Lecture 12: Maximization

COSC 311 *Algorithms,* Fall 2022

Overview

1. Recurring Recurrences
2. Maximizing Profit

Last Time

The "Master Theorem"

- Given recurrence $T(n) = aT(n/b) + f(n)$
- Define $c = \log_b a$
- Three cases:

1. If $f(n) = O(n^d)$ for $d < c$ then $T(n) = O(n^c)$
2. If $f(n) = \Theta(n^c \log^k n)$ then $T(n) = O(n^c \log^{k+1} n)$
3. If $f(n) = \Omega(n^d)$ for $d > c$, then $T(n) = O(f(n))$

generic soln to recurrences

# of sm. pr

size of smaller problems

time to combine

running for inst. of size n

$n^c \approx$ # of recursive calls

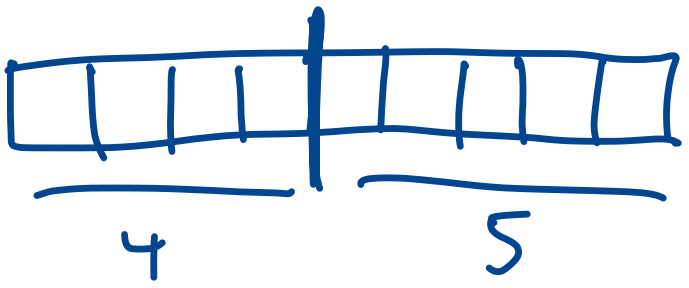
$k=0$

For Binary Search

$f(n) = \Theta(n^0) = \Theta(n^c)$

$\Theta(\log n)$

n = 9



A Technical Note

In formal statement we have

- $T(n) = aT(n/b) + f(n)$

In practice, might have

- $T(n) = aT(\lceil n/b \rceil) + f(n)$

$\lfloor n/2 \rfloor$ floor = round down

$\lceil n/2 \rceil$ ceiling = round up

The conclusion of the theorem still holds in this case!

Master Theorem for Binary Search

does a contain val?

$$T(n) = aT(n/b) + f(n)$$

sorted a:


```
BinarySearch(a, val, i, j):
  if j = i then return false
  if j - i = 1 then return (a[i] = val)
  m <- (j + i) / 2
  if a[m] > val then
    return BinarySearch(a, val, i, m)
  else
    return BinarySearch(a, val, m, j)
  endif
```

base case

search left

search right

$f(n) = \Theta(1) = \Theta(n^0) \Rightarrow d = 0$

$a = 1$

$b = 2$

$c = \log_b a = \log_2 1 = 0$

$\Rightarrow$ running time $\Theta(\log n)$

Master Theorem for MergeSort

$$T(n) = aT(n/b) + f(n)$$

$n = j - i$

```
MergeSort(a, i, j):
  if j - i = 1 then
    return
  endif
  m <- (i + j) / 2
  MergeSort(a,i,m)
  MergeSort(a,m,j)
  Merge(a,i,m,j)
```

base case

$\Theta(1)$

$\Theta(n)$

$\Theta(n)$

$a = 2$

$b = 2$

$\Rightarrow c = \log_b a = \log_2 2 = 1$

running time is $\Theta(n \log n)$

Master thm: land in case 2

Profit Maximization

Goal. Pick day b to buy and day s to sell to maximize profit.

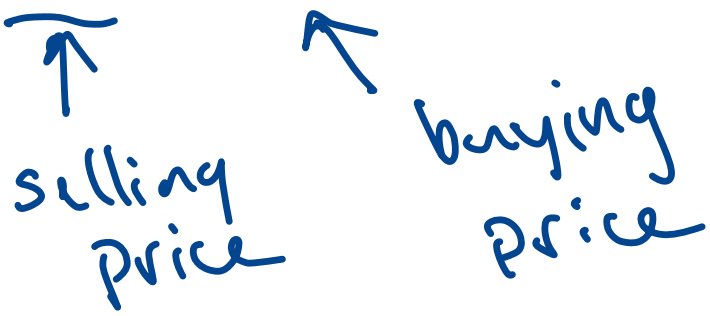
Formalizing the Problem

Input. Array a of size n

- $a[i]$ = price of Alphabet stock on day i

Output. Indices b (buy) and s (sell) with $1 \leq b \leq s \leq n$ that maximize profit

- $p = a[s] - a[b]$



Simple Procedure

Devise a procedure to determine max profit in time $O(n^2)$.

Brute force: b buy day
s sell day
$b \leq s$

consider all pairs of days!

