CSCI 3104: Algorithms

Lecture 12: Midterm Exam 1 Review

Rachel Cox

Department of Computer Science

Starting at 9:35am

What will we learn today?

- ❑ Review of All Topics covered so far.

Exam 1 Info

- February 24 6PM - February 25 8PM Need to start by 5:30pm
- 2.5 hours
- Access on Gradescope
- Practice the upload process with the practice exam submission

Insertion Sort

Example: Use the Insertion Sort algorithm to sort the following array: $A = [4, 6, 1, 2, 11, 9]$

A[1] = 4

j = 2

k = A[2] = 6

i = 1

while 1 > 0 and A[1] > K

A[2] = K [4 6 1 2 11 9]

j = 3

*k = A[3] = 1

i = 2

while 2 > 0 and A[2] > 1

A[3] = A[2] [4 6 6 2 11 9]

i = i - 1

while 1 > 0 and A[1] > 1

A[2] = A[1] [4 4 6 2 11 9]

i = 1 - 1 = 0

A[1] = 1 [1 4 6 2 11 9]

```
procedure InsertionSort(A)
    for j in 2...len(A):
        k = A[j]
        i = j-1
        while i>0 and A[i]>k:
            A[i+1] = A[i]
            i = i-1
        •A[i+1] = k
```

Loop Invariants

Example: Prove the correctness of the algorithm with a loop invariant proof.

procedure *multiply*(m, n: integers)

S_1 { **if** $n < 0$ **then** $a := -n$ / **else** $a := n$ } $a = |n|$

S_2 { $k := 0$ / $x := 0$ } initialize

S_3 { **while** $k < a$: $x := x + m$, $k := k + 1$ }

S_4 { **if** $n < 0$ **then** $product := -x$ / **else** $product := x$ }

return *product*

{*product* equals mn}

Scratch work:

- $k=0$, $x=0$
- $x = x + m$
- $k=1$ $x = m \cdot 1$
- $k=2$ $x = 2m$
- $k=3$ $x = 3m$
- ⋮

Loop Invariant:

$x = mk$ and $k \le a$

Initialization:

From S_1, we know that $a = |n|$

From S_2, we know $k=0$, $x=0$

Prior to first execution of while loop. $0=0$

- $x = m \cdot 0$
- $a = |n| \geq 0$

Because $k=0 \Rightarrow x = m \cdot 0 = 0$ $0=0$ ✓

Loop Invariants

Example: Prove the correctness of the algorithm with a loop invariant proof.

procedure *multiply*(m, n: integers)

S_1 **if** $n < 0$ **then** $a := -n$ **else** $a := n$

S_2 $k := 0$; $x := 0$

S_3 **while** $k < a$: $x := x + m$; $k := k + 1$

S_4 **if** $n < 0$ **then** $product := -x$ **else** $product := x$

return *product*

{*product* equals mn}

Maintenance: At the start of the j^{th} iteration of the while loop

- $x = m(j-1)$ $\quad k = j-1 \le a = |n|$

In one run of the loop:

$$x = x + m = m(j-1) + m = mj - m + m$$

$$x = mj$$

$$k = k + 1 = j - 1 + 1 = j$$

Thus prior to the $j+1$ iteration, we have $x = mj$, $k = j \le a$

Loop Invariants

Example: Prove the correctness of the algorithm with a loop invariant proof.

procedure *multiply*(m, n: integers)

S_1 **if** $n < 0$ **then** $a := -n$ **else** $a := n$

S_2 $k := 0$; $x := 0$

S_3 **while** $k < a$: $x := x + m$; $k := k + 1$

S_4 **if** $n < 0$ **then** $product := -x$ **else** $product := x$

return *product*

{*product* equals mn}

Termination:

At the start of the last loop,

$K = a - 1 \leq a$

$x = m(a-1)$

After running the last loop

$x = m(a-1) + m$

$= ma - m + m$

$= ma$

$K = K + 1 = a - 1 + 1 = a$

The while loop exits since $K = a \geq a$.

Since $a = |n|$ we have two cases:

Loop Invariants

Example: Prove the correctness of the algorithm with a loop invariant proof.

$|n| = -(-n)$

procedure *multiply*(m, n: integers)

S_1 **if** $n < 0$ **then** $a := -n$ **else** $a := n$

S_2 $k := 0$; $x := 0$

S_3 **while** $k < a$: $x := x + m$; $k := k + 1$

S_4 **if** $n < 0$ **then** $product := -x$ **else** $product := x$

return *product*

{*product* equals mn}

Case 1 : If $n < 0$,

product $-ma = -m \cdot |n| = -m(-n)$

$= mn$

Case 2 : If $n > 0$

product $= ma = m|n| = mn$

Thus the product is mn as desired.

Because S_1, S_2, S_3, S_4 terminate we have verified the correctness of the algorithm.

Concluding

Induction

Example: Use mathematical induction to show that $2^n > n^2 + n$ whenever n is an integer greater than 4.

