CSCI 132:
Basic Data Structures and Algorithms

Recursion (Part 1)

Reese Pearsall

Spring 2025

Announcements

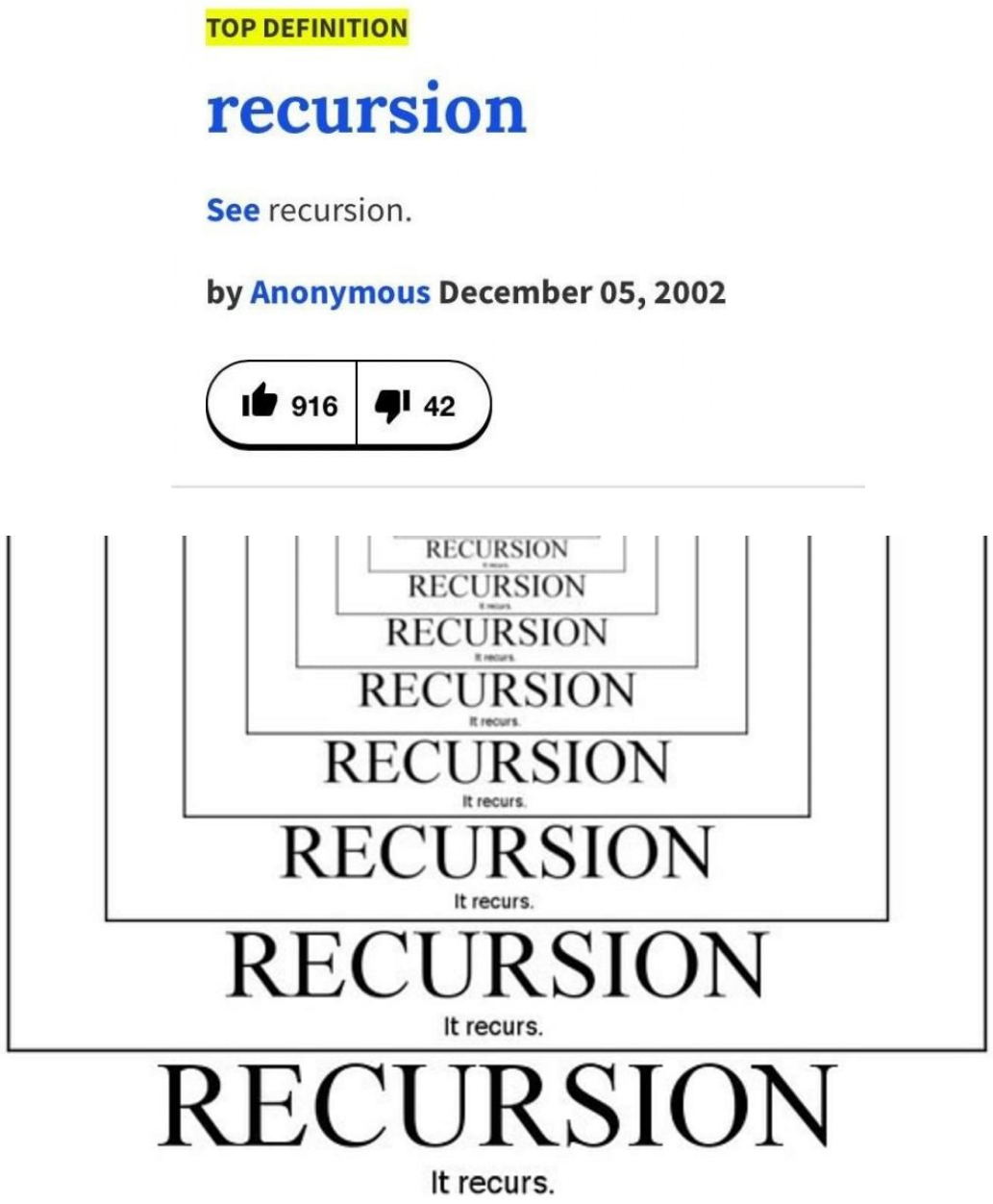
Program 3 due Friday

Friday will be a help session for program 3

Recursion is a problem-solving technique that involves a *method calling itself* to solve some smaller problem

```
static int factorial(int n)
    {
        if (n == 0)
            return 1;

        return n * factorial(n - 1);
    }
```



countX(“oxxo”)