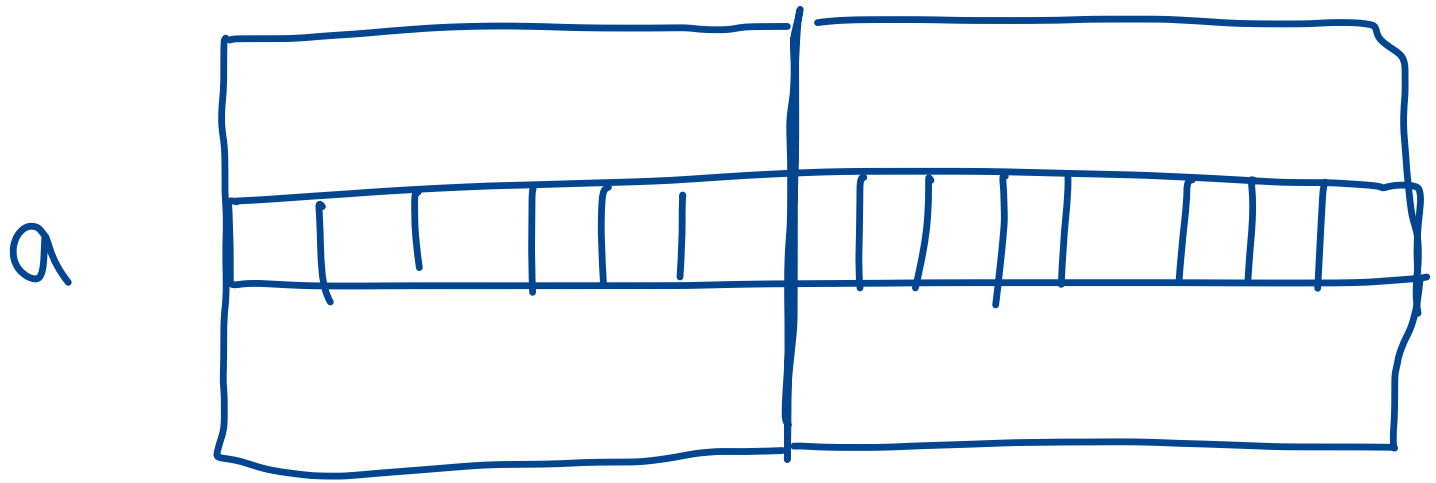
```
max ← 0
for b = 1 up to n                          n iter.
    for s = b up to n                      ≤ n iter
        if a[s] − a[b] > max then          O(1)
            max ← a[s] − a[b]              O(n)
return max
```

