Week 6: Recursive Algorithms

CSC 236:Introduction to the Theory of Computation

Summer 2024

Instructor: Lily

Announcement

- Tutorial BA 1130 *only*
  - Tutorials now focus on more examples
- Midterm details
  - Multiple Choice and True/False: +1 if answer completely correctly, -1 if answer incorrectly. Can be left blank. *Don't guess*.
  - Short answer: no justification required.
  - Other types: 20% IDK for entire question or part of a question.
- **No office hours** during the exam season; Last office hour in June is this Friday. There will be online office hours the Friday before and on the Monday of the A3 deadline.

Recursion

Iterative algorithms:

LOOP

LI : partial correctness + termination

precond → → POST COND

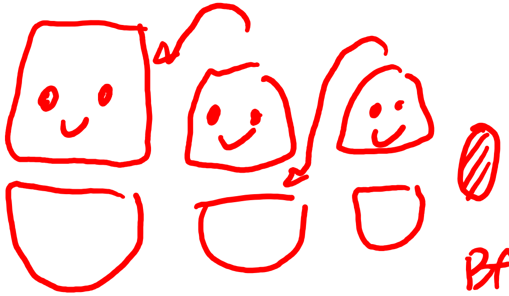
Fibonacci sequence 0, 1, 1, 2, 3, 5, ...

f_0 f_1

$$f_{n+2} = f_{n+1} + f_n$$

precond → POST COND (BASE)

→ intermediate STEP



BASE

```
def fib(n):
    # Pre: natural number n
    # Post: returns f_n
if n == 0:
    return 0
if n == 1:
    return 1
return fib(n-1) + fib(n-2)
```

base cases

recurse

From last lecture

```python
def mult(a, b):
    # Pre: a and b are natural number with at most n bits
    # Post: returns a*b
    m = 0
    count = 0
    while count < b:
        m += a
        count += 1
    return m
```

$\rightarrow O(1)$

$O(b)$ iterations

$\rightarrow O(1)$

TOTAL: $O(b)$

Ex. mult(1, 100) → $c \cdot 100$ steps

mult(1, 1000) → $c \cdot 1000$ steps.

n = length of input

$\log_{10} a + \log_{10} b$.

Running time WRT n:

$\Omega(2^n)$

eg mult(1, 1000) input size ~4

number of steps ~1000

Elementary Multiplication

```python
def mult_elementary(a, b):
    # Pre: a and b are natural number with at most n bits
    # Post: returns a*b
m = 0
while b > 0:
    m += a * (b % 10)
    b //= 10 # int division
return m
```

$m = 0 \rightarrow O(1)$

$\text{while } b > 0 \rightarrow O(\overbrace{\log_{10} b}^{n})$ iterations

$O(n)$

$O(1)$

TOTAL: $O(n^2)$

$$
\begin{array}{rl}
\underbrace{10\,24} & \\
\times\ 1729 & \\
\hline
9216 & \leftarrow 1024 \times 9 \\
20480 & \leftarrow 1024 \times 20 \\
716800 & \leftarrow 1024 \times 700 \\
1024000 & \leftarrow 1024 \times 1000 \\
\hline
1770496 &
\end{array}
$$

Fast Multiplication (Karatsuba's Algorithm)

```
def karatsuba(a, b):
    # Pre: a and b are natural number with at most n digits
    # Post: returns a*b
    if |a| == 1 and |b| == 1:
        return a*b
    a1, a2 = a[0..n/2], a[n/2..n]
    b1, b2 = b[0..n/2], b[n/2..n]
    p1 = karatsuba(a1, b1)
    p2 = karatsuba(a1+a2, b1+b2)
    p3 = karatsuba(a2, b2)
    return p1*10^{n} + (p2-p1-p3)*10^{n/2} + p3
```

a has 1 digit b has 1 digit

$O(1)$

$a = 49 \quad (|a| = 2 \quad a[0] = 4 \quad a[1] = 9)$

$b = 98 \quad (|b| = 2 \quad a[0] = 9 \quad b[1] = 8)$

$(4 \cdot 10 + 9) \cdot (9 \cdot 10 + 8)$

$\overset{36}{4 \cdot 9}(100) + (9 \cdot 9 + 4 \cdot 8) \cdot 10 + \overset{72}{9 \cdot 8}$