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“xxo”)

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“xxo”)

1 + countX(“xo”)

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“**x**xo”)

1 + countX(“**x**o”)

1 + countX(“o”)

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x'){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“oxxo”)

0 + countX(“xxo”)

1 + countX(“xo”)

1 + countX(“o”)

0 + countX(“”)

```java
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“xxo”)

1 + countX(“xo”)

1 + countX(“o”)

0 + countX(“”)

0

```java
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“xxo”)

1 + countX(“xo”)

1 + countX(“o”)

0 + 0

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“xxo”)

1 + countX(“xo”)

1 + 0

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“xxo”)

1 + countX(“xo”)

1 + 0

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“**o**xxo”)

0 + countX(“**x**xo”)

1 + 1

```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

countX(“oxxo”)

0 + 2

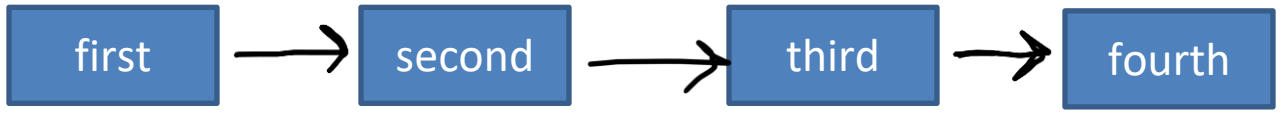
```
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

Final answer = 2

```java
public static int countX(String str) {
    if(str.length() == 0){
        return 0;
    }
    if(str.charAt(0) == 'x’){
        return 1 + countX(str.substring(1));
    }
    else{
        return 0 + countX(str.substring(1));
    }
}
```

Limitations of recursion?

Example #3: Printing a Linked List in **Reverse**



Goal: Print contents of linked list in reverse order using recursion

Expected Output

```
fourth
third
second
first
```

Base Case?

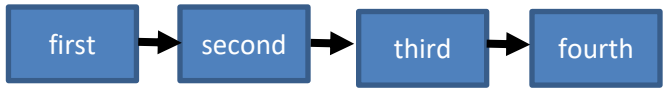
If the size of the LL is 1, print out the only node

Recursive Case?

Remove a node (but don't print it yet), call the recursive method and pass it the new LL. When method returns, print out the node we saved

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

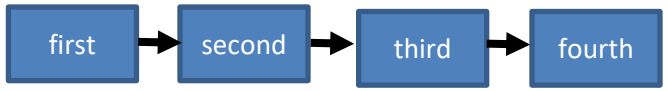


value saved: “first”

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”



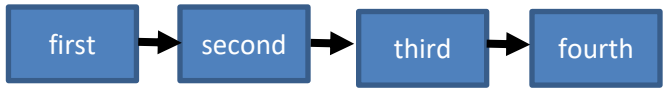

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”

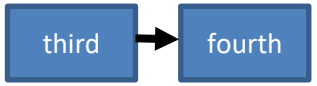



```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”


```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

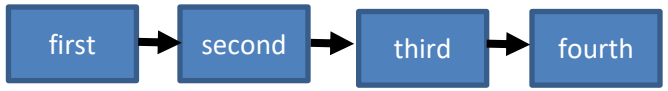
value saved: “third”



```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”

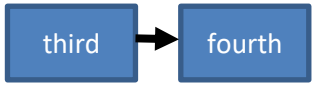



```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”


```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

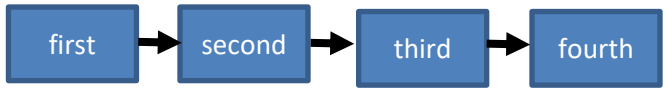
value saved: “third”



```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”

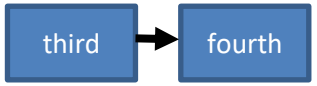



```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”


```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “third”




```java
System.out.println(ll.getFirst());
return;
```

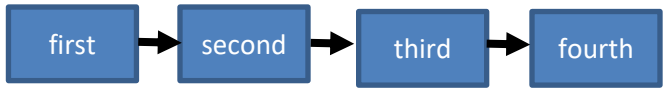
```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

Output

fourth

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”

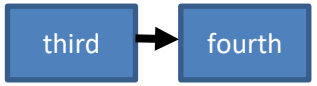