Pf: We prove by weak induction.

Base Case: $n=5$

$2^5 = 32$ $\quad$ $5^2+5 = 30$

$32 > 30$ ✓ $\quad$ So the Base Case holds.

Inductive Step: Assume $2^k > k^2+k$ for some $k \geq 5$

Show: $2^{k+1} > (k+1)^2 + (k+1)$

$$(k+1)^2 + (k+1) = k^2+2k+1 + k+1$$
$$= k^2 + k + 2k + 2$$
$$< 2^k + (2k+2)$$
$$< 2^k + 2^k$$
$$= 2 \cdot 2^k$$
$$= 2^{k+1}$$

By inductive hypothesis because $k \geq 5$]

Sub: prof by induction

Since $k \geq 5$

$\Rightarrow 2k \geq 10$

$2k+2 \geq 12$

$k \geq 5 \Rightarrow 2^k \geq 2^5$

$2^k \geq 32$

Induction

Example: Use mathematical induction to show that $2^n > n^2 + n$ whenever n is an integer greater than 4.

conclusion:

Thus we've proven by weak induction that

$2^n > n^2 + n$ when $n > 4$.

Asymptotic Analysis – Big-O, Big-Ω, Big-Θ, little-o, little-ω

Example: Given the function $f(n) = 23n^3 \log n + 5n^2 \log n^2 - 3n$
Find $big - O,\ big - \Omega,\ big - \Theta$ bounds

Big-O: $f(n) = 23n^3 \log n + 5n^2 \log n^2 - 3n$

- $23n^3 \log n \leq 23n^3 \cdot n \quad n > 0$

$5n^2 \log n^2 = 10n^2 \log n \leq 10n^3 \leq 10n^4 \quad n > 0$

$-3n \leq 0$

$\Rightarrow 23n^3 \log n + 5n^2 \log n^2 - 3n \leq 23n^4 + 10n^4 + 0$

$= 33n^4$

$\Rightarrow$ $f(n)$ is $O(n^4)$ with $C = 33,\ k = 1$

Asymptotic Analysis – Big-O, Big-Ω, Big-Θ, little-o, little-ω

Example: Given the function $f(n) = 23n^3 \log n + 5n^2 \log n^2 - 3n$
Find $big - O,\ big - \Omega,\ big - \Theta$ bounds

Find big-O, big-Ω of the form $O(n^p)$

Big-Ω

$23n^3 \log n \geq 23n^3 \cdot$ for $n > 1$

$5n^2 \log n^2 \geq 5n^2 \geq 0$

$-3n \geq -3n^3$ for $n > 1$ $\quad 3n < 3n^3$

$23n^3 \log n + 5n^2 \log n^2 - 3n \geq 23n^3 + 0 - 3n^3 = 20n^3$

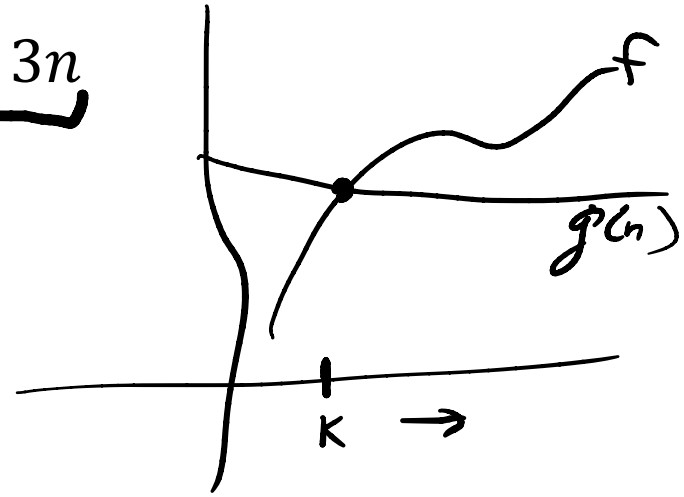
$f(n)$ is $\Omega(n^3)$ with $c = 20, k = 1$

Asymptotic Analysis – Big-O, Big-Ω, Big-Θ, little-o, little-ω

Example: Given the function $f(n) = 23n^3 \log n + 5n^2 \log n^2 - 3n$

Find $big - O,\ big - \Omega,\ big - \Theta$ bounds

$5n^2 \log n^2 - 3n > 0$



$c_1(g(n)) \leq f(n) \leq c_2(g(n))$ •

Big-Θ: $23n^3 \log n \leq 23n^3 \log n$

$10n^2 \log n \leq 10n^3 \log n \quad n > 1$

$-3n \leq 0$

$\Rightarrow f(n)$ is $O(n^3 \log n)$

$C_2 = 33,\ k = 1$

Asymptotic Analysis – Big-O, Big-Ω, Big-Θ, little-o, little-ω

Example: Given the function $f(n) = 23n^3 \log n + 5n^2 \log n^2 - 3n$
Find $big - O,\ big - \Omega,\ big - \Theta$ bounds

$23n^3 \log n \geq 23n^3 \log n$

$5n^2 \log n^2 - 3n \geq 5n^2 \log n^2 - n^2 \log n^2$

$= 4n^2 \log n^2$

$\Rightarrow 23n^3 \log n + 5n^2 \log n^2 - 3n \geq 23n^3 \log n + 4n^2 \log n^2$

$\geq 23n^3 \log n$

Why?

