# Course Mean

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

*This is actual midpoint of course*

*(Just wanted you to know)*

# Sample Mean

- Consider *n* random variables $X_1, X_2, \ldots X_n$
  - $X_i$ are all independently and identically distributed (I.I.D.)
  - Have same distribution function *F* and $E[X_i] = \mu$
  - We call sequence of $X_i$ a **sample** from distribution *F*
  - Sample mean: $\overline{X} = \sum_{i=1}^{n} \frac{X_i}{n}$
  - Compute $E[\overline{X}]$

$$E[\overline{X}] = E\left[\sum_{i=1}^{n} \frac{X_i}{n}\right] = \frac{1}{n} E\left[\sum_{i=1}^{n} X_i\right]$$

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

  - $\overline{X}$ is "unbiased" estimate of $\mu$ $(E[\overline{X}] = \mu)$

# Boole was so 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 $E[X_i] = P(E_i)$
  - Now, let $X = \sum_{i=1}^{n} X_i$ and let Y = 1 if X ≥ 1, 0 otherwise
  - Note: $X \geq Y \Rightarrow E[X] \geq E[Y]$

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

$$E[Y] = P(\text{at least one event } E_i \text{ occurs}) = P\left(\bigcup_{i=1}^{n} E_i\right)$$

  - Boole's inequality: $\sum_{i=1}^{n} P(E_i) \geq P\left(\bigcup_{i=1}^{n} E_i\right)$
    - Boole died from being too cool (literally)!

# Expectation of (Negative) Binomial

- Let $Y \sim \text{Bin}(n, p)$
  - $n$ independent trials
  - Let $X_i = 1$ if $i$-th trial is "success", 0 otherwise
  - $X_i \sim \text{Ber}(p)$
  - $E[X_i] = p \quad (= 1p + 0(1 - p))$
  - $E[Y] = E[X_1] + E[X_2] + ... + E[X_n] = np$
- Let $Y \sim \text{NegBin}(r, p)$
  - Recall Y is number of trials until r "successes"
  - Let $X_i$ = # of trials to get success after ($i$ – 1)st success
  - $X_i \sim \text{Geo}(p)$ (i.e., Geometric RV) $\quad E[X_i] = 1/p$
  - $E[Y] = E[X_1] + E[X_2] + ... + E[X_r] = r/p$

# Hash Tables (a.k.a. Coupon 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: $X_i \sim \text{Geo}((n - i) / n)$
    - $E[X_i] = 1 / p = n / (n - i)$
  - $X = X_0 + X_1 + \ldots + X_{n-1} \Rightarrow E[X] = E[X_0] + E[X_1] + \ldots + E[X_{n-1}]$

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

# 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

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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;
}
```

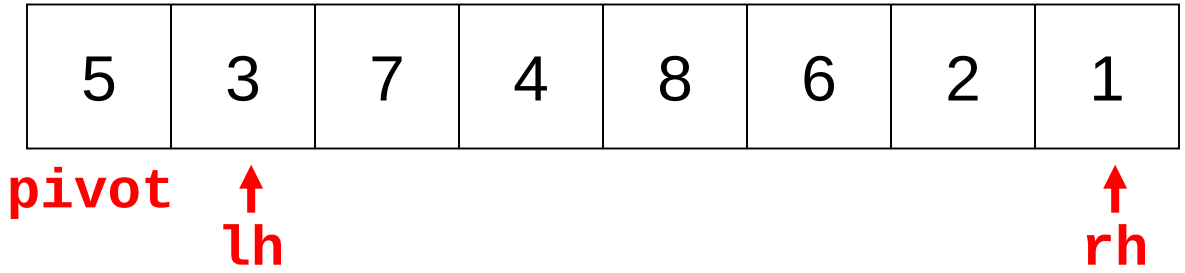
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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;
}
```



