Great Expectations

Chris Piech (and Dickens)
CS109, Stanford University

Course Mean

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

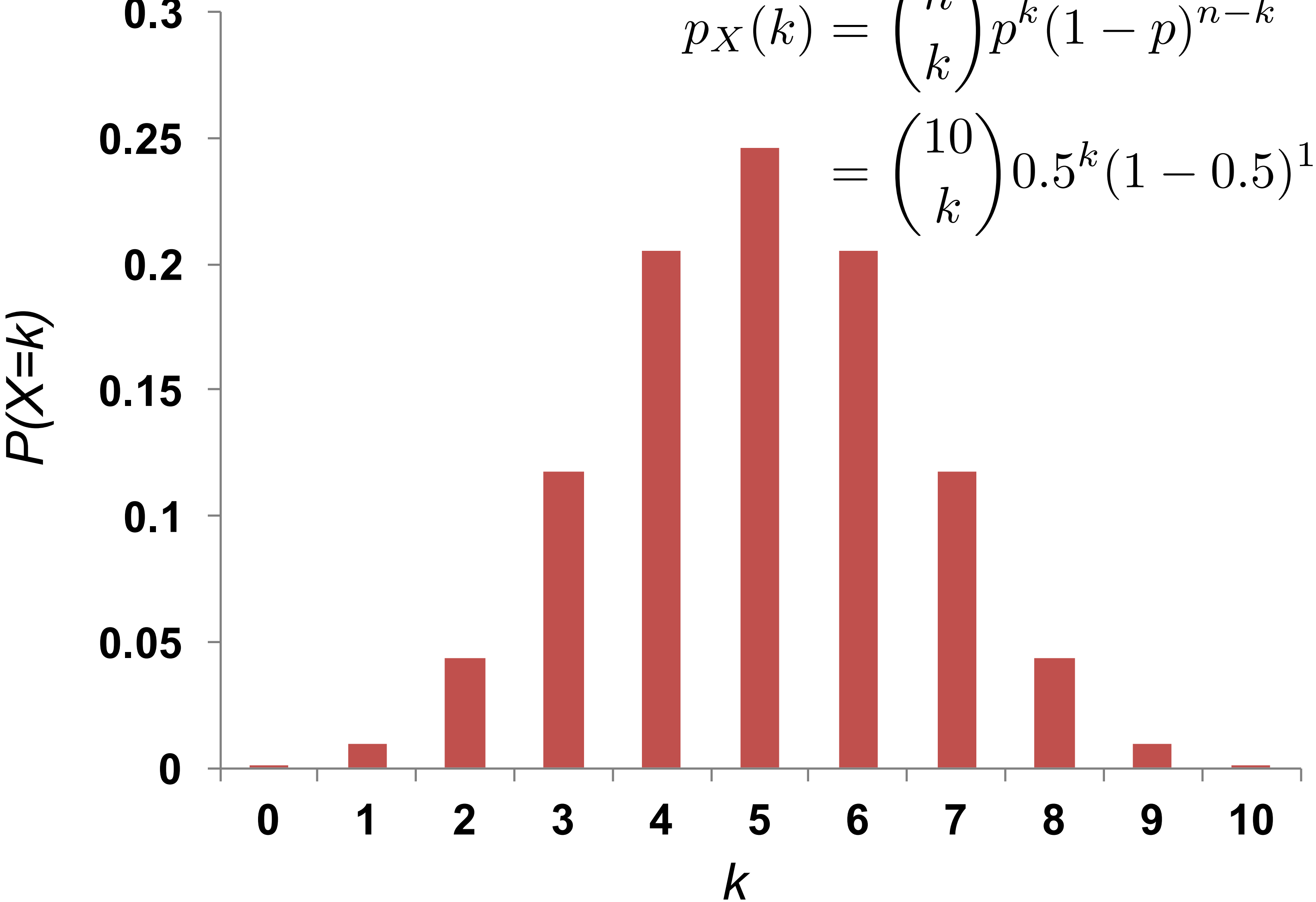
This is actual midpoint of course
(Just wanted you to know)

French Elections

Review

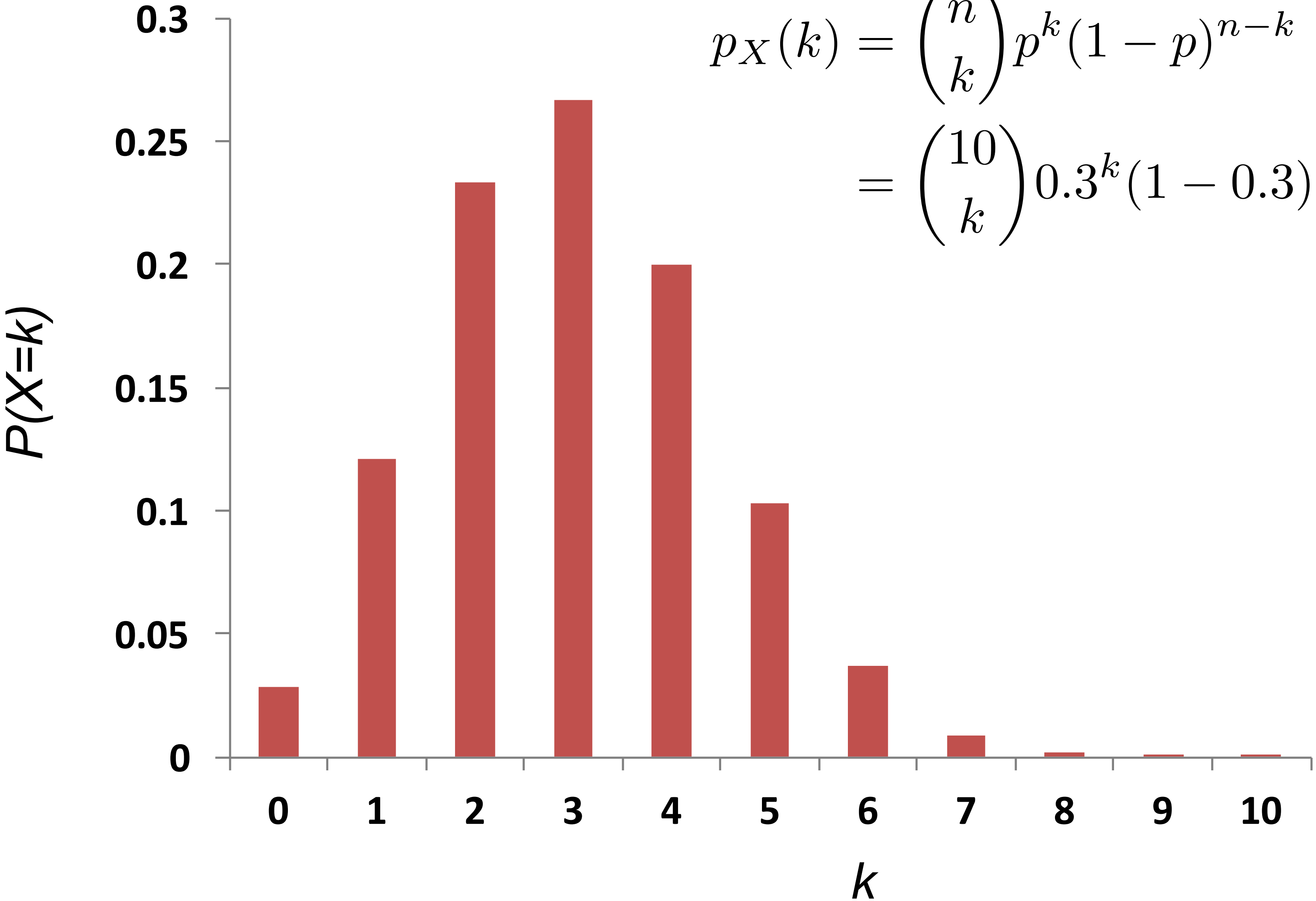
PMF for X ~ Bin(10, 0.5)

