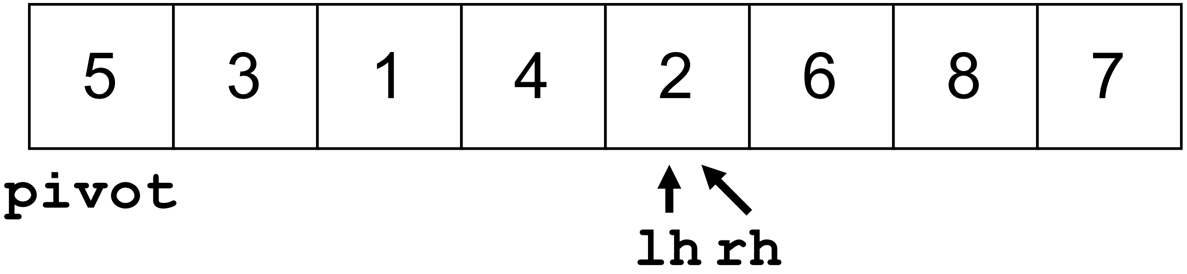
5	3	1	4	2	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



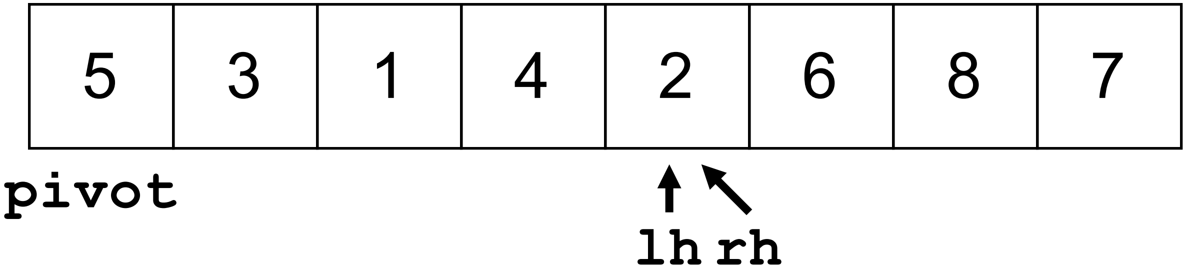
5	3	1	4	2	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
      while (lh < rh && arr[rh] >= pivot) rh--;
      while (lh < rh && arr[lh] < pivot) lh++;
      if (lh == rh) break;
      Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