$\underline{a_0 b_0} \qquad \boxed{a_1 b_0 \quad a_0 b_1} \qquad \underline{a_1 b_1}$

compute

compute

$(a_0 + a_1) \cdot (b_0 + b_1)$ ←

compute

$\Rightarrow a_0 b_0 + \boxed{a_1 b_0 + a_0 b_1} + a_1 b_1$

$(4+9) \cdot (9+8) = 13 \cdot 17 = 170 + 51$
$= 221$

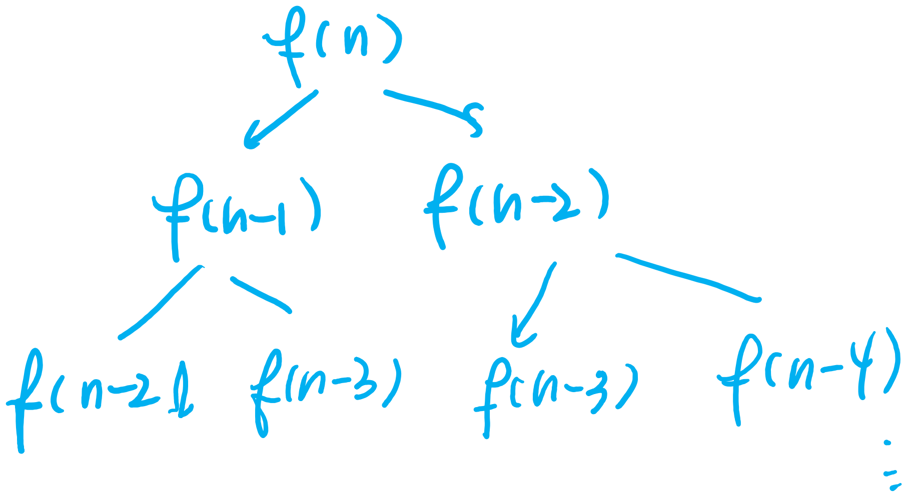
$a_1 b_0 + a_0 b_1 = 221 - 36 - 72$
$= 113$

$36(100) + 113(10) + 72$

Now your turn!

1. When computing f_n using the algorithm shown to the right how many times does `fib(k)` get called for $0 \leq k \leq n$?
2. Use Karatsuba's algorithm to multiply together the number 1024 and 1729.

```python
def fib(n):
    # Pre: natural number n
    # Post: returns f_n
if n == 0:
    return 0
if n == 1:
    return 1
return fib(n-1) + fib(n-2)
```



Q1. When computing f_n using the algorithm shown to the right how many times does `fib(k)` get called for $0 \le k \le n$?

function	# of calls	
fib(n)	1	f_1
fib(n-1)	1	f_2
fib(n-2)	2	f_3
fib(n-3)	3	f_4
fib(n-4)	5	f_5
⋮		

```
def fib(n):
  # Pre: natural number n
  # Post: returns f_n
if n == 0:
    return 0
if n == 1:
    return 1
return fib(n-1) + fib(n-2)
```

$$F = \left[\begin{array}{ccc} f_0 & f_1 & f_2 \\ 0 & 1 & 1 \end{array}\right.$$

proof by induction

generally fib(k) called f_{n-k+1} times.

$\sim \frac{\varphi^n}{2}$

Q2. Use Karatsuba's algorithm to multiply together the number 1024 and 1729.

```
def karatsuba(a, b):
    # Pre: a and b are natural number with at most n digits
    # Post: returns a*b
if |a| == 1 and |b| == 1:
  return a*b
a1, a2 = a[0..n/2], a[n/2..n]
b1, b2 = b[0..n/2], b[n/2..n]
p1 = karatsuba(a1, b1)
p2 = karatsuba(a1+a2, b1+b2)
p3 = karatsuba(a2, b2)
return p1*10^{n} + (p2-p1-p3)*10^{n/2} + p3
```

kar(1024, 1729)

$a_1 = [1, 0]$ $a_2 = [2, 4]$

$b_1 = [1, 7]$ $b_2 = [2, 9]$

$P_1 = \text{kar}(10, 17)$ $n = 2$

$a_1 = [1]$ $a_2 = [0]$

$b_1 = [1]$ $b_2 = [7]$

$p_1 = 1$ $p_2 = 8$ $p_3 = 0$

return $100 + 70 + 0 = 170$

$P_2 = \text{kar}(34, 46) = 1564$

$P_3 = \text{kar}(24, 29) = 696$

return $1700000 + (1564 - 866)100 + 696$

Fast Multiplication (Karatsuba's Algorithm)

```
def karatsuba(a, b):
    # Pre: a and b are natural
number with at most n digits
    # Post: returns a*b
    if |a| == 1 and |b| == 1:
(*)   return a*b
    a1, a2 = a[0..n/2], a[n/2..n]
    b1, b2 = b[0..n/2], b[n/2..n]
    p1 = karatsuba(a1, b1)
    p2 = karatsuba(a1+a2, b1+b2)
    p3 = karatsuba(a2, b2)
    return p1*10^{n} + (p2-p1-
p3)*10^{n/2} + p3
```

$\to O(1)$

$O(1)$

3x recursive calls.

$O(1)$

assume a and b have length n

$T(n)$ = running time of Karatsuba's algorithm on <u>these inputs</u>.