```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”


```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “third”



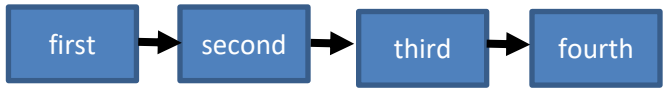

```java
System.out.println(ll.getFirst());
return;
```

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

Output

fourth

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

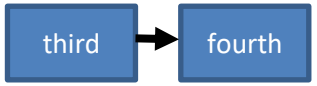


value saved: “first”

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```



value saved: “third”

```java
System.out.println(ll.getFirst());
return;
```

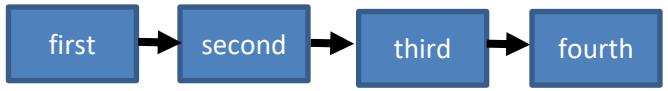
```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

Output

fourth

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”

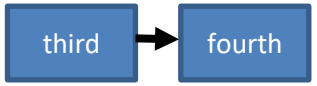



```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”


```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “third”



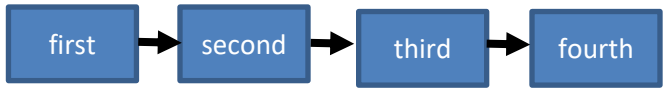
```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

Output

```
fourth
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”

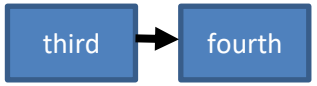



```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”


```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “third”



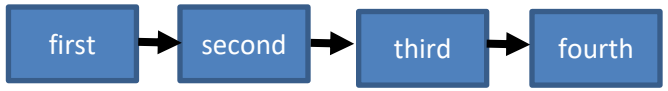
```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

Output

```
fourth
third
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”

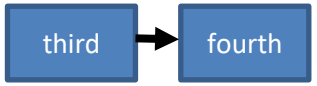



```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”


```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

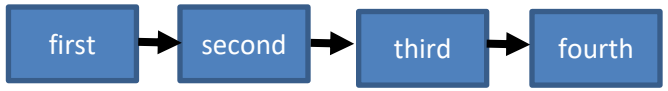
value saved: “third”



```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”




```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “second”

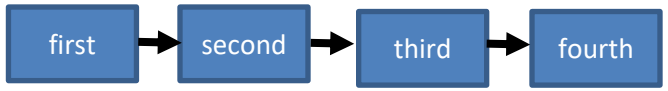
Output

```
fourth
third
```

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”




```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

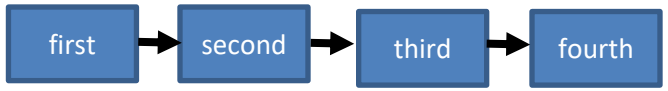
value saved: “second”

Output

```
fourth
third
second
```

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```



value saved: “first”

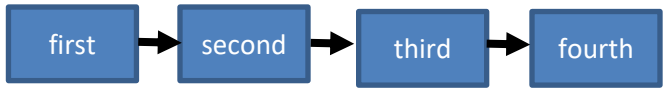
Output

```
fourth
third
second
```

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

```java
String removed = ll.removeFirst();
print_LL_reverse(ll);
System.out.println(removed);
return;
```

value saved: “first”



Output

```
fourth
third
second
first
```

```java
public static void print_LL_reverse(LinkedList<String> ll) {
  if(ll.size() == 1) {
    System.out.println(ll.getFirst());
    return;
  }
  else {
    String removed = ll.removeFirst();
    print_LL_reverse(ll);
    System.out.println(removed);
    return;
}
```

Output

```
fourth
third
second
first
```

Linear Search

“Reese”	“Steve”	“Nando”	“Sally”	“Bob”	“Jill”

```
public static int linear_search(String[] names, String target, int current){



}
```

Linear Search

“Reese”	“Steve”	“Nando”	“Sally”	“Bob”	“Jill”