$\Rightarrow O(n^2)$

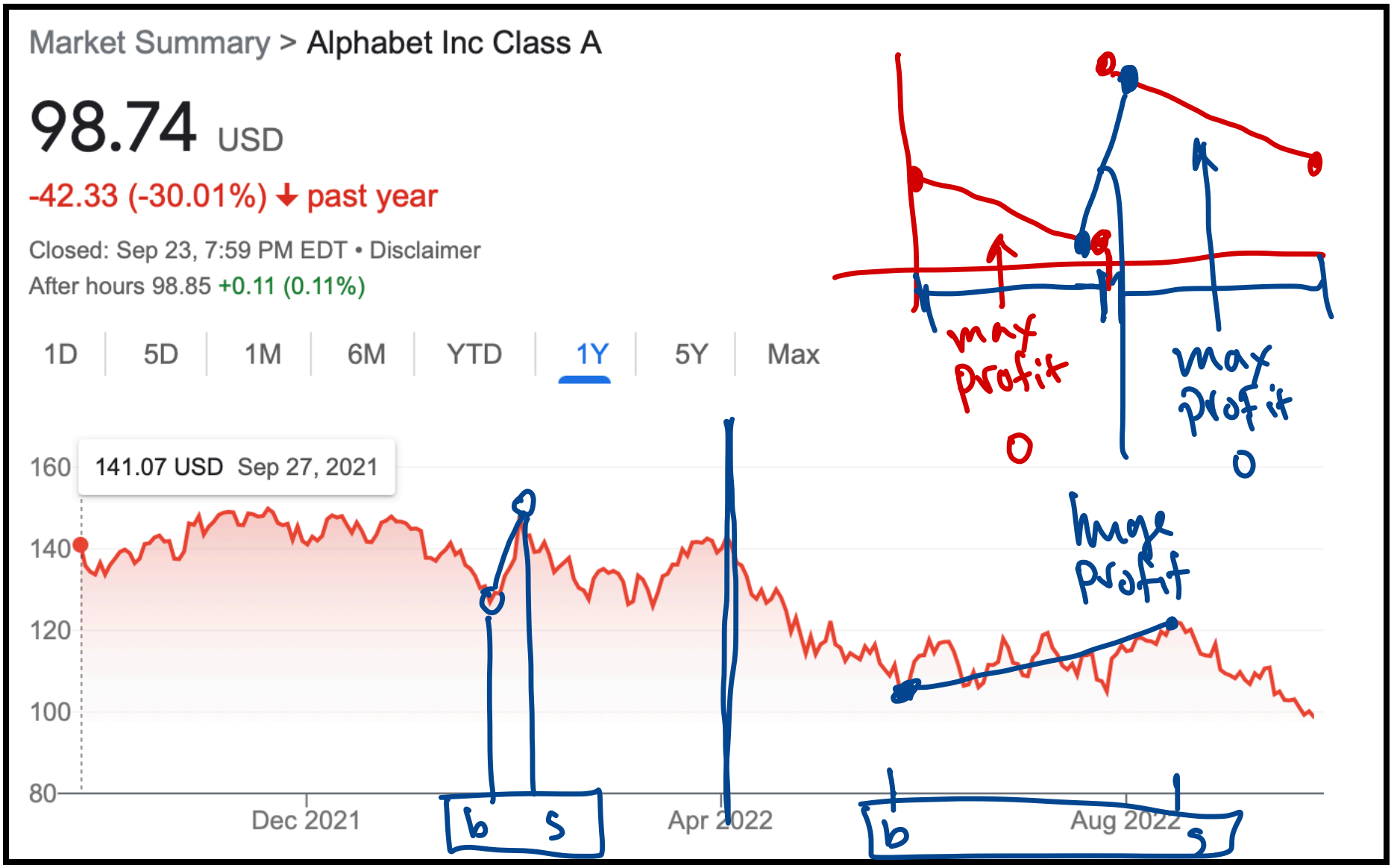
Divide and Conquer?

Question. Can we compute maximum profit faster?

- Use divide an conquer?



Maximum Profit via D&C?



Issue w/ dpc:

cut input in half left / right half of array

recursion → (1) max profit w/ b, s in left

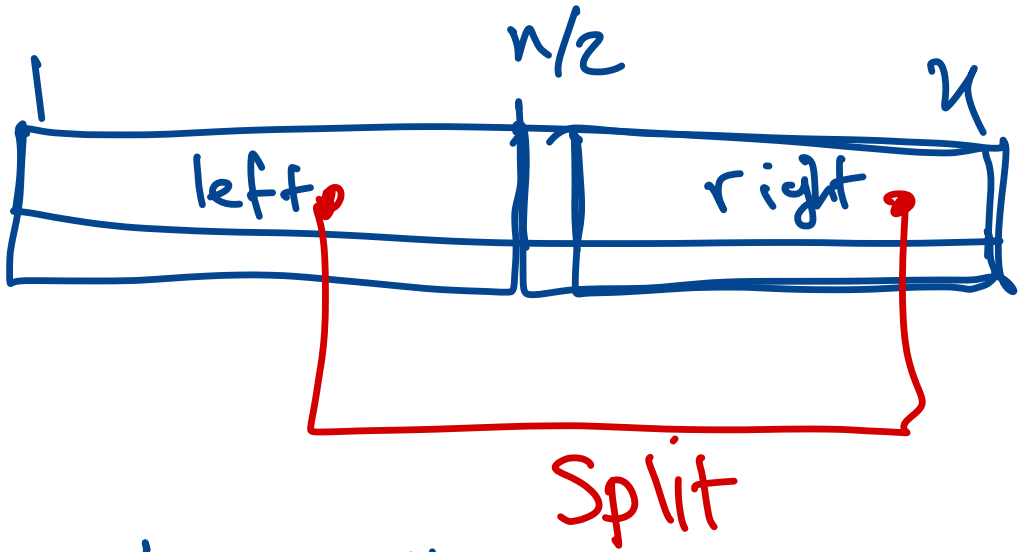
→ (2) ~~——————————————~~ in right

→ (3) max profit w/ b in left
s in right

Three Cases

Array a of size n, midpoint $m = n/2$

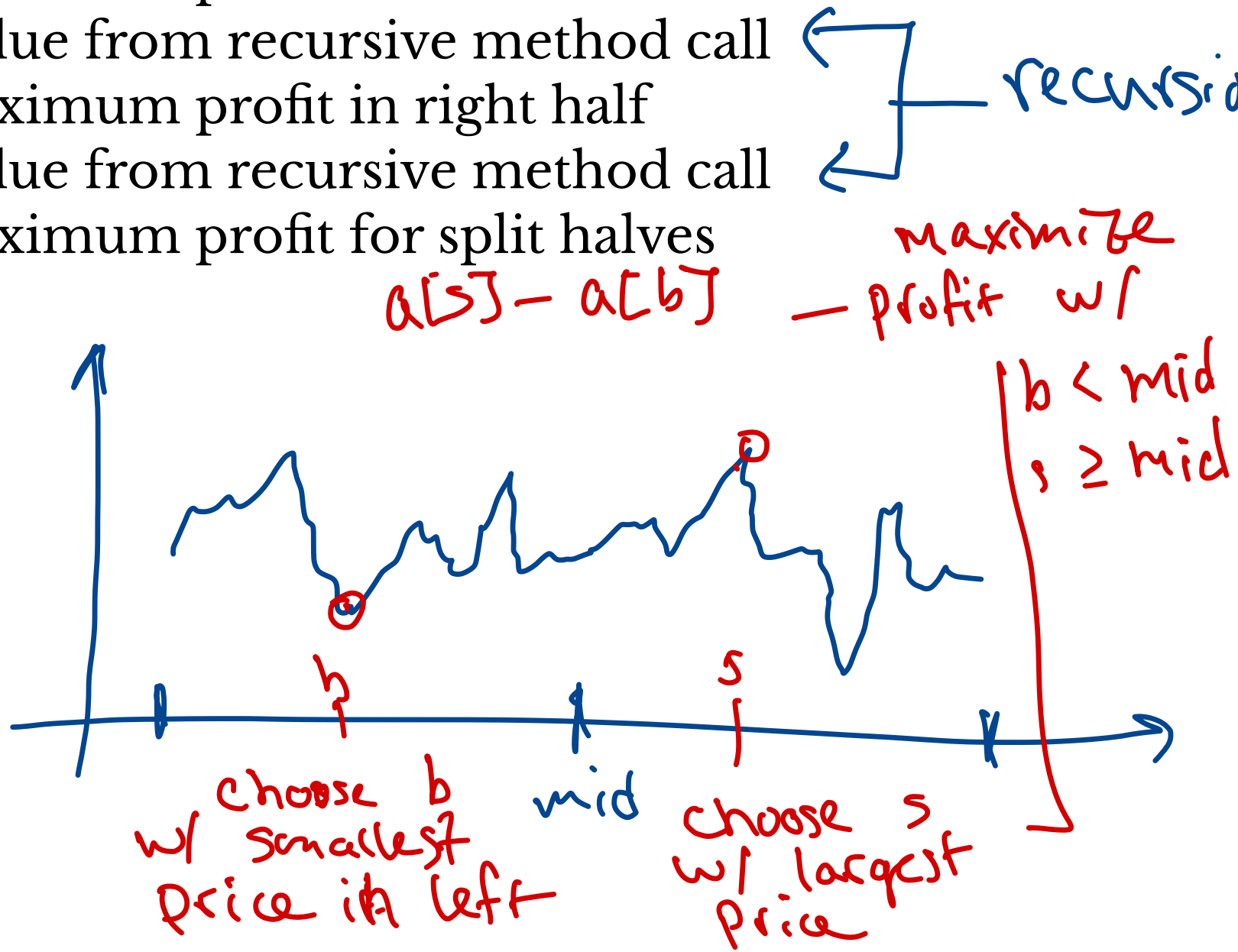
1. Maximum profit in left half: $b, s < n/2$
2. Maximum profit in right half: $n/2 \leq b, s$
3. Maximum profit splits the halves: $b < n/2 \leq s$



Strategy: try all three cases, return best profit of three.

Algorithmic Approach

1. Find maximum profit in left half
   - get value from recursive method call
2. Find maximum profit in right half
   - get value from recursive method call
3. Find maximum profit for split halves
   - how?



Observation

To find the maximum profit for split halves:

1. find the lowest price $a[b]$ for $b < n/2$
2. find the highest price $a[s]$ for $s \geq n/2$

Maximum profit is $a[s] - a[b]$.

Note: values can be found in $O(n)$ time.