5	3	1	4	2	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```

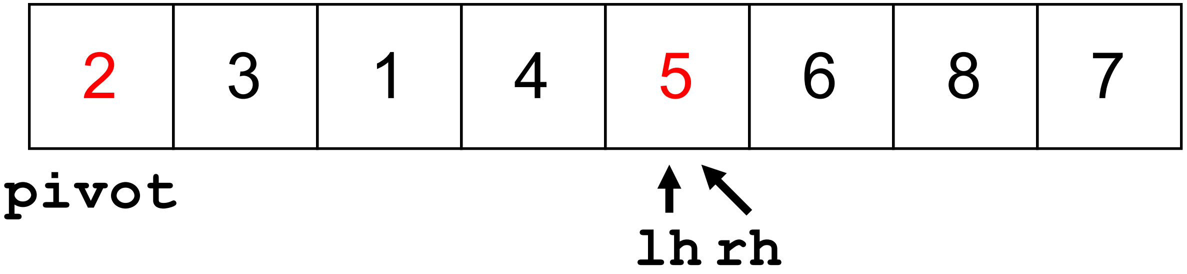


5	3	1	4	2	6	8	7

pivot

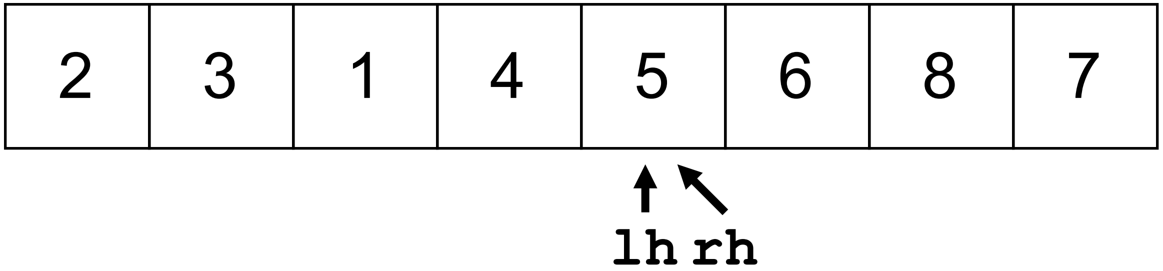
lh rh

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



2	3	1	4	5	6	8	7

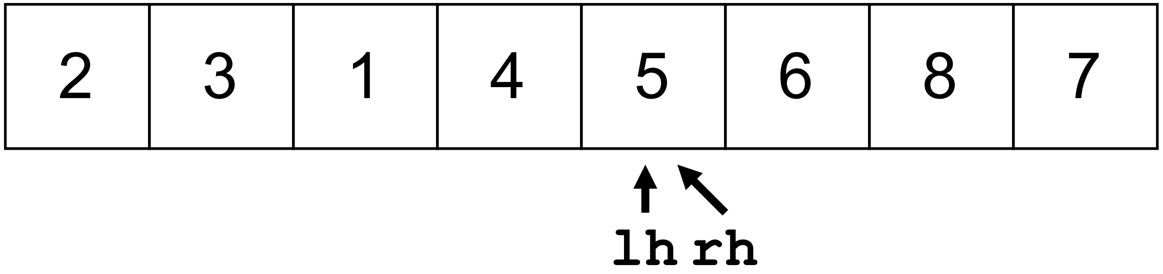
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```



2	3	1	4	5	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```

Returns 4 (index of pivot)



2	3	1	4	5	6	8	7

```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   int pivot = arr[0];
   while (true) {
     while (lh < rh && arr[rh] >= pivot) rh--;
     while (lh < rh && arr[lh] < pivot) lh++;
     if (lh == rh) break;
     Swap(arr[lh], arr[rh]);
   }
   if (arr[lh] >= pivot) return 0;
   Swap(arr[0], arr[lh]);
   return lh;
}
```

- Complexity of algorithm determined by number of comparisons made to pivot

Complexity QuickSort

- QuickSort is O(n log n), where n = # elems to sort
  - But in "worst case" it can be O(n^2)
  - Worst case occurs when every time pivot is selected, it is maximal or minimal remaining element
- What is P(QuickSort worst case)?
  - On each recursive call, pivot = max/min element, so we are left with n – 1 elements for next recursive call
  - 2 possible "bad" pivots (max/min) on each recursive call

$$P(\text{Worst case}) = \frac{2}{n} \cdot \frac{2}{n-1} \cdot \ldots \cdot \frac{2}{2} = \frac{2^{n-1}}{n!}$$

  - Saw similar behavior for BSTs on problem set #1
    - P(Worst case) gets small very fast as *n* grows!

Expected Running Time of QuickSort

- Let X = # comparisons made when sorting n elems
  - E[X] gives us expected running time of algorithm
  - Given $V_1, V_2, \ldots, V_n$ in random order to sort
  - Let $Y_1, Y_2, \ldots, Y_n$ be $V_1, V_2, \ldots, V_n$ in sorted order
  - Let $I_{a,b} = 1$ if Y_a and Y_b are compared, 0 otherwise
  - Order where $Y_b > Y_a$, so we have: $X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} I_{a,b}$

Expected Running Time of QuickSort

Aside: $X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} I_{a,b}$

When a = 1 $\quad I_{1,2} + I_{1,3} + \ldots + I_{1,n}$

When a = 2 $\quad + I_{2,3} + \ldots + I_{2,n}$

When a = n-1 $\quad + I_{n-1,n}$

Contains a comparison between each i and j
(where i does not equal j)
exactly once

Expected Running Time of QuickSort

- Let X = # comparisons made when sorting n elems
  - E[X] gives us expected running time of algorithm
  - Given $V_1, V_2, \ldots, V_n$ in random order to sort
  - Let $Y_1, Y_2, \ldots, Y_n$ be $V_1, V_2, \ldots, V_n$ in sorted order
  - Let $I_{a,b}$ = 1 if Y_a and Y_b are compared, 0 otherwise
  - Order where $Y_b > Y_a$, so we have: $X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} I_{a,b}$

$$E[X] = E\left[\sum_{a=1}^{n-1} \sum_{b=a+1}^{n} I_{a,b}\right] = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[I_{a,b}] = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} P(Y_a \text{ and } Y_b \text{ ever compared})$$

What is the probability that Y_a and Y_b are compared?