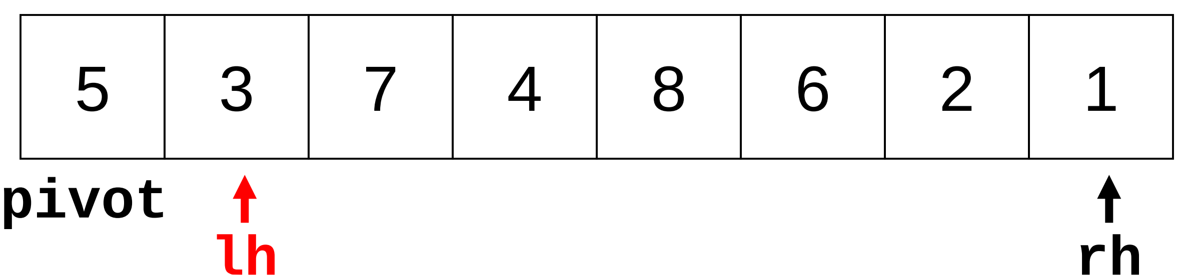
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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;
}
```



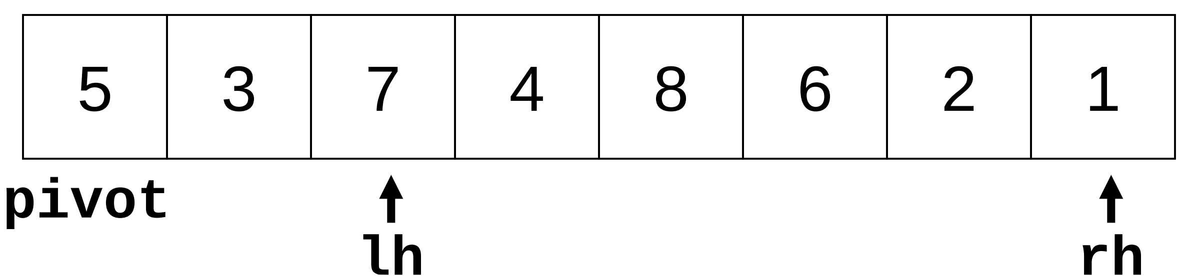
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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 |

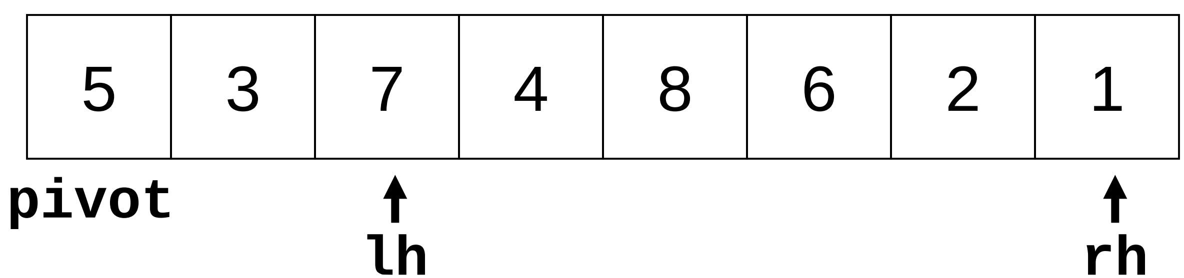
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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;
}
```



```
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 |

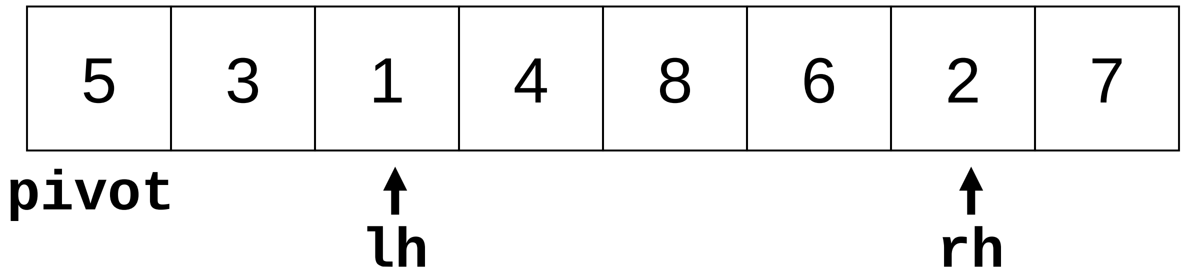
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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 | |

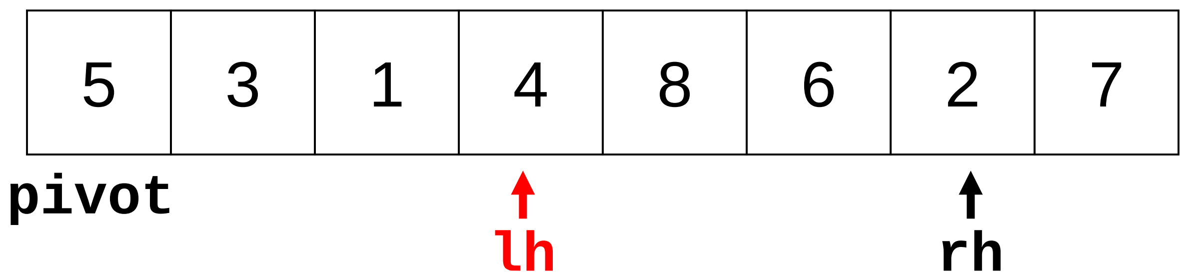
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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;
}
```



```
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;
}
```