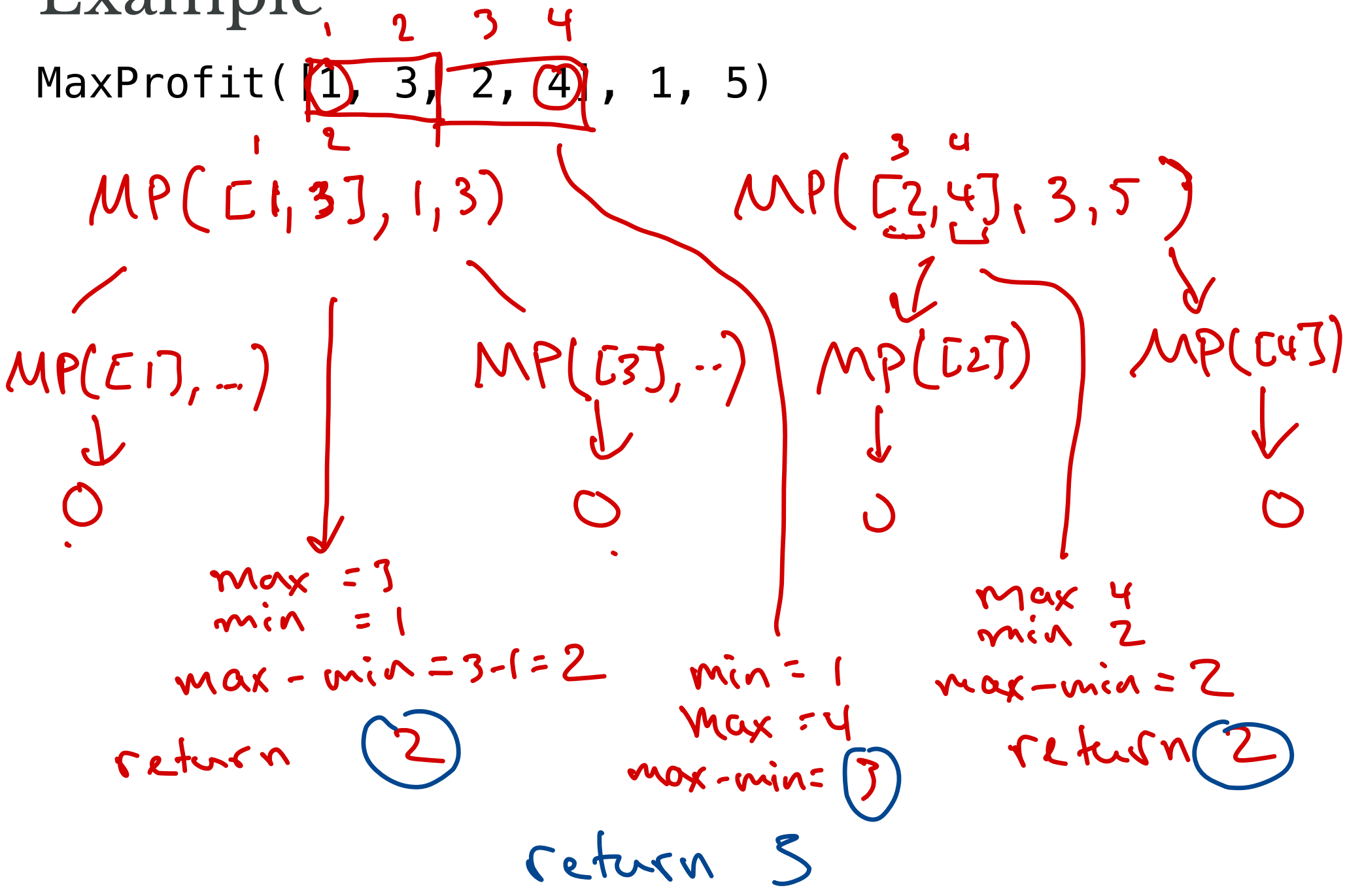
Maximization Pseudocode

```
# find maximum profit achievable for b, s
# satisfying i <= b <= s < j

MaxProfit(a, i, j):
  if j - i = 1 then return 0
  m <- (i + j) / 2
  left <- MaxProfit(a, i, m)
  right <- MaxProfit(a, m, j)
  min <- FindMin(a, i, m)
  max <- FindMax(a, m, j)
  return Max(left, right, max - min)
```

Base case
mid pt.
recursive calls
O(n)

Example


MaxProfit([1, 3, 2, 4, 1, 5])
1 2 3 4
1 2
MP([1,3], 1, 3)
3 4
MP([2,4], 3, 5)
MP([1], ...)
MP([3], ...)
MP([2])
MP([4])
0
0
0
0
max = 3
min = 1
max - min = 3-1 = 2
return 2
min = 1
max = 4
max-min = 3
max 4
min 2
max-min = 2
return 2
return 5

Algorithm Correctness

Claim. `MaxProfit(a, i, j)` returns the maximum value of $a[s] - a[b]$ over all s, b satisfying $i \le b \le s < j$.