Lets Imagine Our Array in Sorted Order

		Y_a		Y_b	
1	3	5	7	9	11
Y_1	Y_2	Y_3	Y_4	Y_5	Y_6

Whether or not they are compared depends on pivot choice

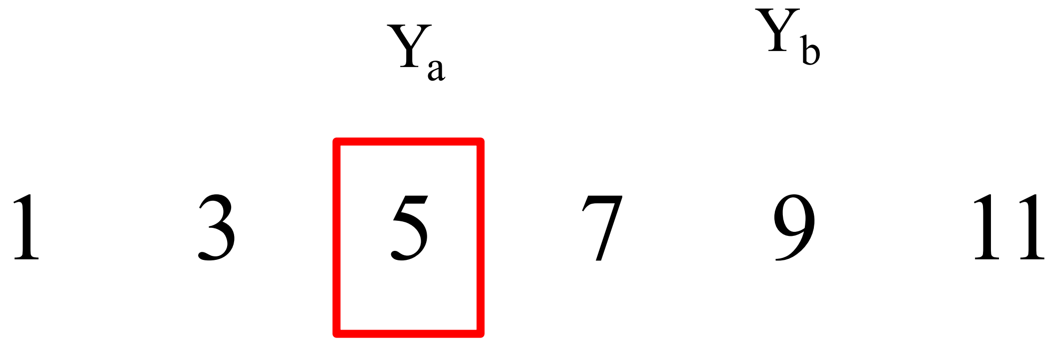
Lets Imagine Our Array in Sorted Order

Y_a Y_b

1 3 5 7 9 11

Whether or not they are compared depends on pivot choice

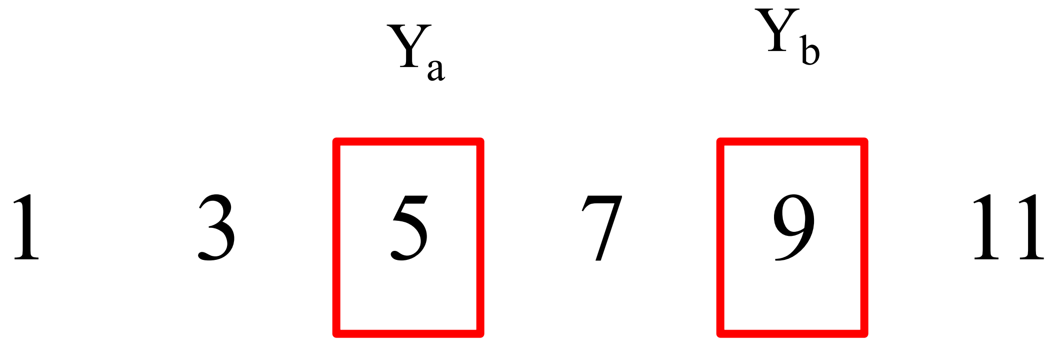
P(Y_a and Y_b ever compared)



Consider pivot choice: Y_a

They are compared

P(Y_a and Y_b ever compared)