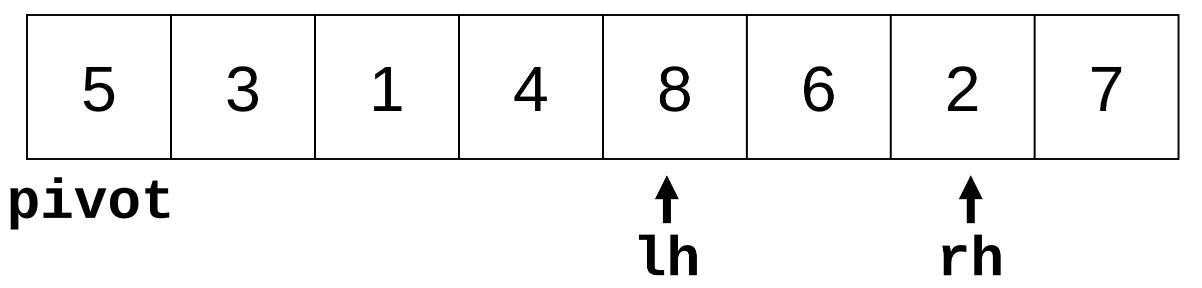
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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;
}
```



```
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 | |

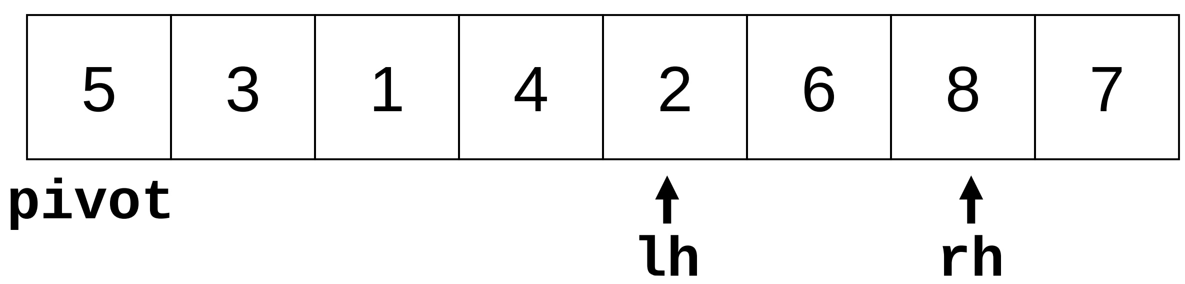
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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 | |

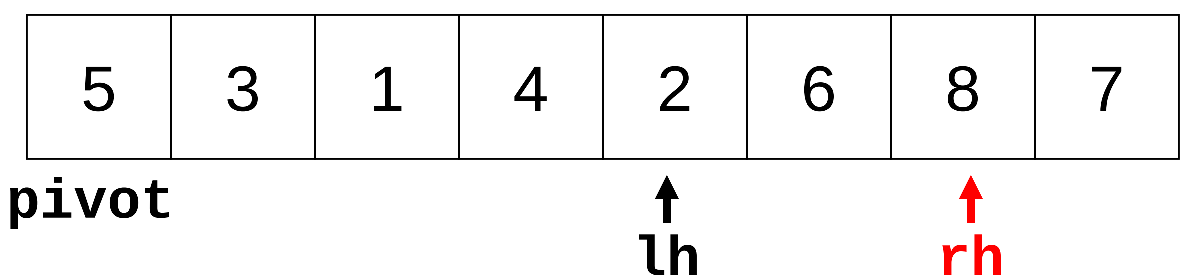
```
int Partition(int arr[], int n)
{
   int lh = 1, rh = n - 1;

   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;
}
```