```
public static int linear_search(String[] names, String target, int current){

        //base case


        //recursive case


}
```

Linear Search

“Reese”	“Steve”	“Nando”	“Sally”	“Bob”	“Jill”

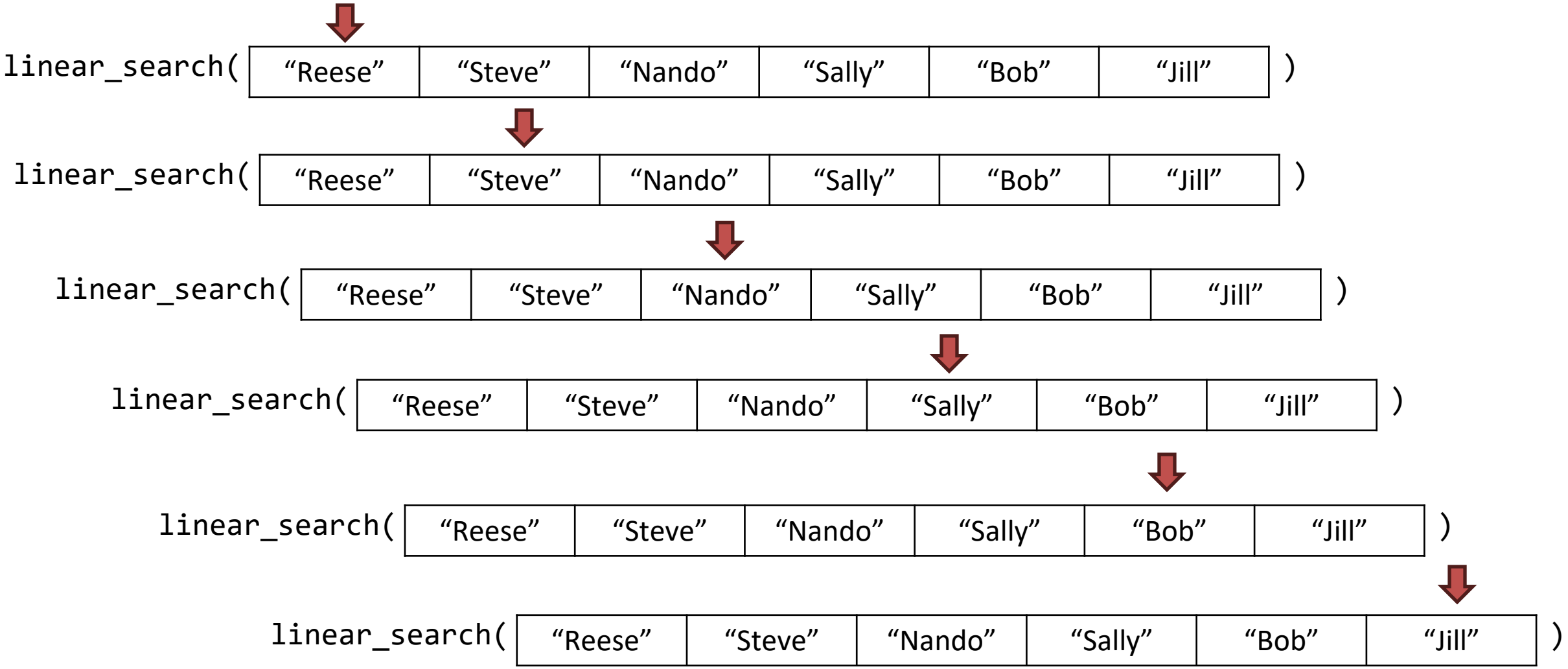
```
public static int linear_search(String[] names, String target, int current){

        //base case
        if(current >= names.length){
                return -1;
        }
        if(target.equals(names[current])){
                return current;
        }

        //recursive case
        return linear_search(names, target, current+1);