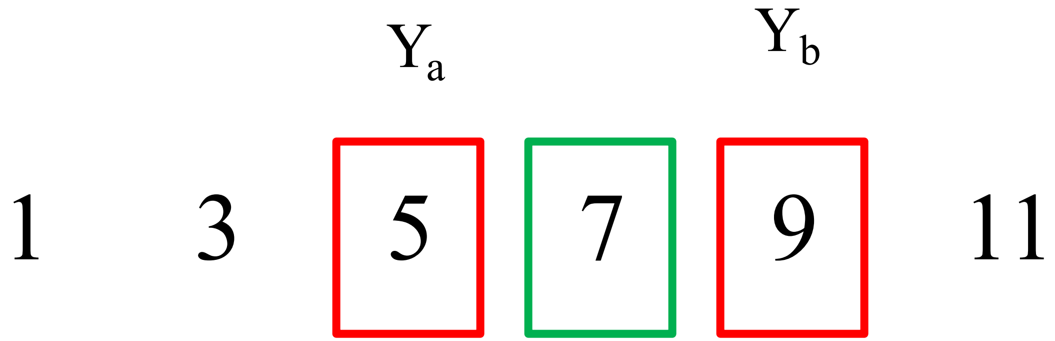
Y_a Y_b

1 3 $\boxed{5}$ 7 $\boxed{9}$ 11

Consider pivot choice: Y_b

They are compared

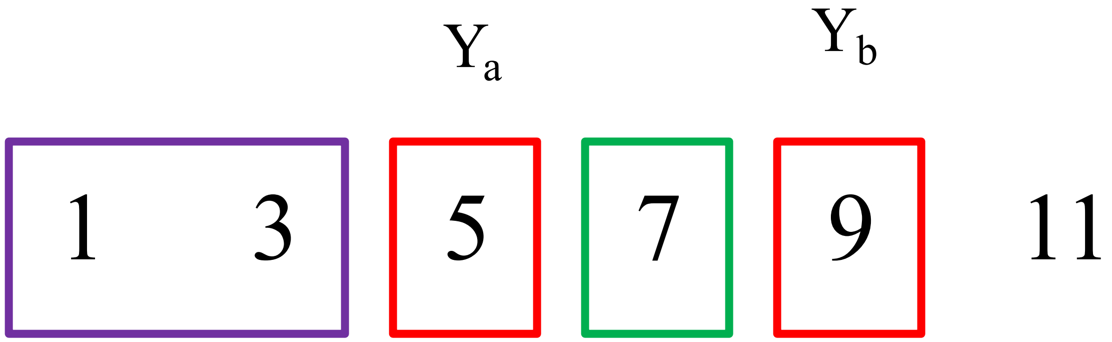
P(Y_a and Y_b ever compared)



Consider pivot choice: 7

They are **not** compared

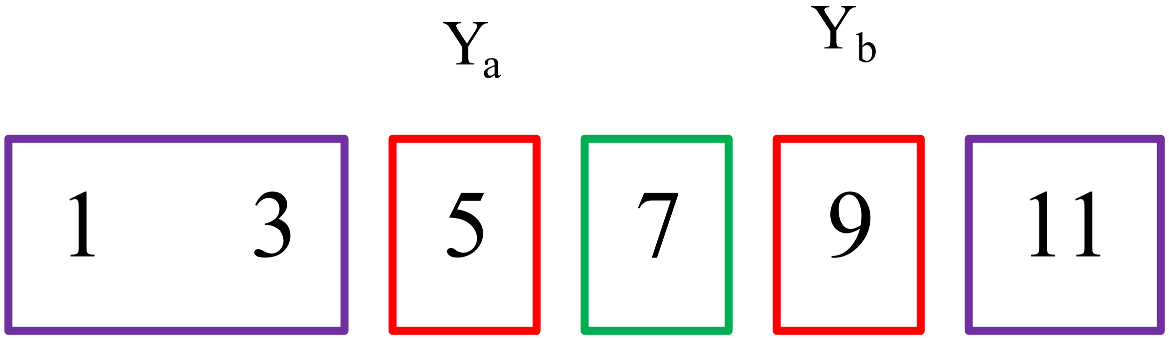
P(Y_a and Y_b ever compared)



Consider pivot choice: < Y_a

Whether or not they are compared depends on future pivots

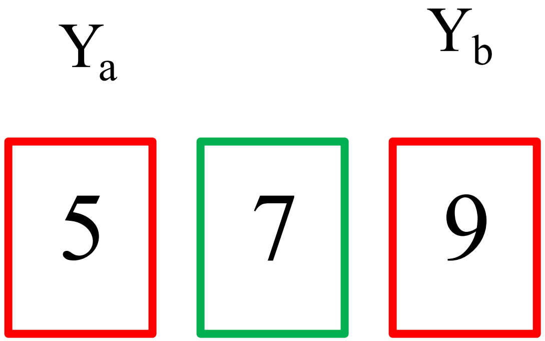
P(Y_a and Y_b ever compared)



Consider pivot choice: > Y_b

Whether or not they are compared depends on future pivots

P(Y_a and Y_b ever compared)



Are Y_a and Y_b compared?

