Recursion, Big-O, Tractability, & Search Algorithms

Brought to you by your TA's!

Recursion

Recursion

1. Make the problem smaller
2. Get the smaller problem's solution
    a. Pretend that recursion automatically solves the problem correctly!
3. Combine leftover solution with smaller problem's solution

Try it: Finding the sum of a list of ints

Recursion

1. **Make the problem smaller**
2. Get the smaller problem's solution
   a. Pretend that recursion automatically solves the problem correctly!
3. Combine leftover solution with smaller problem's solution

Try it: Finding the sum of a list of ints

1. Smaller problem: lst[1:]

8	2	6	9	16	35	21

Recursion:

1. Make the problem smaller
2. **Get the smaller problem's solution**
    a. **Pretend that recursion automatically solves the problem correctly!**
3. Combine leftover solution with smaller problem's solution

Try it: Finding the sum of a list of ints

1. Smaller problem: lst[1:]

8	2	6	9	16	35	21

2. 2 + 6 + 9 + 16 + 35 + 21 = 89
getSum(lst[1:]) # Smaller Result

Recursion

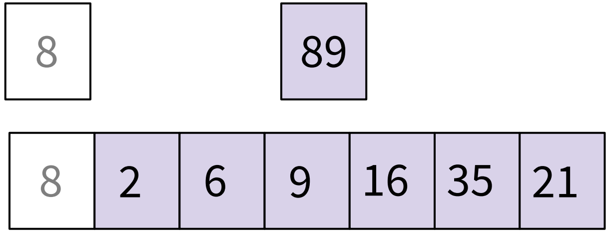
1. Make the problem smaller
2. Get the smaller problem's solution
   a. Pretend that recursion automatically solves the problem correctly!
3. **Combine leftover solution with smaller problem's solution**

Try it: Finding the sum of a list of ints

1. Smaller problem: lst[1:]

2. 2 + 6 + 9 + 16 + 35 + 21 = 89
   getSum(lst[1:])

3. 8 + 89 = 97 **lst[0] + lst[1:]**



Recursion: Get Sum of List

- Base Case(s)
  - Length of list is 0 or 1
- Recursive Case
  - smallerProblem = lst[1:]
  - smallerResult = getSum(smallerProblem)
  - add it to the rest of the input
    - lst[0] + smallerResult

```
def getSum(lst):
    if len(lst) == 0:
        return 0
    elif len(lst) == 1:
        return lst[0]
    else:
        smallerProblem = lst[1:]
        smallerResult = getSum(lst[1:])
        return lst[0] + smallerResult
```

Multiple Recursive Calls: Fibonacci

- Fibonacci Numbers
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34…
- Two Base Cases
  - 0th number: 0
  - 1st Number: 1
- Recursive Case
  - Adding the two numbers that came before to get the next number

```
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Recursion Reminders

- You have to call the function within its own body for it to be recursion
  - Make sure you're making the problem smaller when you're calling it
    - Otherwise it'll go on forever
- Your return types have to match!
  - Your base can't return an int while your recursive case returns a list

Big-O

Big-O

- Big-O: runtime it takes to execute a program based on its input
  - Simplest and tightest bound
    - O(n + 3) -> O(n)
    - O(n^2 + n) -> O(n^2)
- Common Big-O classes:
  - O(1)
  - O(logn)
  - O(n)
  - O(n^2)
  - O(2^n)

Determining Big-O

- For each line of a function, determine the runtime
  - Common O(1): print, return, >, <, initializing variables
  - Common O(n): in, .index()
  - When looking at loops:
    - Identify how many times the loop iterates
    - Identify the runtime of each line in the loop
    - Multiply the number of times the loop iterates by the longest runtime in the loop
- Runtime of the whole function is the runtime of the longest step

Loops Example

```
def f(number):
    for num in range(number):
        print(i)
def g(number):
    L = []
    for num in range(number):
        if num in L:
            L.append(num)
    return L
```

Loops Example

```
def f(number):
    for num in range(number):        O(n)
        print(i)
def g(number):
    L = []
    for num in range(number):
        if num in L:
            L.append(num)
    return L
```

Loops Example

```
def f(number):
    for num in range(number):      O(n)
        print(i)                   O(1)
def g(number):
    L = []
    for num in range(number):
        if num in L:
            L.append(num)
    return L
```

Loops Example

