Announcements

Problem Set #4 due today
Problem Set #5 released, due Monday 8/6
PSA: Only one late day (free or otherwise) allowed for PS #6, so plan ahead

Midterm results out after class
Summary of last time

Beta and Bayes’ Theorem:

def: conjugate distribution: prior and posterior have same parametric forms

1. Prior: Beta($a$, $b$) subjective belief of coin’s bias
   “imaginary trials”: saw (a-1) heads out of (a + b – 2) trials
   Where Beta(1,1) = Unif(0,1): “no imaginary trials”

2. Then observe $n + m$ trials, where $n$ are heads

3. Posterior: Beta($a + n$, $b + m$) your new belief of coin bias

Exercise:

Subjective belief: “this coin is $p = 1$ I think”, so 3 H, 0 T (imaginary)

Prior: Beta(1+3,1+0) = Beta(4,1)

Test result: 0 H, 3 T

Posterior: Beta(1+3+0,1+0+3) = Beta(4,3)
Where are we going with all of this?

As engineers, we want to:

1. Model things that can be random.
2. Find average values of situations that let us make good decisions.

We perform multiple experiments to help us with engineering.
Counts and averages are both sums of RVs.

Weeks 4 and 5:
Learning how to model multiple RVs

This week (week 6):
- Finding expectation of complex situations
- Defining sampling and using existing data
- Mathematical bounds w.r.t modeling the average

Week 7: bringing it all together
- Sampling + average = Central Limit Theorem
- Samples + modeling = finding the best model parameters given data
Goals for today

Advanced expectation
  ◦ Conditional expectation
    ◦ Analyzing recursive code
    ◦ Expectation of complex functions of random variables
  ◦ QuickSort, probability, and the
    \[ \sum_{x=k}^{n} \frac{1}{ax+b} \approx \int_{x=k}^{n} \frac{1}{ax+b} \, dx \] approximation
Previously proved properties of expectation

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

\[
E[aX + bY + c] = aE[X] + bE[Y] + c
\]

\[
Var(aX + b) = a^2 Var(X)
\]

For non-negative random variables X and Y:

\[
E[X] = \sum_{k=1}^{\infty} P(X \geq k) \quad E[Y] = \int_{0}^{\infty} P(Y \geq y) \, dy
\]
Conditional PMFs

Let $X$ and $Y$ be jointly discrete random variables.

The conditional PMF of $X$ given $Y$ (where $p_Y(y) > 0$) is defined as:

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

For $X$ and $Y$ as jointly continuous random variables:

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

Let $X$ and $Y$ be jointly discrete random variables.

The conditional expectation of $X$ given $Y = y$ is defined as:

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

For $X$ and $Y$ as jointly continuous random variables:

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

Note: $E[X|Y]$ is a function of $Y!$ It is a random variable!!
Rolling dice

Roll two 6-sided dice $D_1$ and $D_2$.

Let: \[ X = \text{sum of } D_1 + D_2 \]
\[ Y = D_2 \]

What is $E[X \mid Y = 6]$?

Solution 1:

\[
E[X \mid Y = 6] = \sum_x xP(X = x \mid Y = 6) = \left(\frac{1}{6}\right)(7 + 8 + 9 + 10 + 11 + 12) = \frac{57}{6} = 9.5
\]

Solution 2:

\[
E[X \mid Y = 6] = E[\text{<value of } D_1 > + 6 \mid Y = 6] = 6 + E[\text{<value of } D_1 >] = 6 + 3.5 = 9.5
\]
Properties of conditional expectation

Let $X$ and $Y$ be jointly distributed random variables.

Expectation of function of $X$:

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

or

$$\int_{-\infty}^{\infty} g(x)f_{X|Y}(x|y)dx$$

Expectation of conditional sum:

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

Expectation of conditional expectation:

$$E_Y[E_X[X|Y]] = E_X[X]$$

(whoa...)
Expectation of conditional expectation

Prove that $E_Y[E_X[X|Y]] = E_X[X]$.