Keep repeating pivot choice until you get a pivot
In the range [Y_a, Y_b] inclusive

P(Ya and Yb ever compared)

- Consider when Y_a and Y_b are directly compared
  - We only care about case where pivot chosen from set: $\{Y_a, Y_{a+1}, Y_{a+2}, \ldots, Y_b\}$
  - From that set either Y_a and Y_b must be selected as pivot (with equal probability) in order to be compared
  - So,

$$P(Y_a \text{ and } Y_b \text{ ever compared}) = \frac{2}{b-a+1}$$

$$E[X] = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} P(Y_a \text{ and } Y_b \text{ ever compared}) = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b-a+1}$$

Bring it on Home (i.e. Solve the Sum)

$$E[X] = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b-a+1}$$

$$\sum_{b=a+1}^{n} \frac{2}{b-a+1} \approx \int_{a+1}^{n} \frac{2}{b-a+1}\, db$$

Recall: $\int \frac{1}{x}\, dx = \ln(x)$

Thanks Riemann

$$= 2\ln(b-a+1)\Big|_{a+1}^{n} = 2\ln(n-a+1) - 2\ln(2)$$

$\approx 2\ln(n-a+1)$ for large n

$$E[X] \approx \sum_{a=1}^{n-1} 2\ln(n-a+1) \approx 2\int_{a=1}^{n-1} \ln(n-a+1)\, da$$

Let $y = n-a+1$

$$= -2\int_{y=n}^{2} \ln(y)\, dy$$

Recall:

$$\int \ln(x)\, dx = x\ln(x) - x$$

$$= -2(y\ln(y) - y)\Big|_{n}^{2}$$

$$= -2[(2\ln(2)-2) - (n\ln(n) - n)] \approx 2n\ln(n) - 2n = O(n\log n)$$

Ahhh ☺

Variance from first principles

Indicators: Now with pair-wise flavor!

- Recall I_i is indicator variable for event A_i when:

$$I_i = \begin{cases} 1 & \text{if } A_i \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

  - Let X = # of events that occur: $X = \sum_{i=1}^{n} I_i$

$$E[X] = E\left[\sum_{i=1}^{n} I_i\right] = \sum_{i=1}^{n} E[I_i] = \sum_{i=1}^{n} P(A_i)$$

- Now consider pair of events $A_i\ A_j$ occurring
  - $I_i\ I_j$ = 1 if both events A_i and A_j occur, 0 otherwise
  - Number of pairs of events that occur is $\binom{X}{2} = \sum_{i<j} I_i I_j$

I remember you!

From Event Pairs to Variance

- Expected number of pairs of events:

$$E\left[\binom{X}{2}\right] = E\left[\sum_{i<j} I_i I_j\right] = \sum_{i<j} E[I_i I_j] = \sum_{i<j} P(A_i A_j)$$

$$E\left[\frac{X(X-1)}{2}\right] = \frac{1}{2}(E[X^2] - E[X]) = \sum_{i<j} P(A_i A_j)$$

$$E[X^2] - E[X] = 2\sum_{i<j} P(A_i A_j) \quad \Rightarrow \quad E[X^2] = 2\sum_{i<j} P(A_i A_j) + E[X]$$

- Recall: Var(X) = E[X^2] – (E[X])^2

$$\begin{aligned}\mathrm{Var}(X) &= 2\sum_{i<j} P(A_i A_j) + E[X] - (E[X])^2 \\ &= 2\sum_{i<j} P(A_i A_j) + \sum_{i=1}^{n} P(A_i) - \left(\sum_{i=1}^{n} P(A_i)\right)^2\end{aligned}$$