$5n^2 \log n^2 - 3n \overset{?}{\geq} 4n^2 \log n^2$

This is equivalent to

$-3n \geq -n^2 \log n^2$

$3n \leq n^2 \log n^2$

$3 \leq n \log n$

True for $n > 1.267$

(used wolfram alpha)

$\Rightarrow f(n)$ is $\Omega(n^3 \log n)$

$\Rightarrow$ Since $23n^3 \log n \leq f(n) \leq 33n^3 \log n$, $f(n)$ is $\Theta(n^3 \log n)$

Thus witnesses: $c_1 = 23,\ c_2 = 33,\ K = 1.267$

Asymptotic Analysis – Big–O, Big–Ω, Big–Θ, little–o, little–ω

Example: Given the function $f(n) = 23n^3 \log n + 5n^2 \log n^2 - 3n$
Find $big - O$, $big - \Omega$, $big - \Theta$ bounds

Asymptotic Analysis – Big-O, Big-Ω, Big-Θ, little-o, little-ω

Example: Show that $3x^2 + x + 1$ is $\Theta(3x^2)$ by directly finding the constants $k, C_1,$ and C_2.

$3x^2 + x + 1 \leq 3x^2 + 2x^2 + x^2$ for $x > 1$

$= 6x^2$

$= 2(3x^2)$

$3x^2 + x + 1 \geq 3x^2$ for $x > 0$

Thus, $3x^2 + x + 1$ is $\Theta(3x^2)$ with $k = 1$, $C_1 = 1$, and $C_2 = 2$

Divide and Conquer

Divide and Conquer Algorithm Technique:

1) Divide: Take an instance and divide it into smaller instances of the same problem.
2) Conquer: if smaller instance is trivial, then solve it directly; else: divide again
3) Combine: Take the solutions for the smaller instances and combine them to give the solution to the larger instance

What are some examples of divide and conquer algorithms?

Binary Search, Quicksort, Mergesort

Unrolling Method

Example: Given the algorithm below, write down the recurrence and solve it using the unrolling method.

```
Let A = [1 ...n], where A[ i ] = i
def foo(A, s, t):
  if(t >= s):
    mid_l = floor((s + t)/3)
    mid_r = floor(2(s + t)/3)
    add = 0
    for i = s to t:
      add += A[ i ]
    print(add)
    foo(A, s, mid_l)
    foo(A, mid_l + 1, mid_r)
    foo(A, mid_r + 1, t)
```

$\Theta(1)$ (mid_l, mid_r)

$\Theta(1)$ (add = 0)

$\Theta(n)$ (for i = s to t: add += A[i])

$\Theta(1)$ (print(add))

$T(n/3)$ (foo(A, s, mid_l))

$T(n/3)$ (foo(A, mid_l + 1, mid_r))

$T(n/3)$ (foo(A, mid_r + 1, t))

Recurrence Relation:

$T(n) = \Theta(1) + \Theta(n) + 3T(n/3)$

If $n = 1$, then $T(n) = c$

$$\Rightarrow T(n) = \begin{cases} c & \text{when } n = 1 \\ 3T(n/3) + \Theta(n) & \text{when } n > 1 \end{cases}$$

Unrolling Method

Example: Given the algorithm below, write down the recurrence and solve it using the unrolling method.

```
Let A = [1 ...n], where A[ i ] = i
def foo(A, s, t):
  if(t >= s):
    mid_l = floor((s + t)/3)
    mid_r = floor(2(s + t)/3)
    add = 0
    for i = s to t:
      add += A[ i ]
    print(add)
    foo(A, s, mid_l)
    foo(A, mid_l + 1, mid_r)
    foo(A, mid_r + 1, t)
```

unrolling: $T(n) = 3T(n/3) + cn$

$= 3[3T(n/3^2) + cn/3] + cn$ · Plug in def. of $T(n/3)$

$= 3^2 T(n/3^2) + 2cn$ · Distribute the 3

$= 3^2[3T(n/3^3) + cn/9] + 2cn$ · Plug in def. of $T(n/9)$

$= 3^3 T(n/3^3) + 3cn$ · Distribute 3^2 the

$\vdots$

$= 3^k T(n/3^k) + kcn$

The recursion bottoms out when $\lceil n/3^k \rceil = 1$

$\Rightarrow n/3^k < 1$

$\Rightarrow n < 3^k$

$\log_3 n < k$

Ignoring floor/ceiling

let $k = \log_3 n$

next slide →

Unrolling Method

Example: Given the algorithm below, write down the recurrence and solve it using the unrolling method.

```
Let A = [1 ...n], where A[ i ] = i
def foo(A, s, t):
  if(t >= s):
    mid_l = floor((s + t)/3)
    mid_r = floor(2(s + t)/3)
    add = 0
    for i = s to t:
      add += A[ i ]
    print(add)
    foo(A, s, mid_l)
    foo(A, mid_l + 1, mid_r)
    foo(A, mid_r + 1, t)
```

So $T(n) = 3^{\log_3 n} T(1) + \log_3 n \cdot cn$

$= cn + cn \log_3 n$

$\Rightarrow T(n)$ is $\Theta(n \log n)$.