$|a_1|, |a_2|, |b_1|, |b_2|, |a_1+a_2|, |b_1+b_2| = n/2$

$$T(n) = O(1) + T(n/2) \cdot 3$$

$P(n) :=$ Karatsuba(a,b) returns $a \cdot b$ w/ precond $|a|, |b| \le n$

base. if $n = 1$ then true by def $(*)$

Inductive. IH $Kar(a_1,b_1)$, $Kar(a_2,b_2)$, $Kar(a_1+a_2, b_1+b_2)$ OK

$$a \cdot b = (a_1 \cdot 10^{n/2} + a_2)(b_1 \cdot 10^{n/2} + b_2)$$

$$= a_1 b_1 \cdot 10^n + (a_1 b_2 + a_2 b_1) \cdot 10^{n/2} + a_2 b_2$$

$-a_1 b_1 - a_2 b_2$

Quick Sort

assume unique elements only

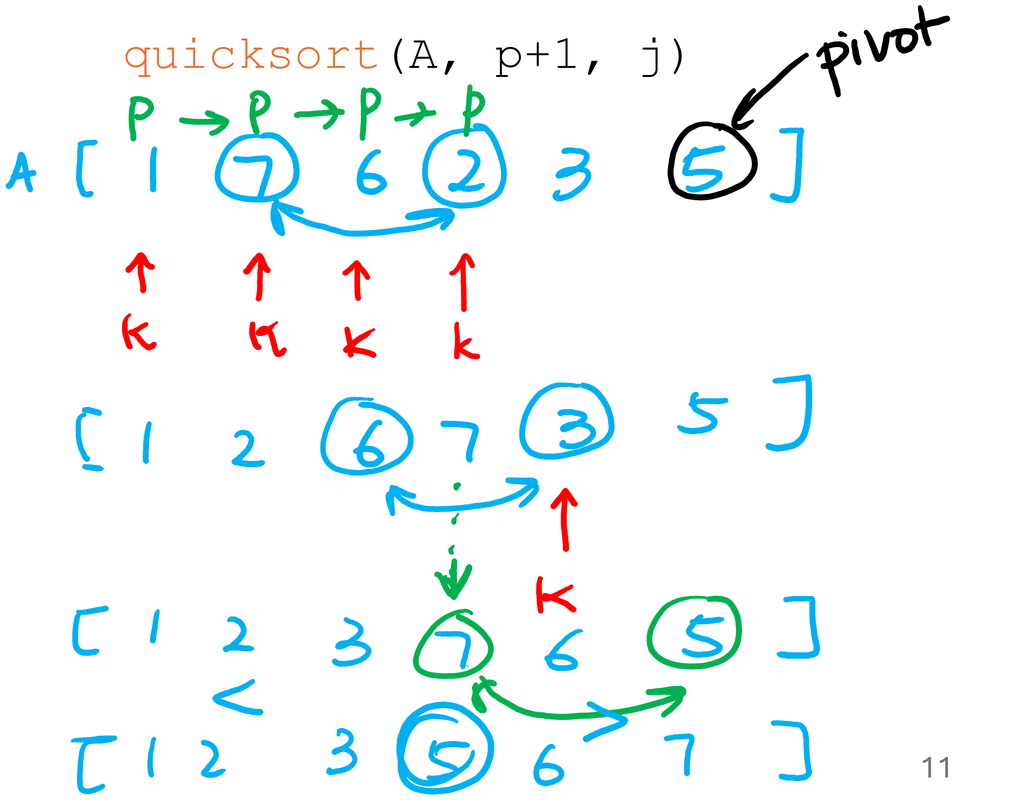
```
def partition(A, i, j):
  # Pre: i, j indices of A (i <= j)
  # Post: index p so that A[k] < A[p] for
  # k = i, …, p-1 and A[l] > A[p] for
  # l = p+1, …, j
  p = i
  pivot = A[j-1]
  for k in range(i, j-1):
    if pivot > A[k]:
      swap(A, p, k)
      p += 1
  swap(A, p, j-1)
  return p
```

last element in interval is pivot

initial call: quickSort(A, 0, n)

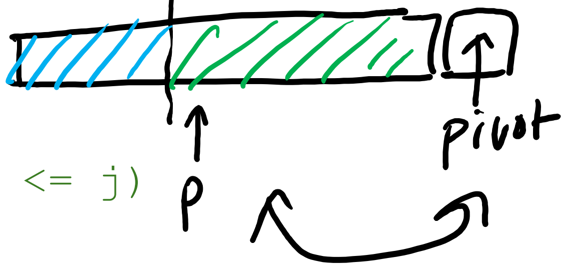
```
def quicksort(A, i, j):
  # Pre: list A. i, j indices
  # Post: A is sorted
  if j-i <= 1:
    return
  p = partition(A, i, j)
  quicksort(A, i, p)
  quicksort(A, p+1, j)
```

for A in [i,…,j)



```
def partition(A, i, j):
  # Pre: i, j indices of A (i <= j)
  # Post: index p so that A[k] < A[p] for
  # k = i, …, p-1 and A[l] > A[p] for
  # l = p+1, …, j
  p = i
  pivot = A[j-1]
  for k in range(i, j-1):
    if pivot > A[k]:
      swap(A, p, k)
 (*)  p += 1
  swap(A, p, j-1)
  return p
```

TERMINATION:



swap.