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



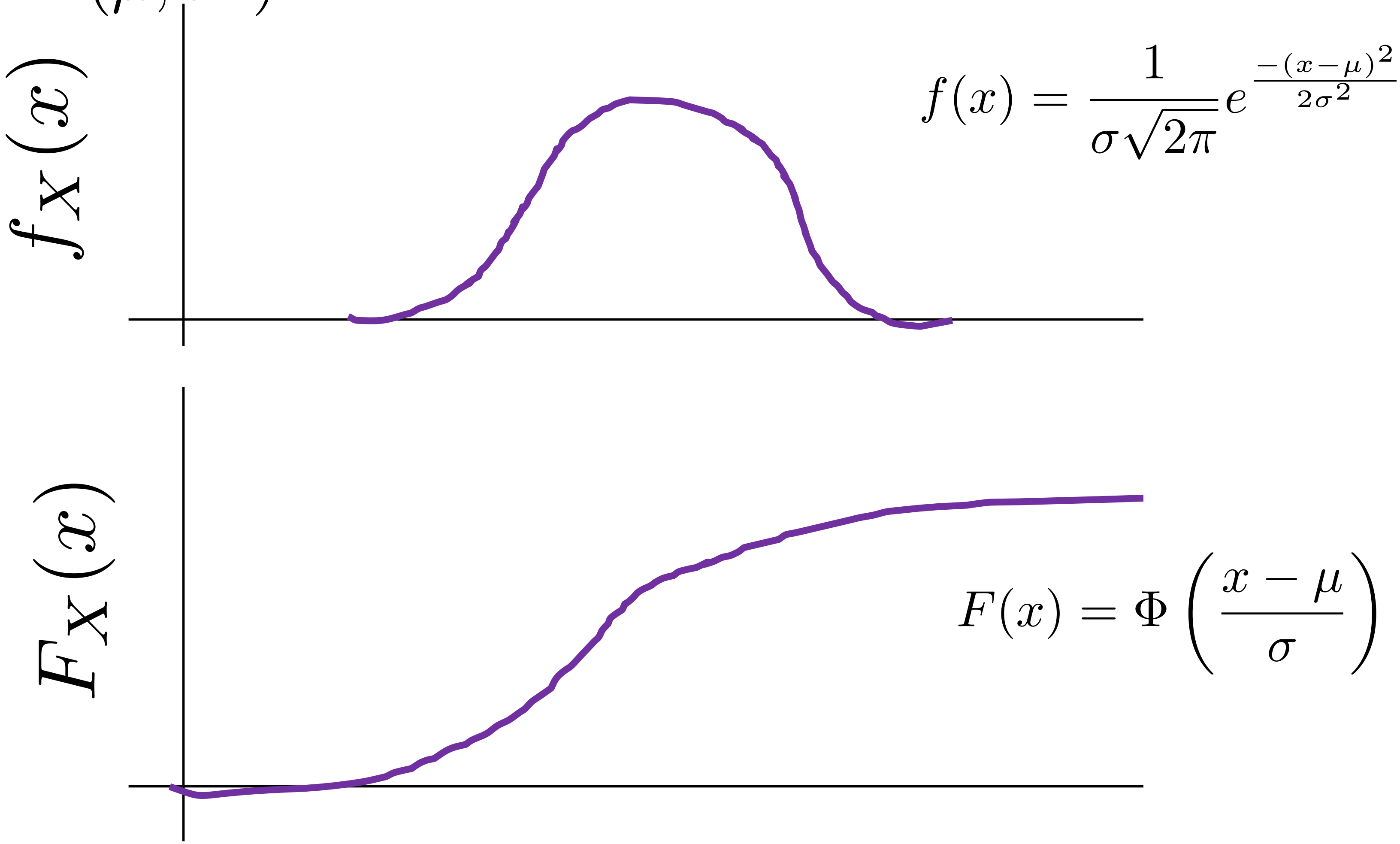
PMF for X ~ Bin(10, 0.3)