```
def f(number):
    for num in range(number):
        print(i)
def g(number):
    L = []
    for num in range(number):
        if num in L:
            L.append(num)
    return L
```

O(n)

O(1)

O(n) * O(1) = O(n)
for the loop

Loops Example

```
def f(number):
    for num in range(number):          O(n)      O(n) * O(1) = O(n)
                                                 for the loop
        print(i)                       O(1)
def g(number):
    L = []                             O(1)
    for num in range(number):
        if num in L:
            L.append(num)
    return L
```

Loops Example

```
def f(number):
    for num in range(number):       O(n)        O(n) * O(1) = O(n)
                                                for the loop
        print(i)                    O(1)
def g(number):
    L = []                          O(1)
    for num in range(number):       O(n)
        if num in L:
            L.append(num)
    return L
```

Loops Example

`def f(number):`		
`    for num in range(number):`	O(n)	O(n) * O(1) = O(n) for the loop
`        print(i)`	O(1)	
`def g(number):`		
`    L = []`	O(1)	
`    for num in range(number):`	O(n)	
`        if num in L:`	O(n)	
`            L.append(num)`		
`    return L`		

Loops Example

def f(number):	
for num in range(number):	O(n)
print(i)	O(1)
def g(number):	
L = []	O(1)
for num in range(number):	O(n)
if num in L:	O(n)
L.append(num)	O(1)
return L	

O(n) * O(1) = O(n)
for the loop

Loops Example

```
def f(number):
    for num in range(number):        O(n)
        print(num)                   O(1)
def g(number):
    L = []                           O(1)
    for num in range(number):        O(n)
        if num not in L:             O(n)
            L.append(num)            O(1)
    return L                         O(1)
```

O(n) * O(1) = O(n)
for the loop

Loops Example

```
def f(number):
    for num in range(number):        O(n)
        print(i)                     O(1)
def g(number):
    L = []                           O(1)
    for num in range(number):        O(n)
        if num in L:                 O(n)
            L.append(num)            O(1)
    return L                         O(1)
```

O(n) * O(1) = O(n) for the loop

O(n) * O(n) * O(1) = O(n^2) for the loop

Big-O Example

```
def addSomeNums(L):
    length = len(L)
    index = 1
    sum = 0
    while index < length:
        sum += L[index]
        index *= 2
    return sum
```

Big-O Example

```
def addSomeNums(L):
    length = len(L)              O(1)
    index = 1
    sum = 0
    while index < length:
        sum += L[index]
        index *= 2
    return sum
```

Big-O Example

```
def addSomeNums(L):
    length = len(L)          O(1)
    index = 1                O(1)
    sum = 0
    while index < length:
        sum += L[index]
        index *= 2
    return sum
```

Big-O Example

```
def addSomeNums(L):
    length = len(L)          O(1)
    index = 1                O(1)
    sum = 0                  O(1)
    while index < length:
        sum += L[index]
        index *= 2
    return sum
```

Big-O Example

```
def addSomeNums(L):
    length = len(L)          O(1)
    index = 1                O(1)
    sum = 0                  O(1)
    while index < length:    O(1) to check index < length, O(logn) iterations of the loop
        sum += L[index]
        index *= 2
    return sum
```

*note that index is multiplied by 2 every time, so the loop can run a maximum of log n times. we simplify this in big-O terms to logn

Big-O Example

def addSomeNums(L):	
length = len(L)	O(1)
index = 1	O(1)
sum = 0	O(1)
while index < length:	O(1) to check index < length, O(logn) iterations of the loop
sum += L[index]	O(1)
index *= 2	
return sum	

*note that index is multiplied by 2 every time, so the loop can run a maximum of log n times. we simplify this in big-O terms to logn

Big-O Example

```
def addSomeNums(L):
    length = len(L)           O(1)
    index = 1                 O(1)
    sum = 0                   O(1)
    while index < length:     O(1) to check index < length, O(logn) iterations of the loop
        sum += L[index]       O(1)
        index *= 2            O(1)
    return sum
```

*note that index is multiplied by 2 every time, so the loop can run a maximum of log n times. we simplify this in big-O terms to logn

Big-O Example