Let's Try it With the Binomial

- X ~ Bin(n, p) $$E[X] = \sum_{i=1}^{n} P(A_i) = np$$
  - Each trial: X_i ~ Ber(p) $$E[X_i] = p$$
  - Let event A_i = trial *i* is success (i.e., $X_i = 1$)

$$\sum_{i<j} P(A_i A_j) = \sum_{i<j} p^2 = \binom{n}{2} p^2 = \frac{n(n-1)}{2} p^2$$

$$\text{Var}(X) = 2\sum_{i<j} P(A_i A_j) + E[X] - (E[X])^2$$

$$\text{Var}(X) = 2\frac{n(n-1)}{2}p^2 + np - (np)^2$$

Substitute in for each term

$$= n^2p^2 - np^2 + np - n^2p^2$$

$$= np - np^2 = np(1-p)$$

Expand and simplify

Computer Cluster Utilization

- Computer cluster with *k* servers
  - Requests independently go to server *i* with probability p_i
  - Let event A_i = server *i* receives no requests
  - X = # of events $A_1, A_2, \ldots A_k$ that occur
  - Y = # servers that receive ≥ 1 request = *k* – X
  - E[Y] after first *n* requests?
  - Since requests independent: $P(A_i) = (1 - p_i)^n$

$$E[X] = \sum_{i=1}^{k} P(A_i) = \sum_{i=1}^{k} (1 - p_i)^n$$

$$E[Y] = k - E[X] = k - \sum_{i=1}^{k} (1 - p_i)^n$$

when $p_i = \frac{1}{k}$ for $1 \le i \le k,\ E[Y] = k - \sum_{i=1}^{k} (1 - \frac{1}{k})^n = k\left(1 - (1 - \frac{1}{k})^n\right)$

amazon

* 52% of Amazons Profits

**As profitable as Amazon’s North America commerce operations

http://www.zdnet.com/article/amazons-finds-its-profit-horse-in-aws-why-its-so-disruptive-to-its-old-guard/

Computer Cluster Utilization

- Computer cluster with *k* servers
  - Requests independently go to server *i* with probability p_i
  - Let event A_i = server *i* receives no requests
  - X = # of events $A_1, A_2, \ldots A_k$ that occur
  - Y = # servers that receive ≥ 1 request = *k* – X
  - Var(Y) after first *n* requests? (= $(-1)^2$ Var(X) = Var(X))
  - Independent requests: $P(A_i A_j) = (1 - p_i - p_j)^n, \quad i \neq j$

$$\mathrm{Var}(X) = 2\sum_{i<j}(1-p_i-p_j)^n + E[X] - (E[X])^2 \qquad E[X] = \sum_{i=1}^{k}(1-p_i)^n$$

$$= 2\sum_{i<j}(1-p_i-p_j)^n + \sum_{i=1}^{k}(1-p_i)^n - \left(\sum_{i=1}^{k}(1-p_i)^n\right)^2 = \mathrm{Var}(Y)$$

Computer Cluster = Coupon Collecting

- Computer cluster with k servers
  - Requests independently go to server i with probability p_i
  - Let event A_i = server i receives no requests
  - X = # of events A_1, A_2, … A_k that occur
  - Y = # servers that receive ≥ 1 request = k – X
- This is really another "Coupon Collector" problem
  - Each server is a "coupon type"
  - Request to server = collecting a coupon of that type
- Hash table version
  - Each server is a bucket in table
  - Request to server = string gets hashed to that bucket

A lemma to start your weekend

Product of Expectations

- Say X and Y are independent random variables, and *g*(•) and *h*(•) are real-valued functions

$$E[g(X)h(Y)] = E[g(X)]E[h(Y)]$$

  - Proof:

$$E[g(X)h(Y)] = \int_{y=-\infty}^{\infty} \int_{x=-\infty}^{\infty} g(x)h(y) f_{X,Y}(x,y)\,dx\,dy$$

$$= \int_{y=-\infty}^{\infty} \int_{x=-\infty}^{\infty} g(x)h(y) f_X(x) f_Y(y)\,dx\,dy$$

$$= \int_{x=-\infty}^{\infty} g(x) f_X(x)\,dx \cdot \int_{y=-\infty}^{\infty} h(y) f_Y(y)\,dy$$

$$= E[g(X)]E[h(Y)]$$

Next time:
Covariance lemmanade