Recursion Tree Method

Example: Determine the asymptotic solution to the recurrence

$T_1(n) = T\left(\frac{n}{100}\right) + T\left(\frac{99n}{100}\right) + cn$

versus the recurrence $T_2(n) = 2T\left(\frac{n}{2}\right) + cn$

- For $T_1(n) = T(n/100) + T(99n/100) + cn$

the longest path to the bottom of the recursion tree would be

$\left(\frac{99}{100}\right)^k \cdot n = 1$ · neglect floor/ceiling here.

$n = \left(\frac{100}{99}\right)^k$

$\Rightarrow k = \log_{100/99} n$ levels.

doesn't matter what base is... we can always apply a change of base formula

- The work done at each level is $\leq cn$

$\Rightarrow T_1(n)$ is $O(n \log n)$

Recursion Tree Method

Example: Determine the asymptotic solution to the recurrence

$T_1(n) = T\left(\frac{n}{100}\right) + T\left(\frac{99n}{100}\right) + cn$

versus the recurrence $T_2(n) = 2T\left(\frac{n}{2}\right) + cn$

$T_2(n) = 2T(n/2) + cn$

This is a binary tree.

$k = \log_2 n$

$\Rightarrow O(n \log n)$

T_1 and T_2 both have the same big-O asymptotic bound. The difference would be in the constant that bounds each recurrence.

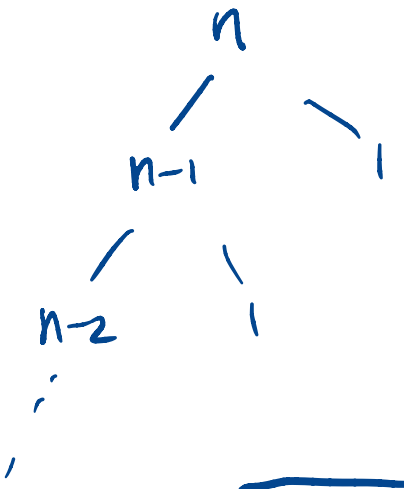
Recursion

Example: Write a recursive algorithm to compute the factorial of a non-negative integer n. Notice that the factorial $n! = 1 \times 2 \times \ldots \times (n-1) \times n$ when $n \geq 1$ and $0! = 1$.

What is the Θ behavior of your algorithm?

```
def factorial(n):
    if (n == 0):
        return 1
    else:
        Return n * factorial(n-1)
```

let's look at the Recursion tree:



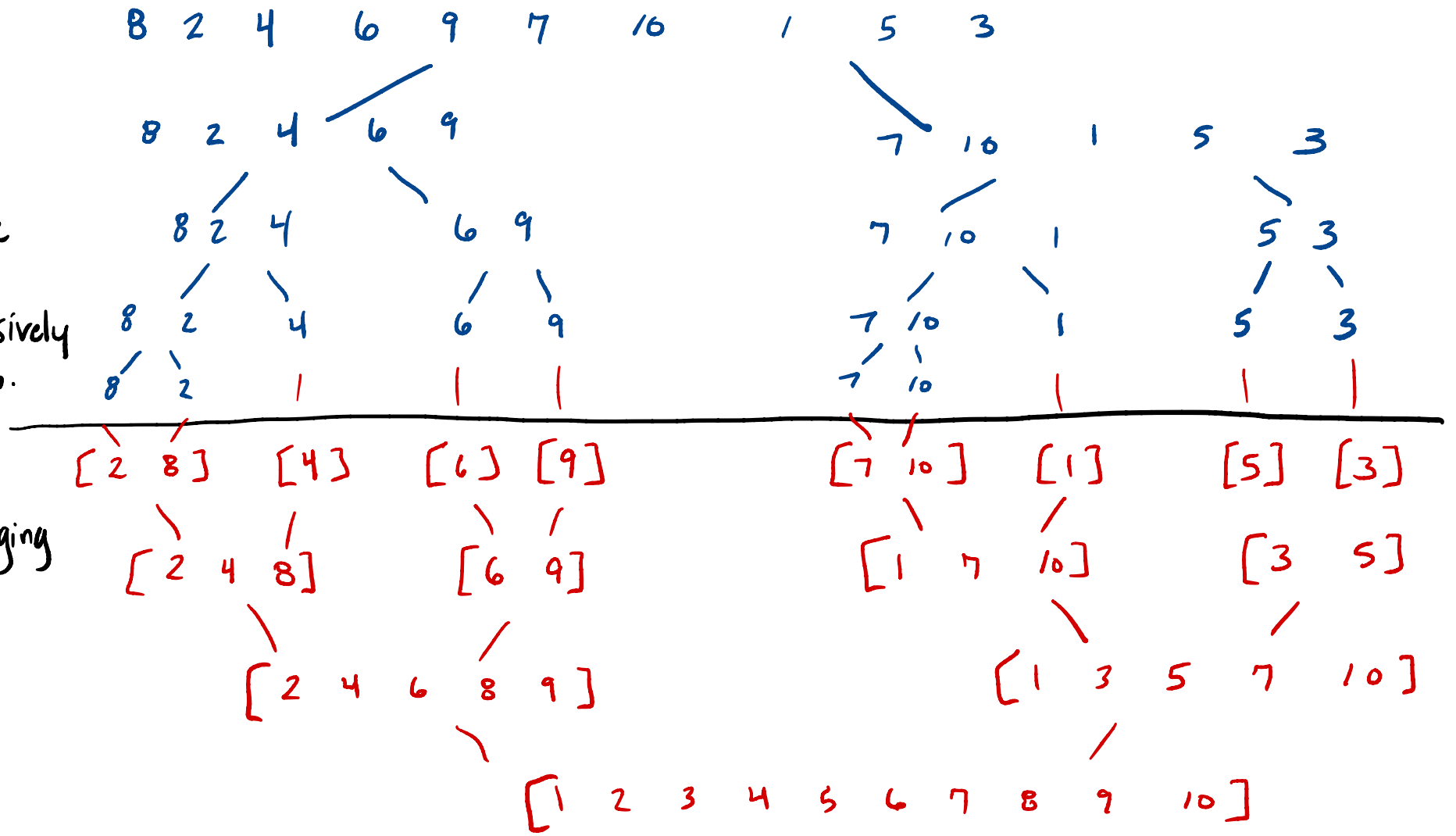
- n levels of the Recursion tree.
- work done at each level is some constant.

$\Rightarrow \Theta(n)$

Merge Sort

Example: Use the merge sort to put the terms of the list in increasing order:

8, 2, 4, 6, 9, 7, 10, 1, 5, 3

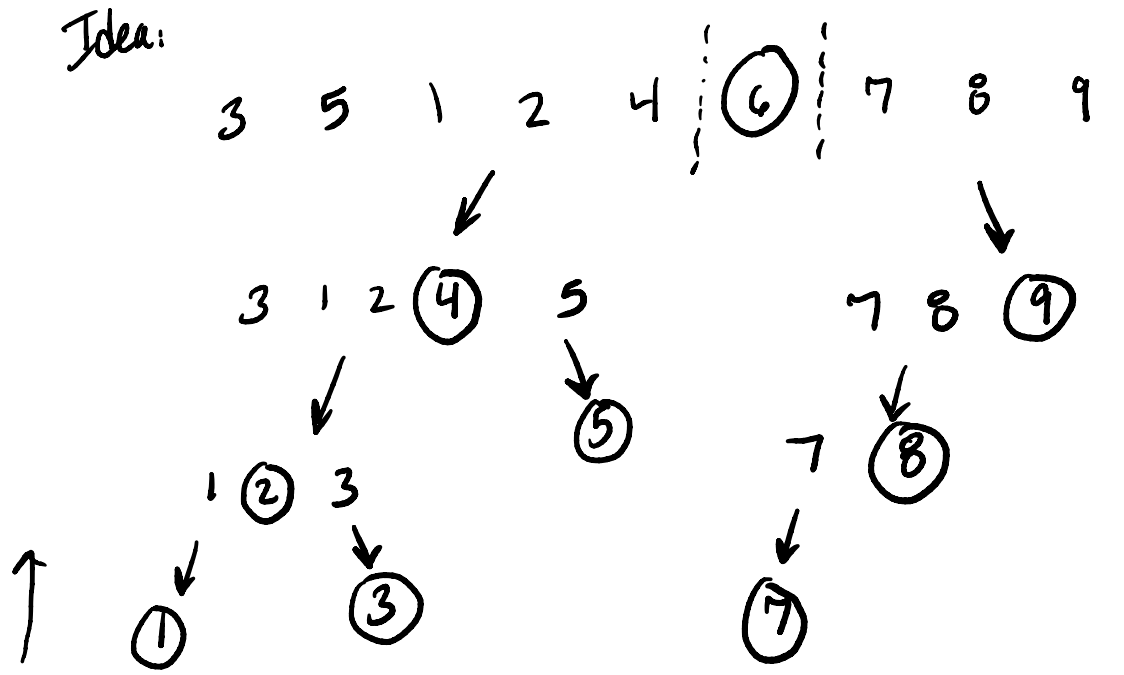


→ A merge sort begins by splitting the list into individual elements by successively splitting lists in two.

→ Sorting is done by successively merging pairs of lists.