Proof. Argue by induction on $n = j - i$ = size of the call.

```
MaxProfit(a, i, j):
  if j - i = 1 then return 0
  ...
```

Base case. $n = 1$.

Val returned is 0

= Max profit b/c $s = b = i$

$a[s] - a[b] = a[i] - a[i] = 0$

Inductive Step I

```
MaxProfit(a, i, j):
  ...
  left <- MaxProfit(a, i, m), right <- MaxProfit(a, m, j)
  min <- FindMin(a, i, m), max <- FindMax(a, m, j)
  return Max(left, right, max - min)
```

max in case 1

max in case 2

max in case 3

Claim. `MaxProfit(a, i, j)` returns the maximum value of $a[s] - a[b]$ over all s, b satisfying $i \leq b \leq s < j$.

Three possible cases...

(1) max occurs w/ $b, s < m$

(2) max occurs w/ $b, s \geq m$

(3) max occurs w/ $b < m \leq s$

Inductive Step II

```
MaxProfit(a, i, j):
  ...
  left <- MaxProfit(a, i, m), right <- MaxProfit(a, m, j)
  min <- FindMin(a, i, m), max <- FindMax(a, m, j)
return Max(left, right, max - min)
```

Claim. `MaxProfit(a, i, j)` returns the maximum value of $a[s] - a[b]$ over all s, b satisfying $i \leq b \leq s < j$.

Inductive hypothesis. Claim holds for all sizes $< n$.

Inductive step. Must show claim holds for size n.

- IH $\implies$ `left` is max profit in `a[i..m-1]`
- IH $\implies$ `right` is max profit in `a[m..j-1]`
- `max - min` is max profit with $b < m \leq s$

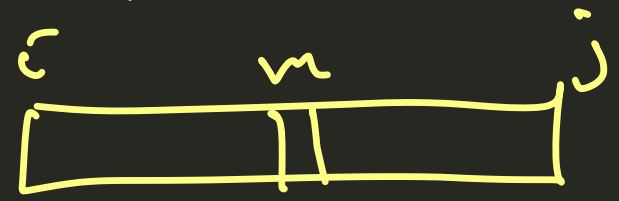
left and right store max profits on left and right half

Running Time Rercurrence?

$$T(n) = aT(n/b) + f(n)$$

```
# find maximum profit achievable for b, s
# satisfying i <= b <= s < j

MaxProfit(a, i, j):
  if j - i = 1 then return 0
  m <- (i + j) / 2
  left <- MaxProfit(a, i, m)
  right <- MaxProfit(a, m, j)
  min <- FindMin(a, i, m)
  max <- FindMax(a, m, j)
  return Max(left, right, max - min)
```



$\Theta(1)$

$\Theta(n)$

$\Theta(1)$

2

2

$f(n) = \Theta(n) = \Theta(n^1) =$

$c = \log_b a = \log_2 2 = 1$

Brute force $\Theta(n^2)$

Case 2 of Master Thm $\Rightarrow$ r.t. $O(n \log n)$

Applying Master Theorem

- Given recurrence $T(n) = aT(n/b) + f(n)$
- Define $c = \log_b a$
- Three cases:

1. If $f(n) = O(n^d)$ for $d < c$ then $T(n) = O(n^c)$
2. If $f(n) = \Theta(n^c \log^k n)$ then $T(n) = O(n^c \log^{k+1} n)$
3. If $f(n) = \Omega(n^d)$ for $d > c$, then $T(n) = O(f(n))$

Exercise

Modify the code to return the indices `b` and `s` that maximize profit.

```
# find maximum profit achievable for b, s
# satisfying i <= b <= s < j

MaxProfit(a, i, j):
  if j - i = 1 then return 0
      m <- (i + j) / 2
      left <- MaxProfit(a, i, m)
      right <- MaxProfit(a, m, j)
      min <- FindMin(a, i, m)
      max <- FindMax(a, m, j)
      return Max(left, right, max - min)
```

Exercise. Can you solve in $O(n)$ time?