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



PDF and CDF of a Normal

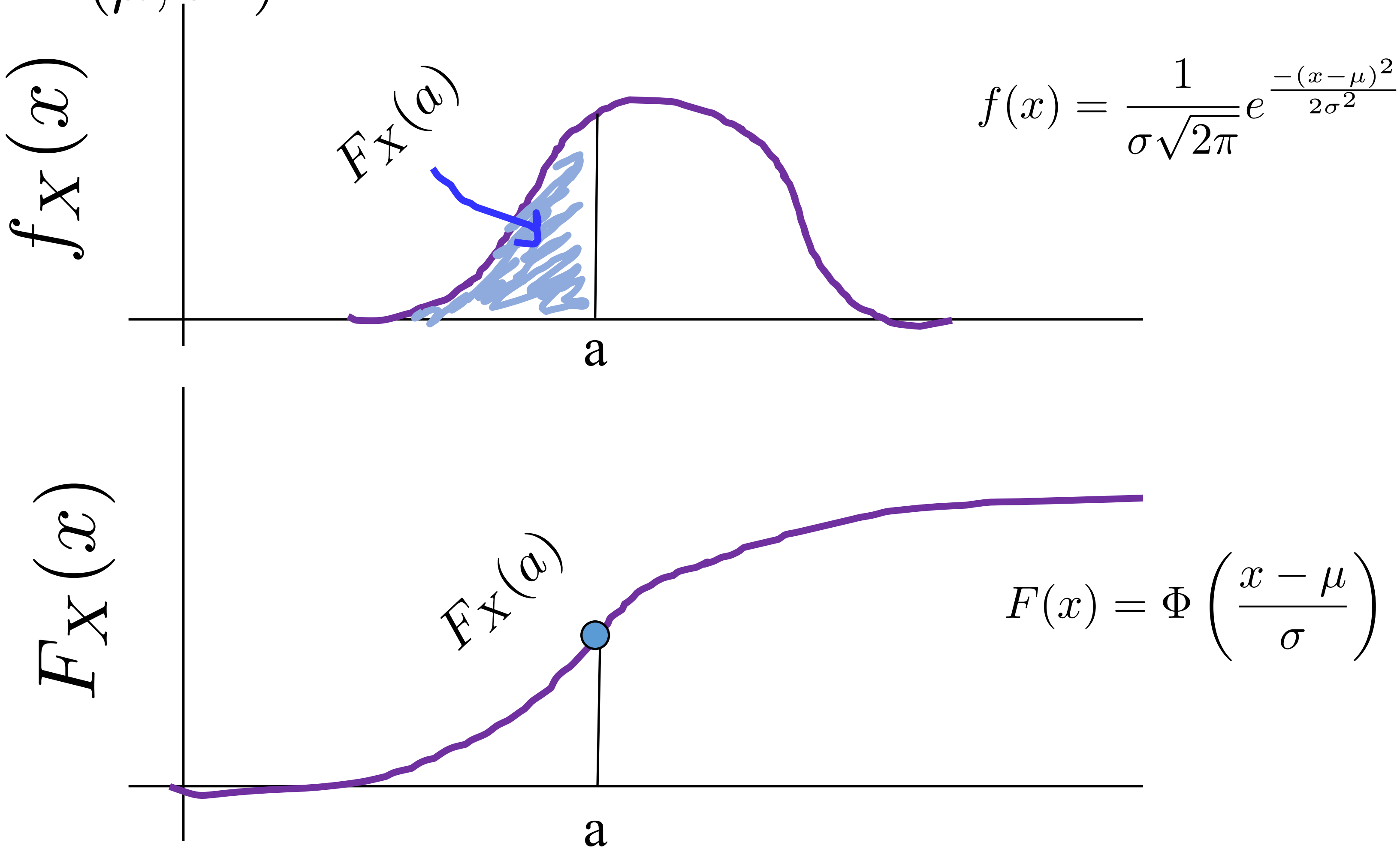
$$X \sim N(\mu, \sigma^2)$$



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

PDF and CDF of a Normal

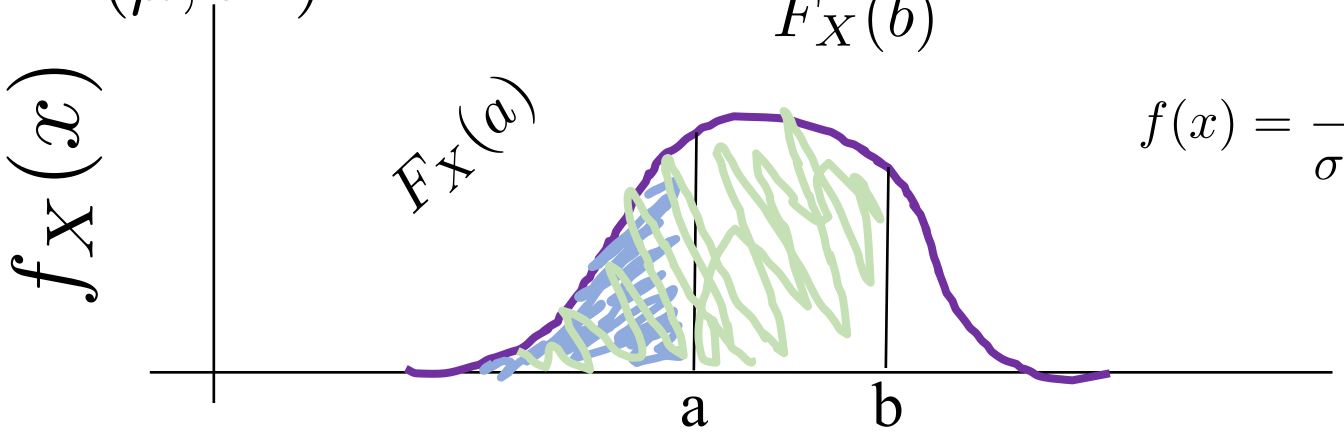
$X \sim N(\mu, \sigma^2)$



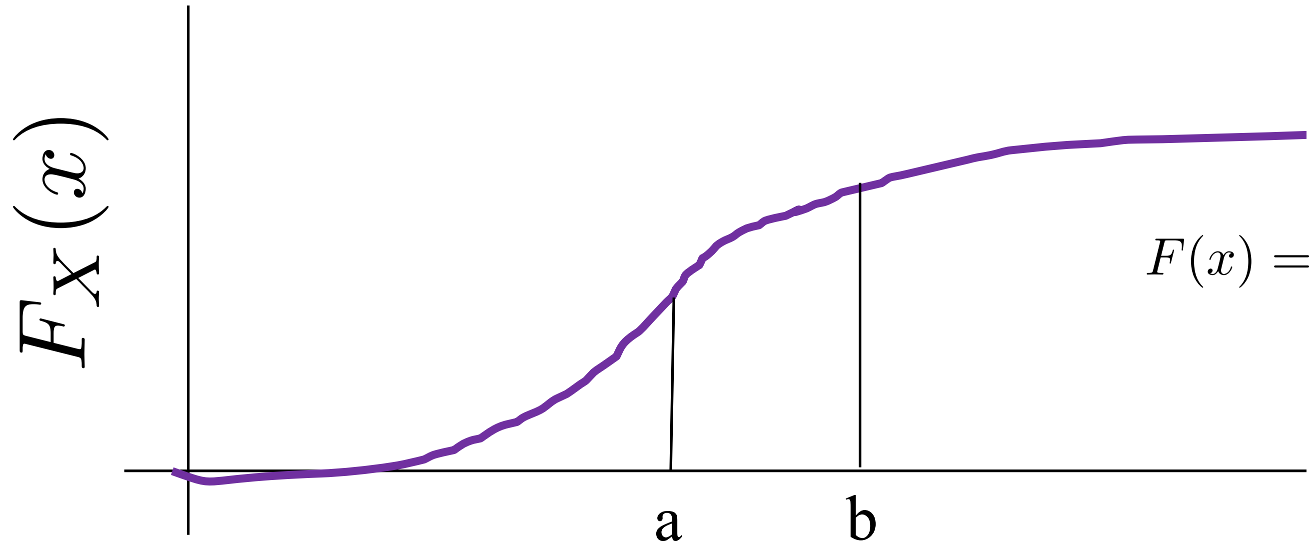