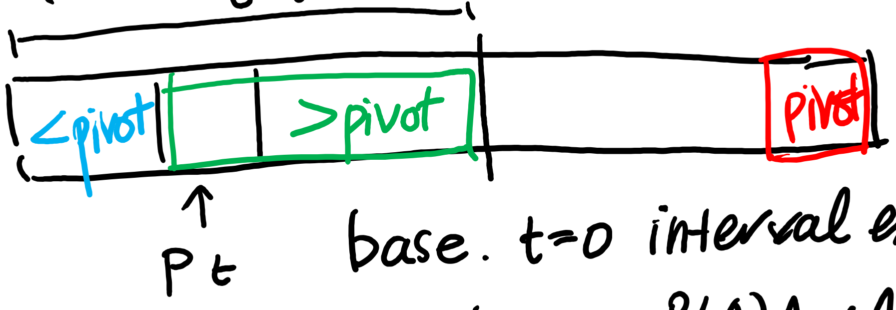
```
def quicksort(A, i, j):
  # Pre: i, j indices of A (i <= j)
  # Post: A is sorted
  if j-i <= 1:
    return                      → O(1)
  p = partition(A, i, j)        → O(n)
  quicksort(A, i, p)            length (p-i)
  quicksort(A, p+1, j)          length (j-p-1)
```

proof of correctness for partition

variable: - $A_t[i, j]$ values of $A[i,j]$ after iter t

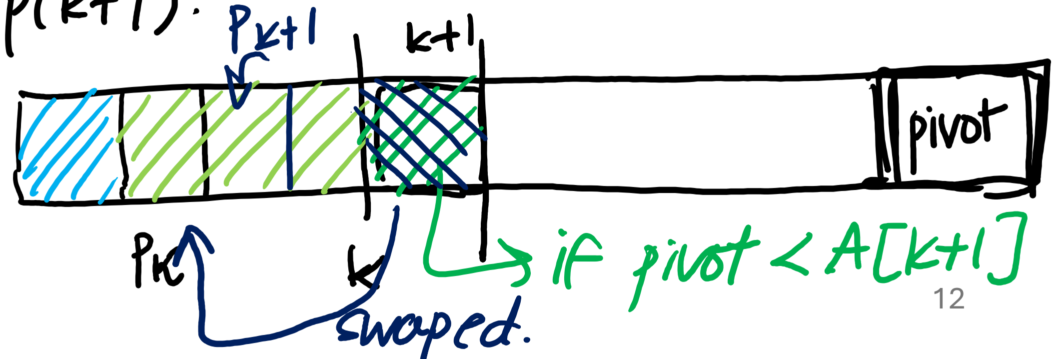
- p_t value of p after iteration t

$P(t)$: $(\underbrace{t \le j-i-1}_{t}) \Rightarrow$



base. $t=0$ interval empty.

inductive case: suppose $k \ge 0$, $P(0) \wedge \dots \wedge P(k)$ WTS $P(k+1)$.



Quick Sort (Worst Case)

$n = j - i$

```
def partition(A, i, j):
  # Pre: i, j indices of A (i <= j)
  # Post: index p so that A[k] < A[p] for
  # k = i, …, p-1 and A[l] > A[p] for
  # l = p+1, …, j
  p = i
  pivot = A[j-1]
  for k in range(i, j-1):
    if pivot > A[k]:
      swap(A, p, k)
      p += 1
  swap(A, p, j-1)
  return p
```

$\rightarrow O(1)$

$\rightarrow O(n)$ iter.

$\rightarrow O(1)$

$\rightarrow O(1)$

TOTAL: $\Theta(n)$

in this case $p = j-1$ (last position)

then # A already sorted.

$$T(n) = \Theta(n) + T(n-1) + T(1)$$

in case $p = n/2$ (middle position)

$$T(n) = \Theta(n) + 2T(n/2)$$

Recap

- Recursive vs iterative algorithms
- Classic Multiplication $O(n^2)$
- Karatsuba Multiplication $T(n) = O(1) + 3T(n/2)$
- Quick Sort
  - → worst case $T(n) = \Theta(n) + T(n-1) + T(1)$
  - → average case $T(n) = \Theta(n) + 2T(n/2)$

Next time… running time of recursive algorithms via the Master Method