Quicksort

Example: Sort 3, 5, 7, 8, 1, 9, 2, 4, 6 using the quick sort.



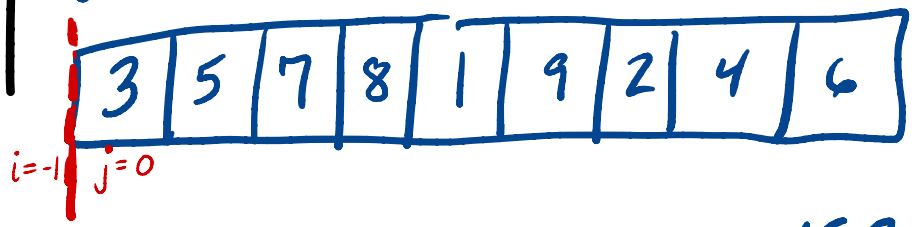
End Result: [1 2 3 4 5 6 7 8 9]

Q(A,0,8)

q = partition(A,0,8)

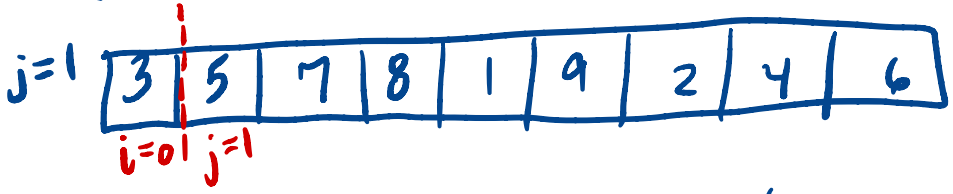
x = 6

i = 0-1 = -1



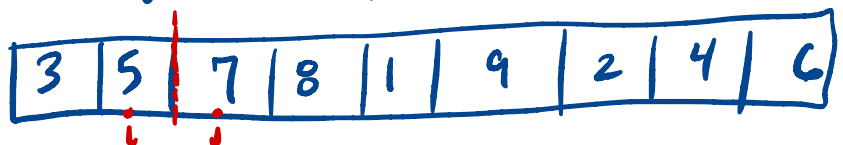
A[0] ≤ 6 yes. exchange A[0] with A[0]

i = -1+1 = 0



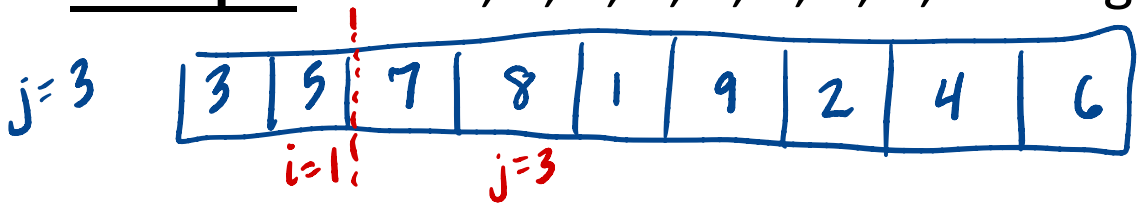
check: A[1] ≤ 6 yes. exchange A[1] with A[1]

i = 0+1 = 1

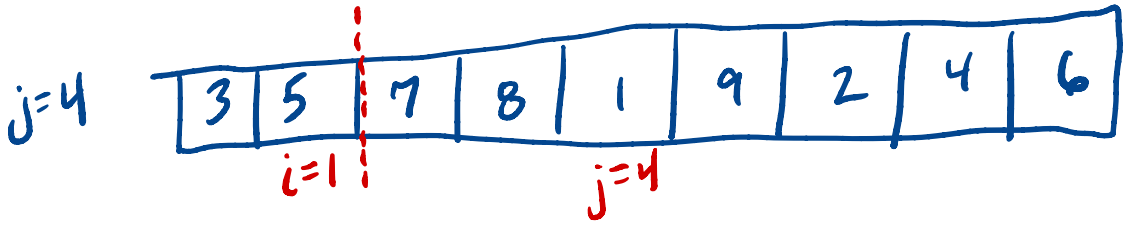


Quicksort

Example: Sort 3, 5, 7, 8, 1, 9, 2, 4, 6 using the quick sort.

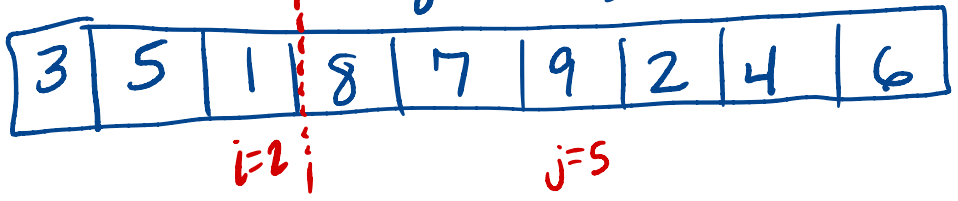


check: $A[3] \leq 6$ no.



check: $A[4] \leq 6$ yes

exchange $A[4]$ with $A[i+1]$, $i=2$



... etc. - eventually, 3 5 1 2 4 (6) 8 7 9

And $q = 5$ would be returned by PARTITION

*Assuming 0-indexing here.