}
```

Linear Search



Permutations

Calculation of the number of ways a particular *set* can be arranged

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

Permutations

Calculation of the number of ways a particular *set* can be arranged

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

Set: a collection of distinct unordered items

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

Permutations

Calculation of the number of ways a particular *set* can be arranged

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

Set: a collection of *distinct* unordered items

Ex:

{1, 2, 3, 4}

{‘H’, ‘I’}

{A, B, C, D}

{“I”, “L”, “A”, “N”}

{1.4, 3.6, 9.5}

{“R”, “E”, “S”}

Permutations

Calculation of the number of ways a particular *set* can be arranged

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

<u>*Set*</u>: a collection of *distinct* unordered items

Ex:

{1, 2, 3, 4}

{‘H’, ‘I’}

{A, B, C, D}

{“I”, “L”, “A”, “N”}

{1.4, 3.6, 9.5}

{“R”, “E”, “S”}

CSCI 246 Discrete Structures: 3 Credits (3 Lec)

PREREQUISITE: M 171Q or M 165Q
COREQUISITE: CSCI 132. (F, Sp, Su) This course covers logic, discrete probability, recurrence relations, Boolean algebra, sets, relations, counting, functions, maps, Big-O notation, proof techniques including induction, and proof by contradiction
.

Permutations

Calculation of the number of ways a particular set can be arranged

Ex: “ABC”

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

Permutations

Calculation of the number of ways a particular set can be arranged

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

Ex: “ABC”

A B C

A C B

B A C

B C A

C A B

C B A

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

Permutations

Calculation of the number of ways a particular set can be arranged

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

Ex: “ABC”

1 A B C

2 A C B

3 B A C

4 B C A

5 C A B

6 C B A

Permutations

Calculation of the number of ways a particular set can be arranged

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

${}_nP_r$ = permutation

n = total number of objects

r = number of objects selected

Ex: “ABC”

1 A B C

2 A C B

3 B A C

4 B C A

5 C A B

6 C B A

$\boxed{3! = 3 * 2 * 1 = \mathbf{6}}$

Permutations

What if we add one more letter?

Calculation of the number of ways a particular set can be arranged

Ex: “ABC” → “ABCD”

Permutations

What if we add one more letter?

Calculation of the number of ways a particular set can be arranged

Ex: “ABC” → “ABCD”

ABCD	BACD	CABD	DABC
ABDC	BADC	CADB	DACB
ACBD	BCAD	CBAD	DBAC
ACDB	BCDA	CBDA	DBCA
ADBC	BDAC	CDAB	DCAB
ADCB	BDCA	CDBA	DCBA

Permutations

What if we add one more letter?

Calculation of the number of ways a particular set can be arranged

Ex: “ABC” → “ABCD”

1	ABCD	7	BACD	13	CABD	19	DABC
2	ABDC	8	BADC	14	CADB	20	DACB
3	ACBD	9	BCAD	15	CBAD	21	DBAC
4	ACDB	10	BCDA	16	CBDA	22	DBCA
5	ADBC	11	BDAC	17	CDAB	23	DCAB
6	ADCB	12	BDCA	18	CDBA	24	DCBA

Permutations

What if we add one more letter?

Calculation of the number of ways a particular set can be arranged

Ex: “ABC” → “ABCD”

1	ABCD	7	BACD	13	CABD	19	DABC
2	ABDC	8	BADC	14	CADB	20	DACB
3	ACBD	9	BCAD	15	CBAD	21	DBAC
4	ACDB	10	BCDA	16	CBDA	22	DBCA
5	ADBC	11	BDAC	17	CDAB	23	DCAB
6	ADCB	12	BDCA	18	CDBA	24	DCBA

$4! = 4 * 3 * 2 * 1 = \mathbf{24}$

Powerball (Real)

Rules:

select five numbers between 1 and 69 for the white balls

select one number between 1 and 26 for the red Powerball

Match	Prize	Odds
○○○○○ + ●	Grand Prize	1 in 292,201,338.00
○○○○○	$1,000,000	1 in 11,688,053.52
○○○○ + ●	$50,000	1 in 913,129.18
○○○○	$100	1 in 36,525.17
○○○ + ●	$100	1 in 14,494.11
○○○	$7	1 in 579.76
○○ + ●	$7	1 in 701.33
○ + ●	$4	1 in 91.98
●	$4	1 in 38.32

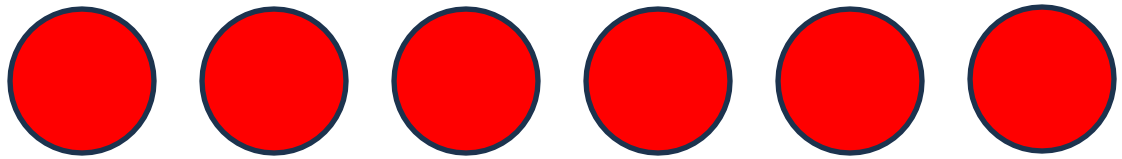
The overall odds of winning a prize are 1 in 24.87.
The odds presented here are based on a $2 play (rounded to two decimal places).

Powerball (CSCI 132 Version)

Rules:

select 6 numbers between 1 and 6

no repeats, numbers must appear in the correct order



Powerball (CSCI 132 Version)

Rules:

select 6 numbers between 1 and 6

no repeats, numbers must appear in the correct order

Potential winning numbers ex:

1	2	3	4	5	6
6	5	4	3	2	1

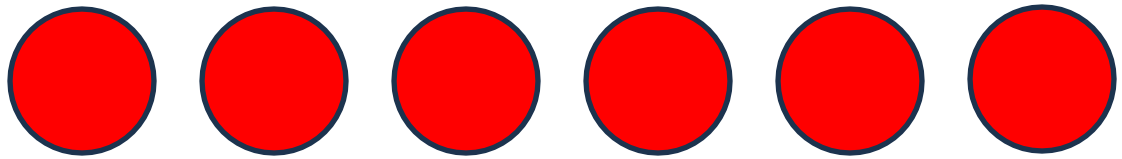
5	2	4	1	3	6
2	4	6	1	3	5

Powerball (CSCI 132 Version)

Rules:

select 6 numbers between 1 and 6

no repeats, numbers must appear in the correct order



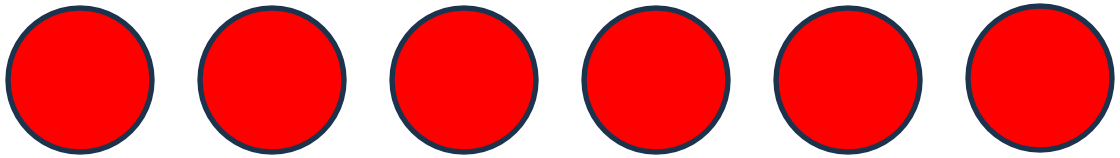
Base case?

Recursive case?

Powerball (CSCI 132 Version)

Rules:

- select 6 numbers between 1 and 6
- no repeats, numbers must appear in the correct order



Base case?

Check if the length of the current permutation matches the length of the input characters (6)

Recursive case?

Iterate through each value in our values array [1-6].

If the character is not already in the current permutation, append it and makes a recursive call to continue building the permutation

Ex: calculating permutations of “123”

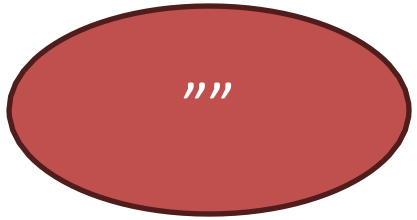
Base case:

```
len(“”) == len(“123”)
```

Recursive case:

```
for chars:
    if char not in perm:
        genPerms()
```

Ex: calculating permutations of "123"



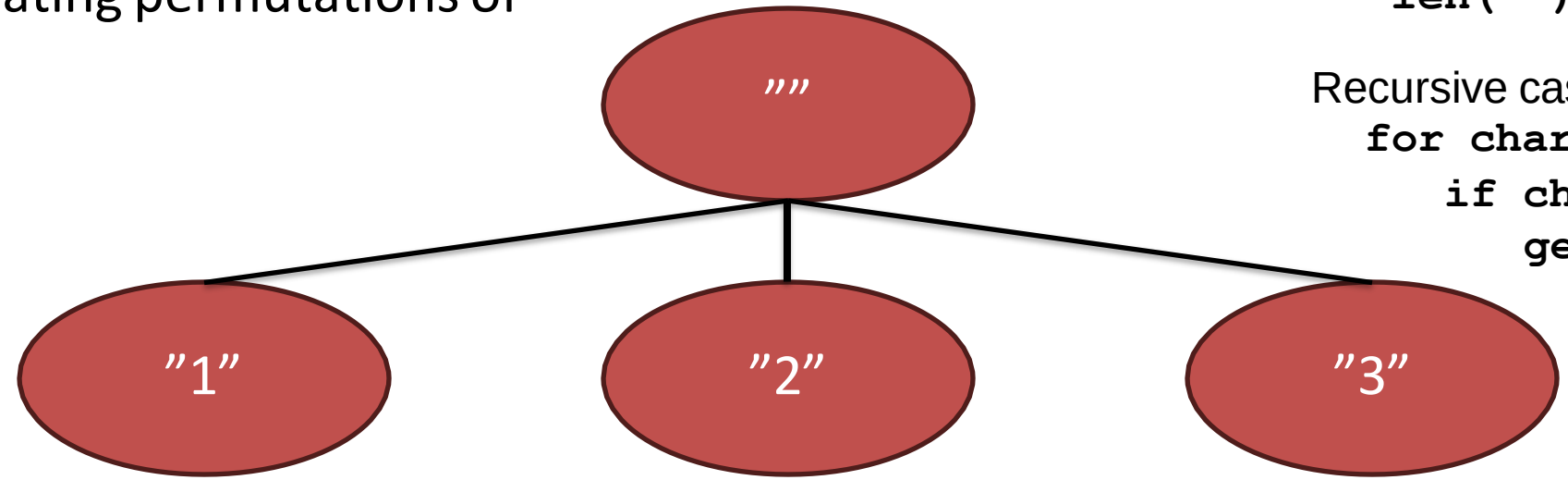
Base case:

```
len("") == len("123")
```

Recursive case:

```
for chars:
    if char not in perm:
        genPerms()
```

Ex: calculating permutations of "123"



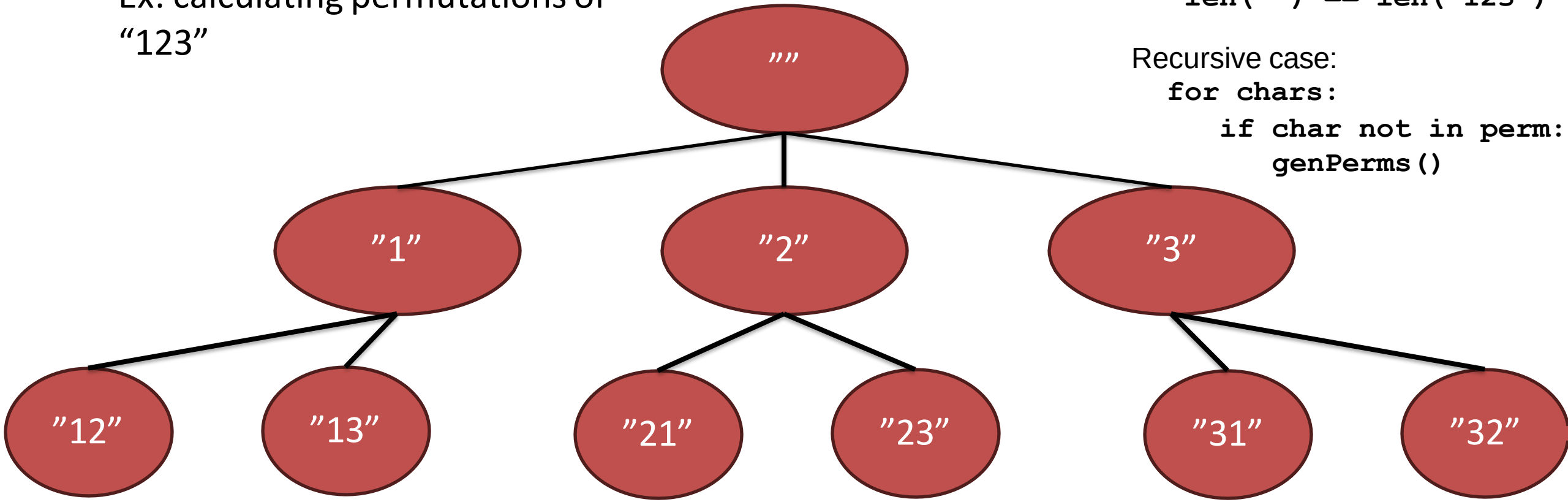
Base case:

```
len("") == len("123")
```

Recursive case:

```
for chars:
    if char not in perm:
        genPerms()
```

Ex: calculating permutations of "123"



Base case:

```
len("") == len("123")
```

Recursive case:

```
for chars:
    if char not in perm:
        genPerms()
```

Ex: calculating permutations of "123"

Base case:

```
len("") == len("123")
```

Recursive case:

```
for chars:
    if char not in perm:
        genPerms()
```