We can define $g(Y) = E_X[X|Y]$

- a random function of $Y$ only, since sum over all values of $X$
- For any $Y = y$, $g(Y = y) = E_X[X|Y = y]$

For discrete $X$ and $Y$:

Proof:

$E_Y[E_X[X|Y]] = E_Y[g(Y)] = \sum_y P(Y = y)g(Y = y)$

$= \sum_y P(Y = y)E_X[X|Y = y] = \sum_y P(Y = y)\left(\sum_x x P(X = x|Y = y)\right)$

$= \sum_y \sum_x x P(X = x|Y = y)P(Y = y) = \sum_y \sum_x x P(X = x, Y = y)$  \quad \text{(def of cond. prob.)}$

$= \sum_x \sum_y P(X = x, Y = y) \quad \sum_x x P(X = x)$  \quad \text{(def of marginal prob.)}$

$= E[X]$  \quad \text{(same for continuous)}$
Simple expectations of functions of multiple RVs so far:

- Linear functions: \( E[X + Y] = E[X] + E[Y] \)
- Products of independent \( X \) and \( Y \): \( E[XY] = E[X]E[Y] \)

Conditional expectation:

\[
E_Y[E_X[X|Y]] = E_X[X]
\]

- Expectations of complex functions, like \( E[\sum_{i=1}^X Y] \)
- Analyze recursive code!
Analyzing recursive code

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

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

Let Y = return value of recurse().

What is $E[Y]$?

Solution:

$$E[Y] = E_X[E_Y[Y|X]]$$


$$E_Y[Y|X = 1] = 3$$


$$E_Y[Y|X = 3] = E[7 + Y] = 7 + E[Y]$$

$$E[Y] = \left(\frac{1}{3}\right)3 + \left(\frac{1}{3}\right)(5 + E[Y]) + \left(\frac{1}{3}\right)(7 + E[Y])$$

$$= \left(\frac{1}{3}\right)(15 + 2E[Y])$$

$$E[Y] = 5 + \frac{2}{3}E[Y]$$

$$E[Y] = 15$$
Random number of random variables

Suppose you have a website.
• $X = \# \text{ people/day who visit. } X \sim N(50,25)$
• $Y_k = \# \text{ minutes spent by visitor } k. \ Y_k \sim Poi(8)$
• $X$ and all $Y_k$ are independent
• $W = \text{ total time spent by all visitors/day, } W = \sum_{k=1}^{X} Y_k$

What is $E[W]$?

Solution:

\[
E[W] = E\left(\sum_{k=1}^{X} Y_k\right) = EX \left(\sum_{k=1}^{X} E[Y_k | X]\right)
\]

\[
E\left[\sum_{k=1}^{X} Y_k | X = n\right] = \sum_{k=1}^{n} E[Y_k | X = n] = \sum_{k=1}^{n} E[Y_k] = nE[Y_k]
\]

\[
E\left[\sum_{k=1}^{X} Y_k | X\right] = XE[Y_k]
\]

\[
E[W] = EX \left[E\left[\sum_{k=1}^{X} Y_k | X\right]\right] = EX [XE[Y_k]] = E[X]E[Y_k]
\]

\[= 50 \times (8) = 400.\]
Break

Attendance: tinyurl.com/cs109summer2018
Hiring software engineers

Your company has only one job opening for a software engineer.

- $n$ candidates interview, in order ($n!$ orderings equally likely)
- Note: There is an $\alpha$-to-1 factor difference in productivity b/t the “best” and “average” software engineer.
- Steve jobs said $\alpha=25$, Mark Zuckerberg claims $\alpha=100$
- Must decide hire/no hire immediately after each interview

Strategy:
1. Interview $k$ (of $n$) candidates and reject all $k$
2. Accept the next candidate better than all of first $k$ candidates.

What is your target $k$ so that you maximize your probability of getting the best candidate?

Solution:

Define: $X = \text{position of best engineer candidate (1, 2, ..., n)}$

$B = \text{event that you hire the best engineer}$

Want to maximize for $k$: $P_k(B) = \text{probability of B when using strategy for a given k}$

$$P_k(B) = \sum_{i=1}^{n} P_k(B|X = i)P(X = i) = \frac{1}{n} \sum_{i=1}^{n} P_k(B|X = i)$$ (law of total probability)
Hiring software engineers

Your company has only one job opening for a software engineer.

Strategy:
1. Interview \( k \) (of \( n \)) candidates and reject all \( k \) candidates.
2. Accept the next candidate better than all of first \( k \) candidates.