A CDF is the integral from –infinity to x of the PDF

PDF and CDF of a Normal

$X \sim N(\mu, \sigma^2)$



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

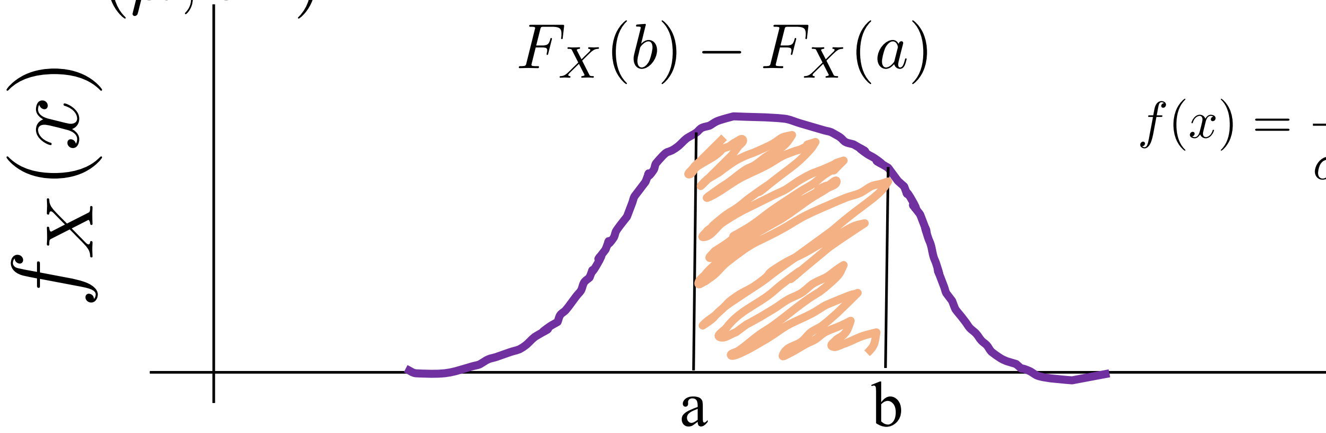


$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

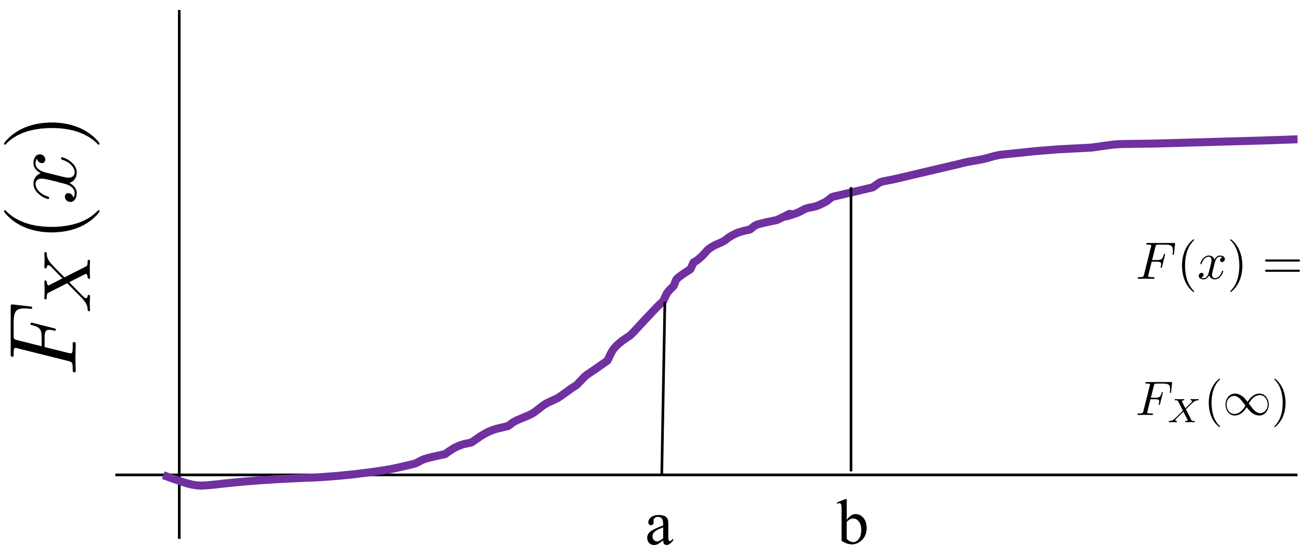
A CDF is the integral from –infinity to x of the PDF