Greedy Algorithms – Interval Scheduling

Example: In the interval scheduling problem, what of the following greedy choices will lead to an optimal solution:

(a) pick the job with the earliest start time among the remaining compatible jobs
(b) pick the job with the earliest finish time among the remaining compatible jobs
(c) pick the job with shortest interval among the remaining compatible jobs?

this maximizes the time left

Greedy Algorithms – Huffman Codes

Example: Given an alphabet of six symbols: $a, b, c, d, e,$ and f, with frequencies $0.4, 0.1, 0.2, 0.15, 0.11, 0.04$ respectively, what are the Huffman codes for the symbols b and c?

a	b	c	d	e	f
0.4	0.1	0.2	0.15	0.11	0.04

order: 0.04 (f), 0.1 (b), 0.11 (e), 0.15 (d), 0.2 (c), 0.4 (a)

append: .14 (0: f, 1: b)

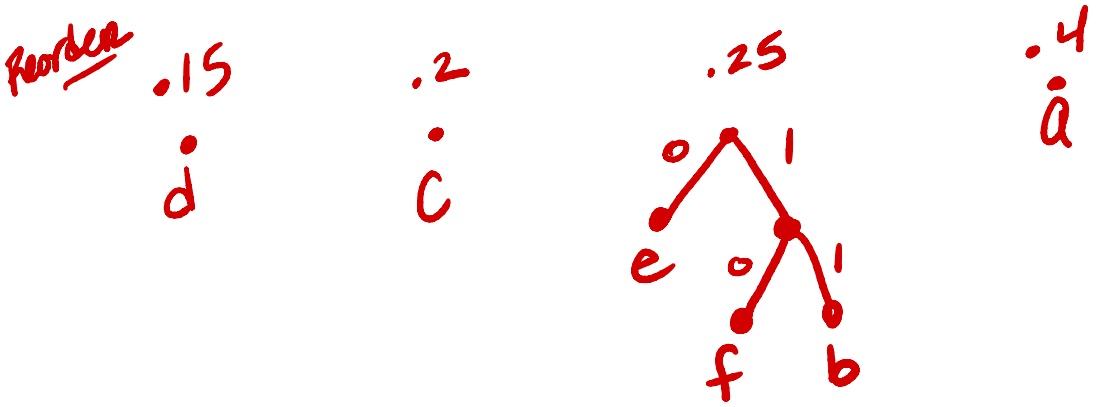
Reorder: .11 (e), .14 (0: f, 1: b), .15 (d), .2 (c), .4 (a)

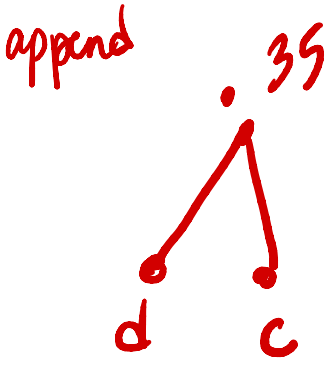
append: .25 (0: e, 1: .14 (0: f, 1: b))