What is your target \( k \) so that you max your probability of getting the best candidate?

Solution:
Define: \( X = \) position of best engineer candidate
\( B = \) event that you hire the best engineer

If \( i \leq k : \quad P_k(B|X = i) = 0 \) (we fired best candidate already)

Else: We must not hire prior to the \( i \)-th candidate.
\( \rightarrow \) We must have fired the best of the \( i-1 \) first candidates.
\( \rightarrow \) The best of the \( i-1 \) needs to be our comparison point for positions \( k+1, \ldots, i-1 \).
\( \rightarrow \) The best of the \( i-1 \) needs to be one of our first \( k \) comparison/auto-fire

\[
P_k(B) = \frac{1}{n} \sum_{i=1}^{n} P_k(B|X = i) = \frac{1}{n} \sum_{i=k+1}^{n} \frac{k}{i-1}
\]
\( \leftarrow \) Want to maximize over \( k \)
Hiring software engineers

Your company has only one job opening for a software engineer.

Strategy:
1. Interview $k$ (of $n$) candidates and reject all $k$
2. Accept the next candidate better than all of first $k$ candidates.

What is your target $k$ so that you max your probability of getting the best candidate?

Solution:
Want to maximize over $k$:

$$P_k(B) = \frac{1}{n} \sum_{i=k+1}^{n} \frac{k}{i-1} \approx \frac{k}{n} \int_{i=k+1}^{n} \frac{1}{i-1} \, di = \frac{k}{n} \ln(i-1) \bigg|_{i=k+1}^{n} = \frac{k}{n} \ln \frac{n-1}{k} \approx \frac{k}{n} \ln \frac{n}{k}$$

Maximize by differentiating w.r.t $k$, set to 0, solve for $k$:

$$\frac{d}{dk} \left( \frac{k}{n} \ln \frac{n}{k} \right) = \frac{1}{n} \ln \frac{n}{k} + \frac{k}{n} \cdot \frac{n}{n} \cdot \frac{-n}{k^2} = 0$$

$$\ln \frac{n}{k} = 1$$

$$k = \frac{n}{e}$$

1. Interview $\frac{n}{e}$ candidates
2. Pick best based on strategy
3. $P_k(B) \approx 1/e \approx 0.368$
Quicksort

We just did the warmup to Quicksort.

You have been told Quicksort is $O(n \log n)$ which is the “average case.”

Now we get to prove it! 😊😊😊😊😊
1. Select “pivot”
Quicksort refresher

1. Select “pivot”
2. Partition the array

- Everything < pivot on left
- Everything ≥ pivot on right
- Pivot in-between
Quick sort refresher

1. Select “pivot”
2. Partition the array
3. Recursively sort left partition
Quicksort refresher

1. Select “pivot”
2. Partition the array
3. Recursively sort left partition
4. Recursively sort right partition
Quicksort refresher

1. Select “pivot”
2. Partition the array
3. Recursively sort left partition
4. Recursively sort right partition

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:
   = index of pivot
   = # 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;
}
**Quicksort’s wingman**  (who does all the work)

```c
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;
}
```
Quicksort’s wingman  (who does all the work)

```c
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;
}
Quicksort’s wingman  (who does all the work)

```c
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;
}
```

Quicksort’s wingman (who does all the work)

```c
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;
}
```
Quicksort’s wingman  (who does all the work)

```c
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;
}
```

![Diagram](image.png)
Quicksort’s wingman  (who does all the work)

```c
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;
}
```
Quicksort’s wingman  (who does all the work)

```c
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;
}
```

![Quicksort visualization](image.png)
Quicksort’s wingman  
(who does all the work)

```c
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

lh  rh
Quicksort’s wingman  (who does all the work)

```c
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;
}
```
Quicksort’s wingman  
(who does all the work)

```c
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;
}
```

```c
5 3 1 4 2 6 8 7
```

```
    lh  rh
      ↑   ↑
```

(who does all the work)
Quicksort’s wingman  (who does all the work)

```c
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

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;
}
Quicksort’s wingman (who does all the work)

```c
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;
}
```
Quicksort’s wingman (who does all the work)

```c
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;
}
```
Quicksort’s wingman  (who does all the work)

```c
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;
}
```

```plaintext
2 3 1 4 5 6 8 7
lh rh
```
Quicksort’s wingman  (who does all the work)

```c
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:
# of comparisons made to pivot

Returns 5 (index of pivot)
Quicksort complexity

- On average, Quicksort is $O(n \log n)$, where $n = \# \text{ elements}$.
- Worst case: $O(n^2)$, when the pivot is maximal or minimal on every recursive call.