PDF and CDF of a Normal

$$X \sim N(\mu, \sigma^2)$$



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



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

$$F_X(\infty) = 1$$

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

F(-infinity)

Expected Values of Sums

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

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

End Review

From Expectation to Correlation via Covariance

Covariance

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) $\quad E[X_i] = p$

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

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

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

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

$$= np$$

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) $E[X_i] = \frac{1}{p}$

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

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

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

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

$$= \frac{r}{p}$$

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

When stuck, brainstorm about random variables

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

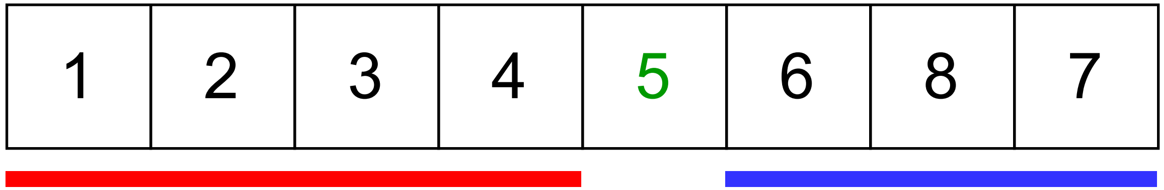
Now recursive sort "red" sub-array

Recursive Insight

1	2	3	4	5	6	8	7

Now recursive sort “red” sub-array

Recursive Insight



Now recursive sort “red” sub-array

Then, recursive sort “blue” sub-array

Recursive Insight

1	2	3	4	5	6	7	8

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

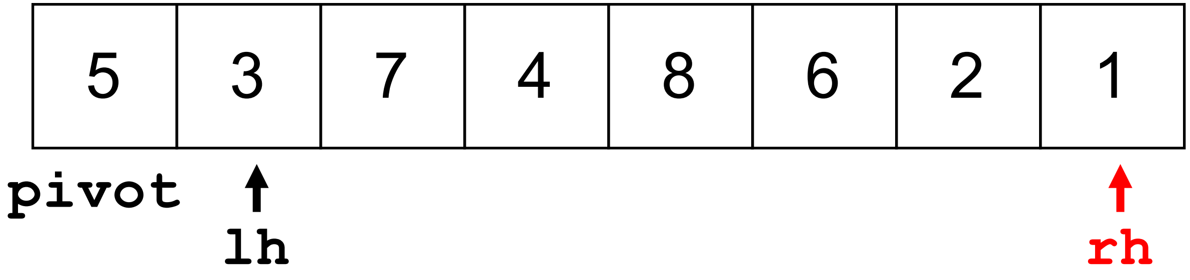
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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
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;
}
```



```
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

pivot

lh

rh

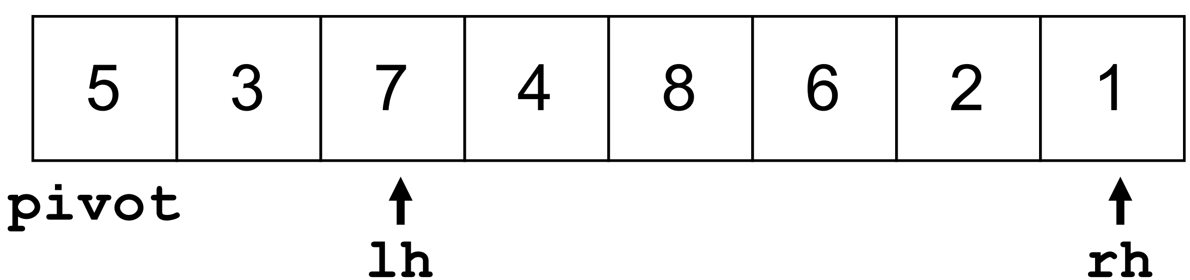
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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
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;
}
```



```
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

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

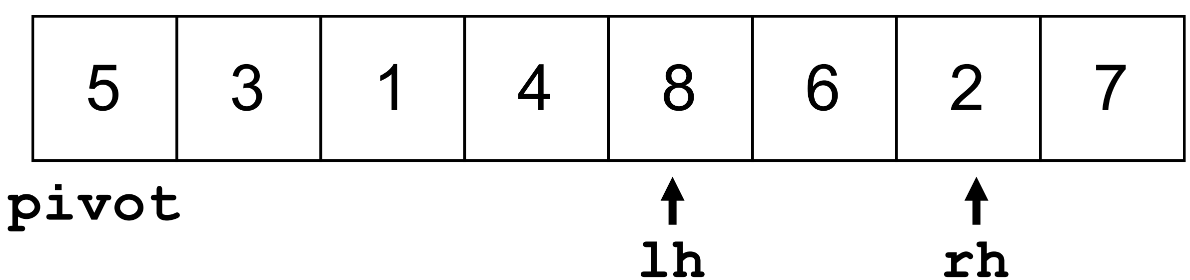
pivot

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;
}
```



```
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	2	6	8	7
pivot				↑ lh		↑ rh	

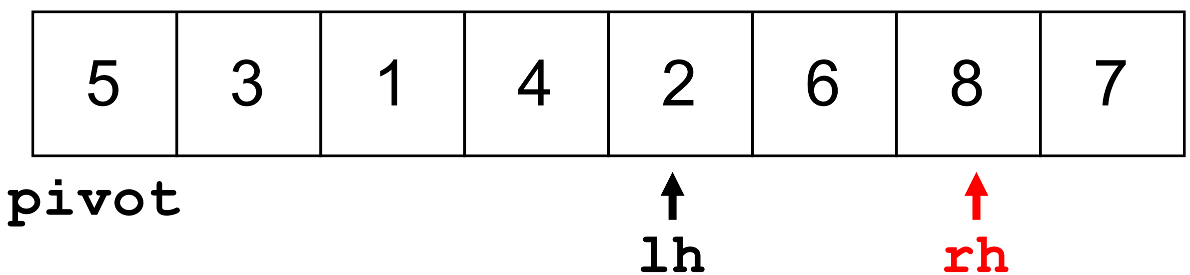
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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
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;
}
```