→

Greedy Algorithms – Huffman Codes

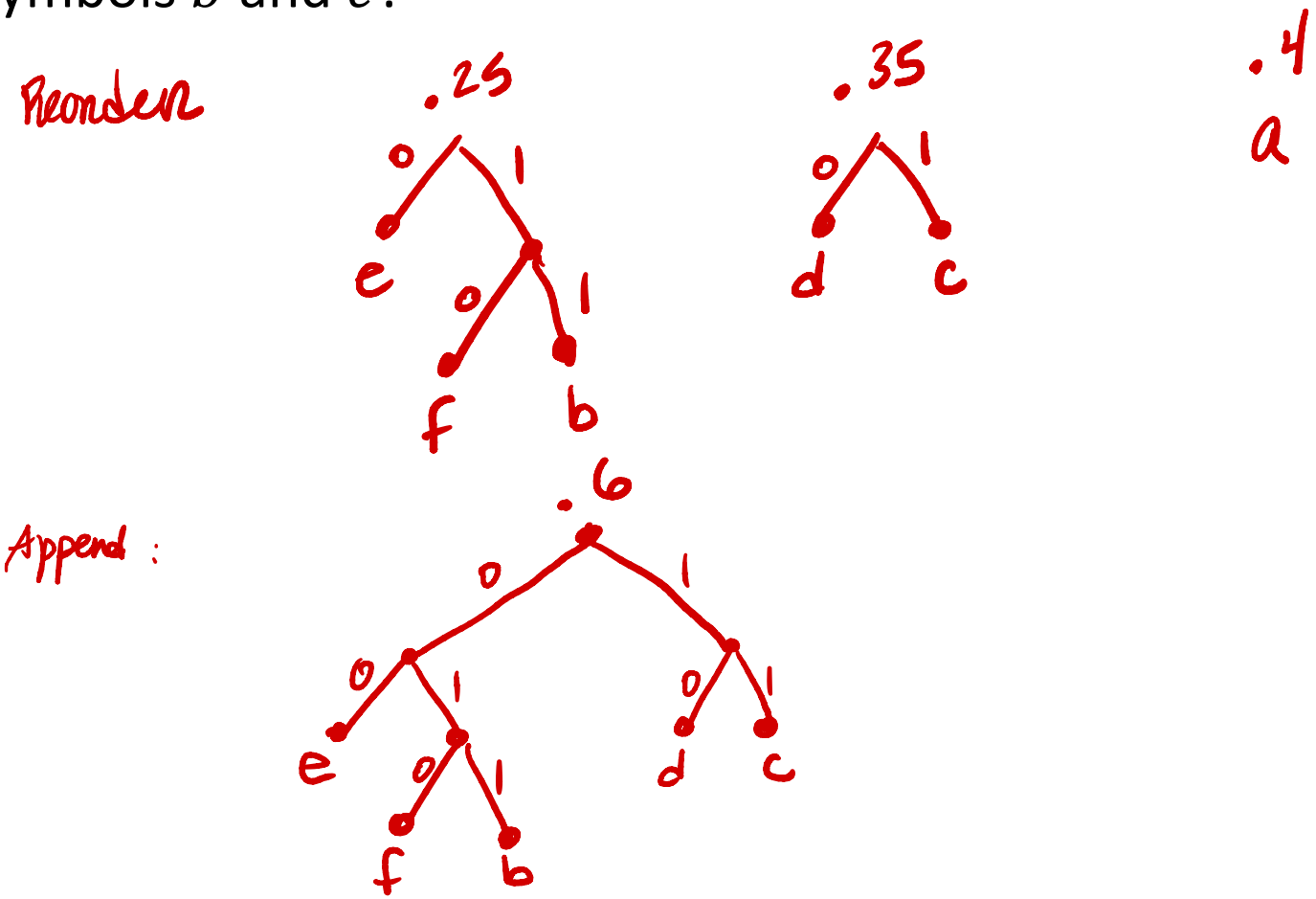
Example: Given an alphabet of six symbols: $a, b, c, d, e,$ and f, with frequencies $0.4, 0.1, 0.2, 0.15, 0.11, 0.04$ respectively, what are the Huffman codes for the symbols b and c?





Greedy Algorithms – Huffman Codes

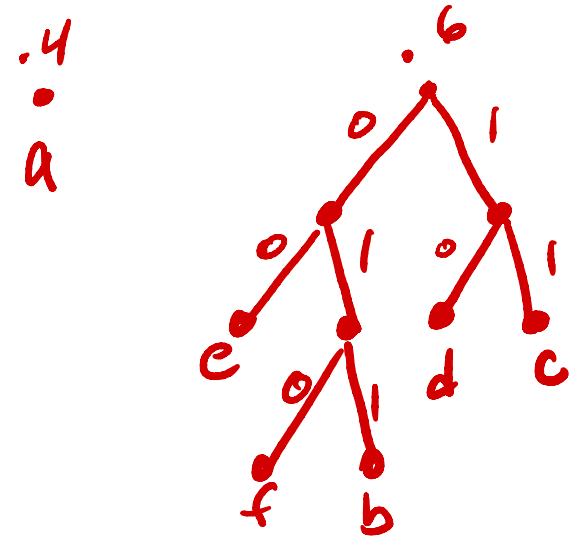
Example: Given an alphabet of six symbols: $a, b, c, d, e,$ and f, with frequencies $0.4, 0.1, 0.2, 0.15, 0.11, 0.04$ respectively, what are the Huffman codes for the symbols b and c?



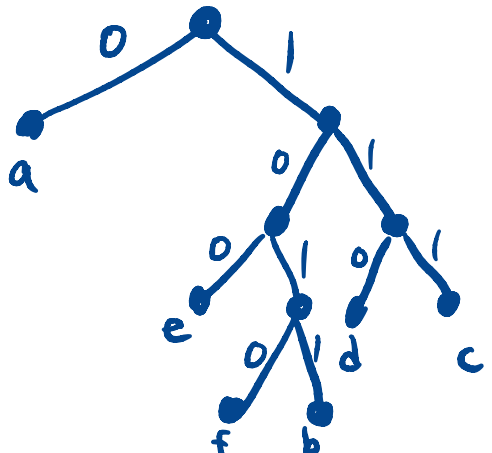
Greedy Algorithms – Huffman Codes

Example: Given an alphabet of six symbols: $a, b, c, d, e,$ and f, with frequencies $0.4, 0.1, 0.2, 0.15, 0.11, 0.04$ respectively, what are the Huffman codes for the symbols b and c?

Reorder



Append:



code for b: 1 0 1 1

C: 1 1 1