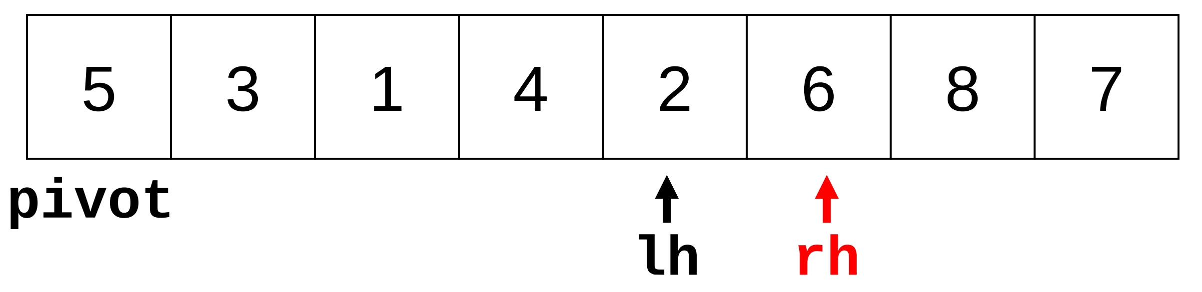
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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;
}
```



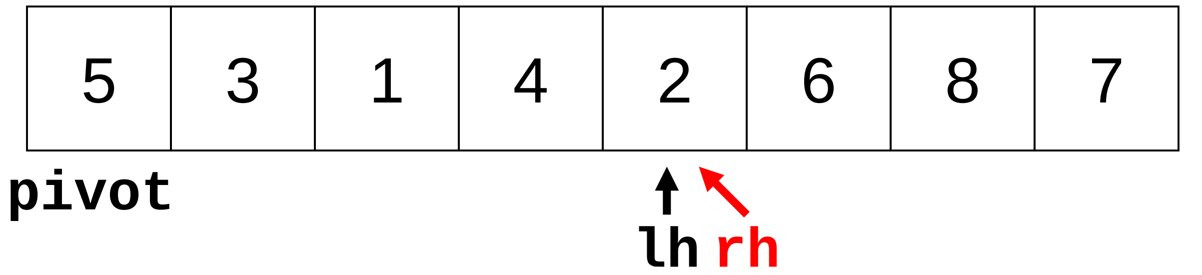
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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;
}
```



```
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 | | | |

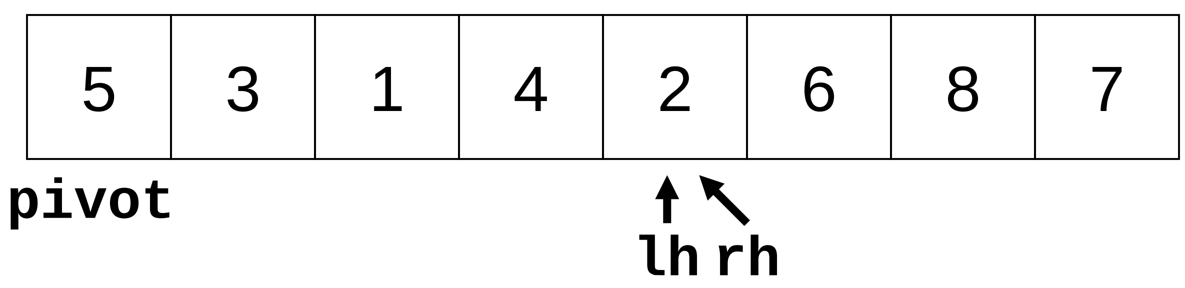
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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;
}
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 3 | 1 | 4 | 2 | 6 | 8 | 7 |
| pivot | | | | lh rh | | | |

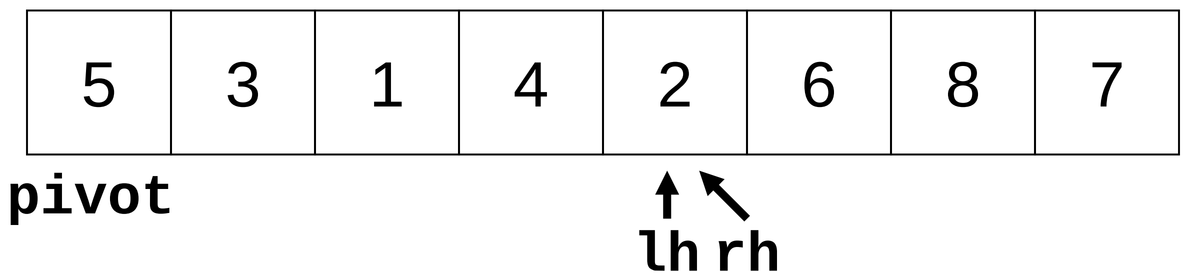
```
int Partition(int arr[], int n)
{
    int lh = 1, rh = n - 1;

    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;
}
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 3 | 1 | 4 | 5 | 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 |
| 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;
}
```

Returns 4 (index of pivot)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 3 | 1 | 4 | 5 | 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;
}
```

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

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

# Determining P($Y_a$ and $Y_b$ ever compared)

- Consider when $Y_a$ and $Y_b$ are directly compared
  - If pivot chosen is < $Y_a$ or > $Y_b$, then $Y_a$ and $Y_b$ are not directly compared (since values only compared to pivot)
    - But $Y_a$ and $Y_b$ might still be compared in a future recursive call
  - So, 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, $P(Y_a \text{ and } Y_b \text{ ever compared}) = \dfrac{2}{b-a+1}$

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

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

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

$$\approx 2\ln(n-a+1)$$ 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 $y = n-a+1$

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

Integration by parts:

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

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

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