```
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		

```cpp
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;
}
```

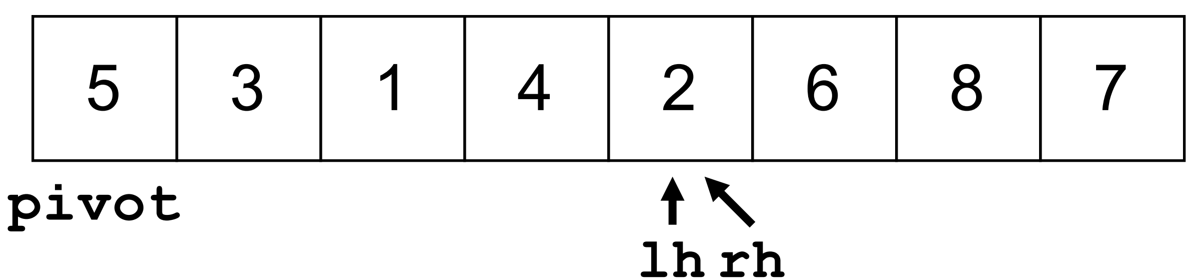
5	3	1	4	2	6	8	7

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;
}
```



```
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;
}
```

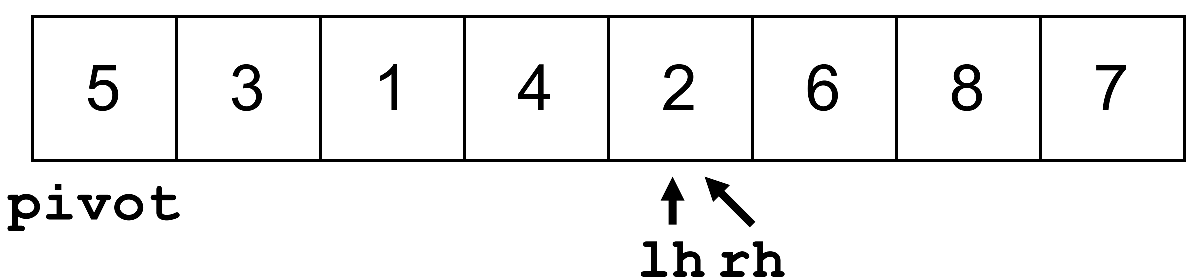
5	3	1	4	2	6	8	7

pivot

lh rh

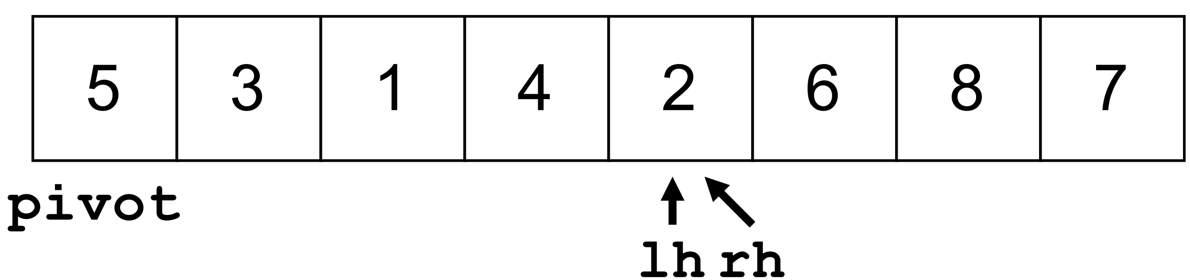
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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;
}
```



```
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;
}
```

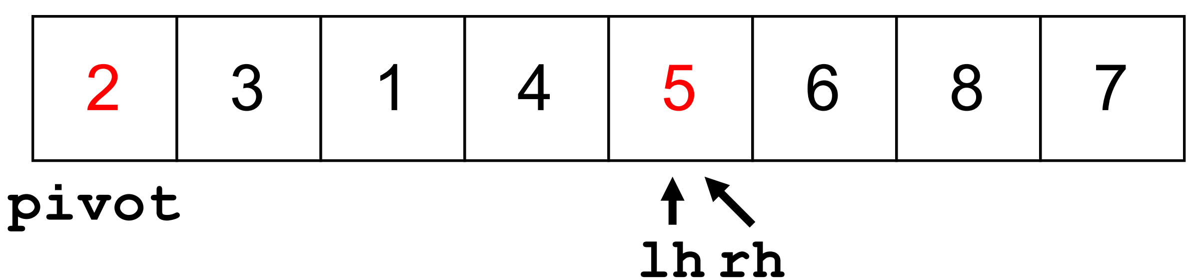
5	3	1	4	2	6	8	7

`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;
}
```



```
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

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;
}
```

Returns 4 (index of pivot)

2	3	1	4	5	6	8	7

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;
}
```