We can ask two probabilistic questions about runtime:

1. What is $P(\text{Quicksort worst case runtime})$?
2. What is $E[\text{Quicksort runtime}]$?
1. What is $P(\text{Quicksort worst case runtime})$?

Solution:

- 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) per call

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

Similar for BSTs (pset #1):
As $n \rightarrow \infty$, $P(\text{worst case}) \rightarrow 0$
Average Quicksort

2. What is $E[\text{Quicksort runtime}]$?

Solution:

Define: $X = \#$ comparisons to pivot

WTF: $E[X]$ (dependent comparisons...use indicator variables!)

Define: $Y_1, Y_2, ..., Y_n$ elements in sorted order

$I_{a,b} = 1$ if $Y_a, Y_b$ ever compared (where $Y_a < Y_b$)

Then,

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

(unique pairs)
Average Quicksort

2. What is $E[\text{Quicksort runtime}]$?

Solution:

Define: $X = \#$ comparisons to pivot

$Y_1, Y_2, \ldots, Y_n$ elements in sorted order

$I_{a,b} = 1$ if $Y_a, Y_b$ ever compared (where $Y_a < Y_b$)

Then,

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

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

- If pivot $< Y_a$ or $> Y_b$, not directly compared (but could be in future recursive call)
- Care only about calls where pivot in $\{Y_a, Y_{a+1}, Y_{a+2}, \ldots, Y_b\}$
- $\Rightarrow Y_a, Y_b$ must be selected (with equal prob.) in order to be directly compared
Average Quicksort

2. What is $E[\text{Quicksort runtime}]$?

Solution:

Define: $X = \# \text{ comparisons to pivot}$

$Y_1, Y_2, \ldots, Y_n$ elements in sorted order

$I_{a, b} = 1$ if $Y_a, Y_b$ ever compared (where $Y_a < Y_b$)

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

(from here on it’s rollercoaster math)

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

$$= [2\ln(b-a+1)]_{b=a+1}^{n} = 2\ln(n-a+1) - 2\ln2 \approx 2\ln(n-a+1)$$

(when $n$ is large)
Average Quicksort

2. What is $E[\text{Quicksort runtime}]$?

Solution:

Define: $X =$ # comparisons to pivot

$Y_1, Y_2, ..., Y_n$ elements in sorted order

$I_{a,b} = 1$ if $Y_a, Y_b$ ever compared (where $Y_a < Y_b$)

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

 stil more rollercoaster math

$$\approx \int_{a=1}^{n-1} 2\ln(n - a + 1) \, da = 2 \int_{a=1}^{n-1} \ln(n - a + 1) \, da = -2 \int_{y=n}^{2} \ln(y) \, dy \quad \text{(u-substitution: Let } y = n - a+1)$$

$$= -2 \left[ y \ln(y) - y \right]_{n}^{2} \quad \text{(Integration by parts: } \int \ln(x) \, dx = x \ln(x) - x)$$

$$= -2\left(2\ln(2) - 2 - (n\ln(n) - n)\right) \approx 2n\ln(n) - 2n \quad = O(n \log n)$$
Summary of this time

Conditional expectation: \[ E_Y[E_X[X|Y]] = E_X[X] \]

- Expectations of complex functions, like \[ E[\sum_{i=1}^X Y] \]
- Analyze recursive code!

Quicksort:

- While recursive, can be solved as an expectation of a sum of indicator random variables.
- When dealing with a sum of non-trivial indicator probabilities, \[ \sum_{x=k}^{n} \frac{1}{ax+b} \approx \int_{x=k}^{n} \frac{1}{ax+b} \, dx \]

(QuickSort is beyond the scope of your HW, but you should understand it)
Midterm results

Mean: 90.67 (75.6%)
Median: 94.0 (78.3%)
Stdev: 22.0
Max: 119.0 (x 1)