```
def addSomeNums(L):
    length = len(L)          O(1)
    index = 1                O(1)
    sum = 0                  O(1)
    while index < length:    O(1) to check index < length, O(logn) iterations of the loop
        sum += L[index]      O(1)
        index *= 2           O(1)
    return sum               O(1)
```

*note that index is multiplied by 2 every time, so the loop can run a maximum of log n times. we simplify this in big-O terms to logn

Big-O Example

```
def addSomeNums(L):
    length = len(L)           O(1)
    index = 1                 O(1)
    sum = 0                   O(1)
    while index < length:     O(1) to check index < length, O(logn) iterations of the loop
        sum += L[index]       O(1)
        index *= 2            O(1)
    return sum                O(1)
```

*note that index is multiplied by 2 every time, so the loop can run a maximum of log n times. we simplify this in big-O terms to logn

O(logn) overall

Big-O Example

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return x * factorial(x - 1)
```

Big-O Example

```
def factorial(x):
    if x == 1:                         O(1)
        return 1
    else:
        return x * factorial(x - 1)
```

Big-O Example

```
def factorial(x):
    if x == 1:                          O(1)
        return 1                        O(1)
    else:
        return x * factorial(x - 1)
```

Big-O Example

```
def factorial(n):
    if n == 1:                        O(1)
        return 1                      O(1)
    else:                             O(1)
        return n * factorial(n - 1)
```

Big-O Example

def factorial(n):		
if n == 1:	O(1)	
return 1	O(1)	
else:	O(1)	
return n * factorial(n - 1)	O(n)	after factorial is called for the first time, it will recursively be called n - 1 times, which is O(n)

Big-O Example

```
def factorial(n):
    if n == 1:                        O(1)
        return 1                      O(1)
    else:                             O(1)
        return n * factorial(n - 1)   O(n)
```

after factorial is called for the first time, it will recursively be called n - 1 times, which is O(n)

O(n) overall

Tractability

Tractability

- A problem is tractable if it has a reasonable efficient (polynomial) runtime. Otherwise, it is intractable.
- Polynomial runtimes:
  - O(1)
  - O(n)
  - O(logn)
  - O(nlogn)
  - O(n^2)
  - O(n^100)
- Non polynomial runtimes:
  - O(2^n)
  - O(n!)

P and NP

- P: the set of problems that can be solved in polynomial time
    - Examples
        - Linear search, binary search
        - Sorting a list
- NP: the set of problems that can be verified in polynomial time
    - Given the answer to the problem, we can verify in polynomial time that it is correct
    - Examples
        - Subset sum
        - Satisfying a circuit
        - All problems in P!

Search Algorithms

Linear Search

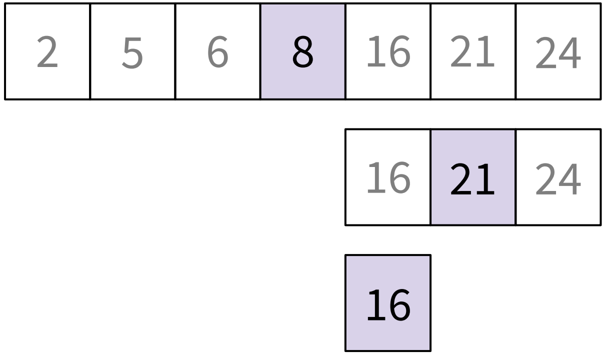
- Checks all values of input
- Best Case
  - Target is first element in list
- Worst Case
  - Target is last element of list or not in list
- What's the Big O?

```
def linearSearch(lst, target):
    for i in range(len(lst)):
        if lst[i] == target:
            return True
    return False


def recursiveLinearSearch(lst, target):
    if lst == [ ]:
        return False
    elif lst[0] == target:
        return True
    else:
        return recursiveLinearSearch(lst[1:],
    target)
```

Binary Search

- Input list has to be **sorted**
- We start by checking the **middle element**



- What's the Big O?

```
def binarySearch(lst, target):
    if lst == [ ]:
        return False
    else:
    midIndex = len(lst) // 2
    if lst[midIndex] == target:
        return True
    elif target < lst[midIndex]:
        return binarySearch(lst[:midIndex],
    target)
    else: # lst[midIndex] < target
        return binarySearch(lst[midIndex+1:],
    target)
```

Good luck on the final exam! :)