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

```
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

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

When are 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 $\qquad$ 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)

Y_a Y_b

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

Consider pivot choice: Y_a

They are compared

P(Y_a and Y_b ever compared)

$$\begin{array}{cccccc} & & Y_a & & Y_b & \\ 1 & 3 & \boxed{5} & 7 & \boxed{9} & 11 \end{array}$$

Consider pivot choice: Y_b

They are compared

P(Y_a and Y_b ever compared)

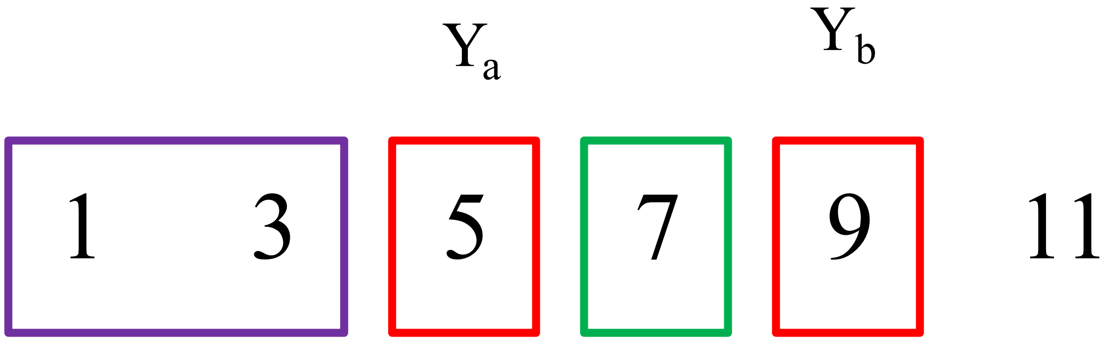
Y_a Y_b

1 3 5 7 9 11

Consider pivot choice: 7

They are **not** compared

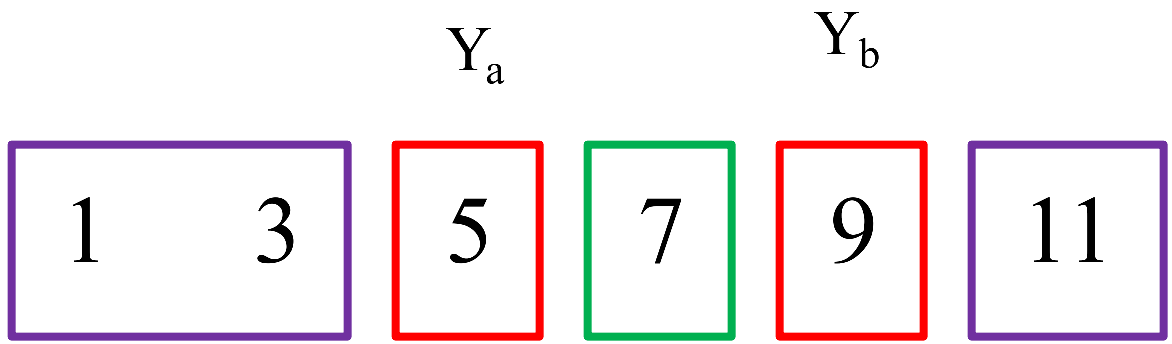
P(Y_a and Y_b ever compared)



Consider pivot choice: < Y_a

Whether or not they are compared
depends on future pivots

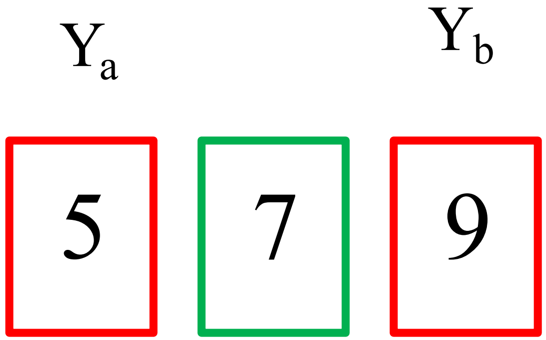
P(Y_a and Y_b ever compared)



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

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 $\mathrm{I}_{a,b}$ = 1 if Y_a and Y_b are compared, 0 otherwise
  - Order where $Y_b > Y_a$, so we have: $X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} I_{a,b}$

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

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

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

$$\approx 2\ln(n-a+1) \quad \text{for large } n$$

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

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 ☺

Break

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, … 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

amazon web services™

* 52% of Amazons Profits

**More 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 = 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

Break

Conditional Expectation

Conditional Expectation

- X and Y are jointly discrete random variables
  - Recall conditional PMF of X given Y = y:

$$p_{X|Y}(x \mid y) = P(X = x \mid Y = y) = \frac{p_{X,Y}(x, y)}{p_Y(y)}$$

- Define conditional expectation of X given Y = y:

$$E[X \mid Y = y] = \sum_x x P(X = x \mid Y = y) = \sum_x x\, p_{X|Y}(x \mid y)$$

- Analogously, jointly continuous random variables:

$$f_{X|Y}(x \mid y) = \frac{f_{X,Y}(x, y)}{f_Y(y)} \qquad E[X \mid Y = y] = \int_{-\infty}^{\infty} x\, f_{X|Y}(x \mid y)\, dx$$

Rolling Dice

- Roll two 6-sided dice D_1 and D_2
  - X = value of $D_1 + D_2$ Y = value of D_2
  - What is E[X | Y = 6]?

$$E[X \mid Y = 6] = \sum_x x P(X = x \mid Y = 6)$$

$$= \left(\frac{1}{6}\right)(7 + 8 + 9 + 10 + 11 + 12) = \frac{57}{6} = 9.5$$

  - Intuitively makes sense: 6 + E[value of D_1] = 6 + 3.5

Properties of Conditional Expectation

- X and Y are jointly distributed random variables

$$E[g(X) \mid Y = y] = \sum_{x} g(x) p_{X|Y}(x \mid y) \quad \text{or} \quad \int_{-\infty}^{\infty} g(x) f_{X|Y}(x \mid y)\, dx$$

- Expectation of conditional sum:

$$E\left[\sum_{i=1}^{n} X_i \mid Y = y\right] = \sum_{i=1}^{n} E[X_i \mid Y = y]$$

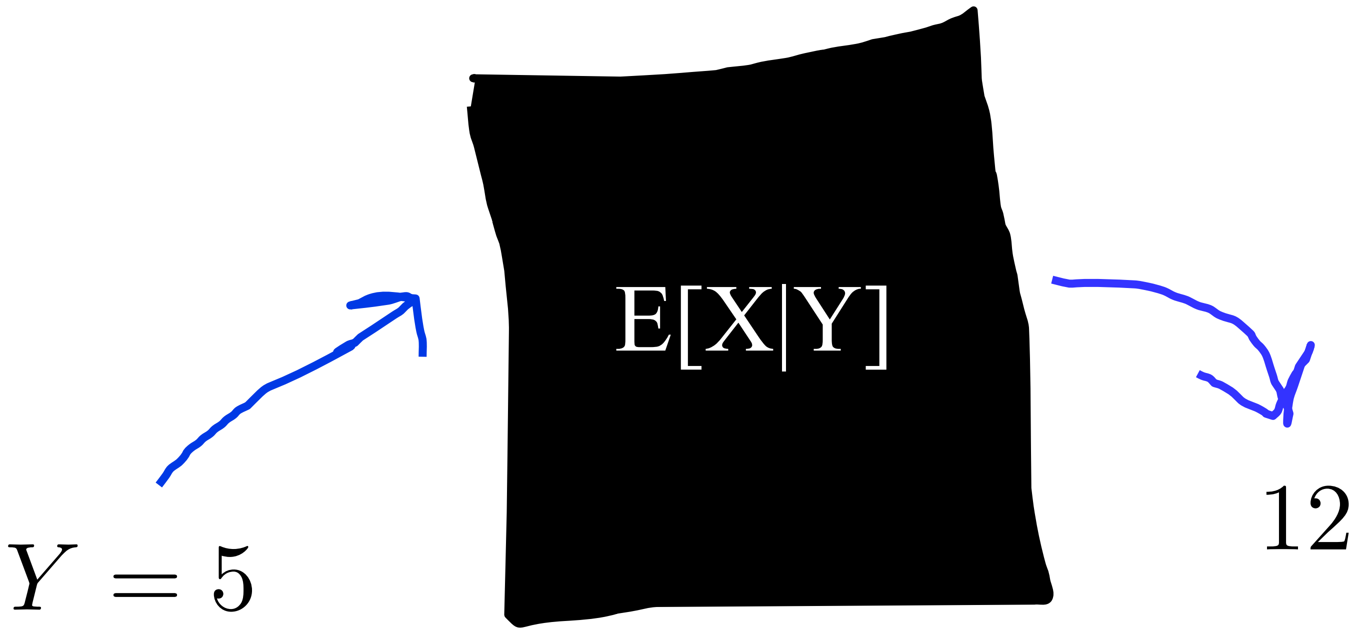
Expectations of Conditional Expectation

- Define $g(Y) = E[X \mid Y]$
  - For any $Y = y$, $g(Y) = E[X \mid Y = y]$
    - This is just function of Y, since we sum over all values of X

This is a function

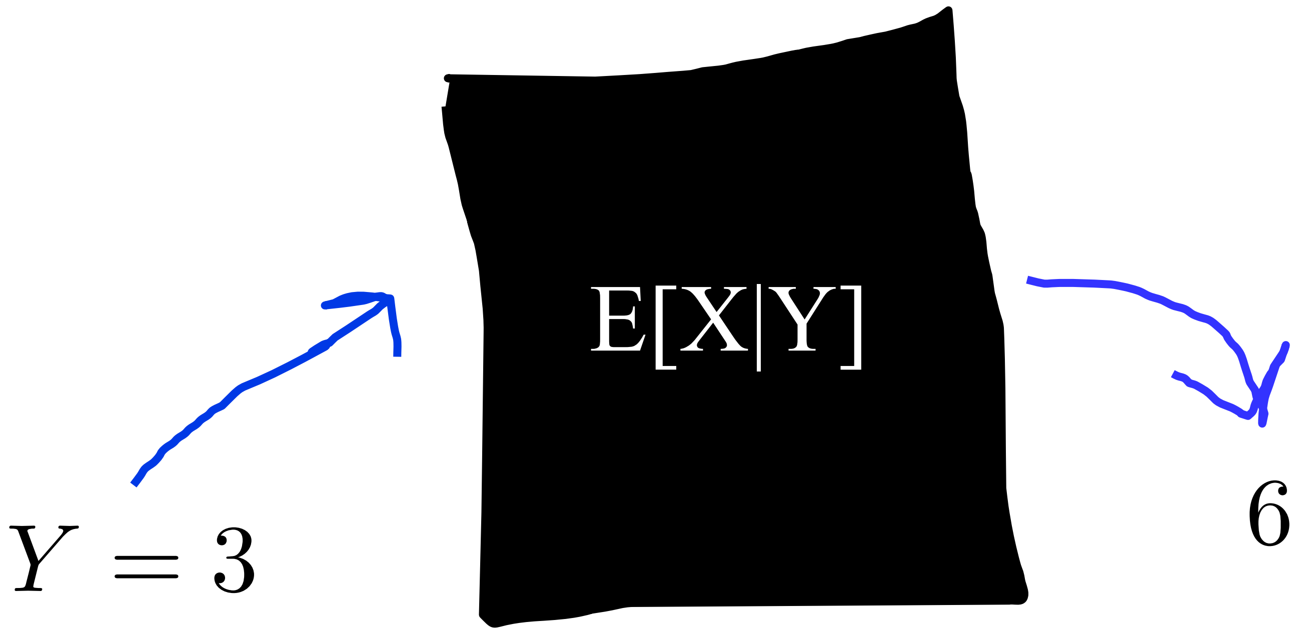
Expectations of Conditional Expectation

- Define $g(Y) = E[X \mid Y]$
  - For any $Y = y$, $g(Y) = E[X \mid Y = y]$
    - This is just function of Y, since we sum over all values of X



Expectations of Conditional Expectation

- Define $g(Y) = E[X \mid Y]$
  - For any $Y = y$, $g(Y) = E[X \mid Y = y]$
    - This is just function of Y, since we sum over all values of X



Analyzing Recursive Code

```
int Recurse() {
   int x = randomInt(1, 3);  // Equally likely values

   if (x == 1) return 3;
   else if (x == 2) return (5 + Recurse());
   else return (7 + Recurse());
}
```

- Let Y = value returned by `Recurse()`. What is E[Y]?

$$E[Y] = E[Y \mid X = 1]P(X = 1) + E[Y \mid X = 2]P(X = 2) + E[Y \mid X = 3]P(X = 3)$$

$$E[Y \mid X = 1] = 3$$

$$E[Y \mid X = 2] = E[5 + Y] = 5 + E[Y]$$

$$E[Y \mid X = 3] = E[7 + Y] = 7 + E[Y]$$

$$E[Y] = 3(1/3) + (5 + E[Y])(1/3) + (7 + E[Y])(1/3) = (1/3)(15 + 2E[Y])$$

$$E[Y] = 15$$

Protip: do this in CS161