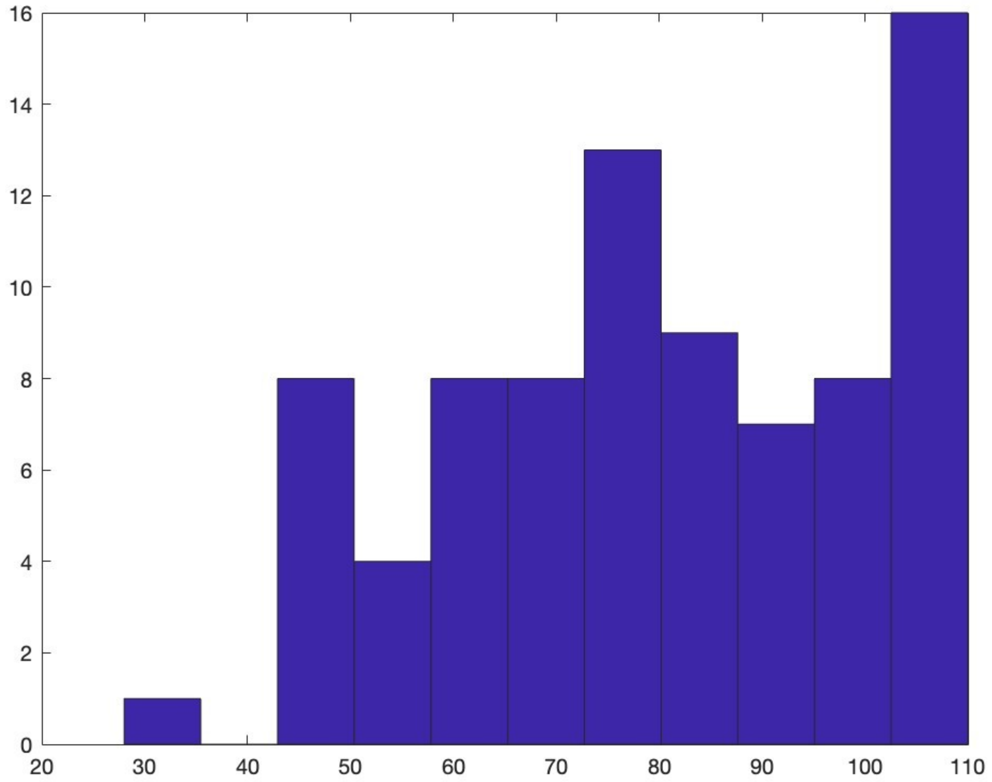
Analysis of Algorithms
CS 477/677

Instructor: Monica Nicolescu

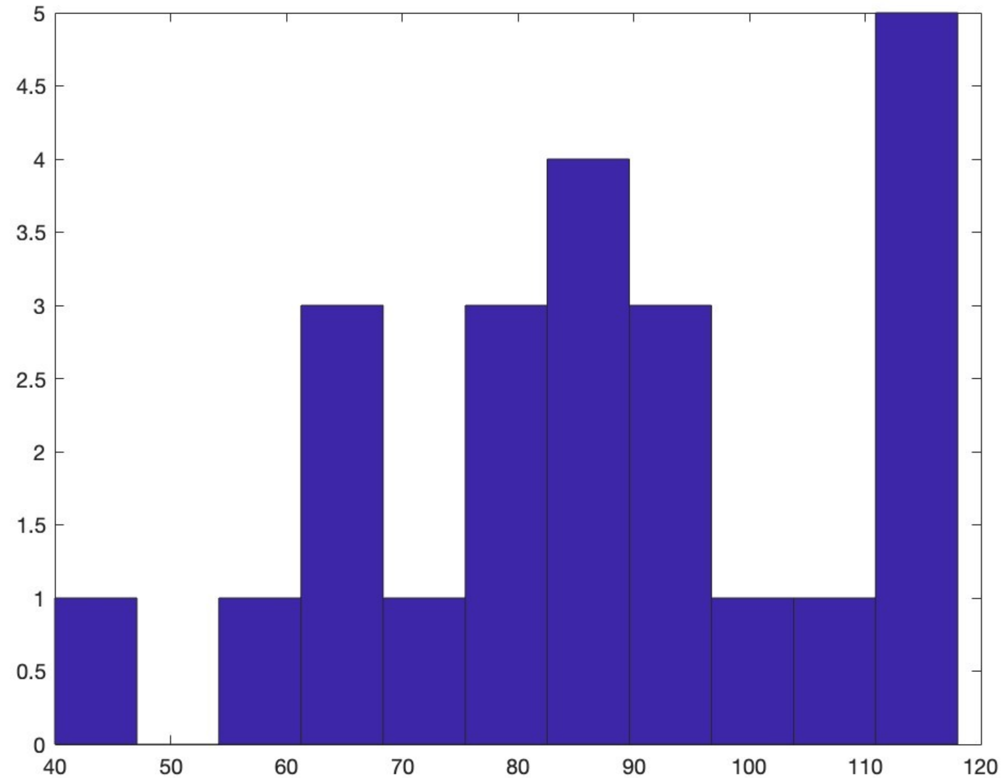
Lecture 21

Midterm 2 Results – CS 477



min = 28 max = 110 average = 79.81

Midterm 2 Results – CS 677



min = 40 max = 118 average = 87.56

Midterm 2 - Feedback

- Heap elements do not have pointers to children
- In RBTs nodes are inserted as red
- OS_Select finds the *i*-th order statistic not element with *key* = *i*
- When working with trees, always check if the tree root is NULL first, before accessing t T.left or T.right
- Cannot add pointers
- The height of a node is the maximum of the left/right subtree heights + 1, not their sum
- Recursive algorithms:
  - Pay attention to the return value for the base case
  - Make sure the return value is always the same type
  - Use the return values

Huffman Codes

- Idea:
  - Use the frequencies of occurrence of characters to build a optimal way of representing each character

	a	b	c	d	e	f
Frequency (thousands)	45	13	12	16	9	5

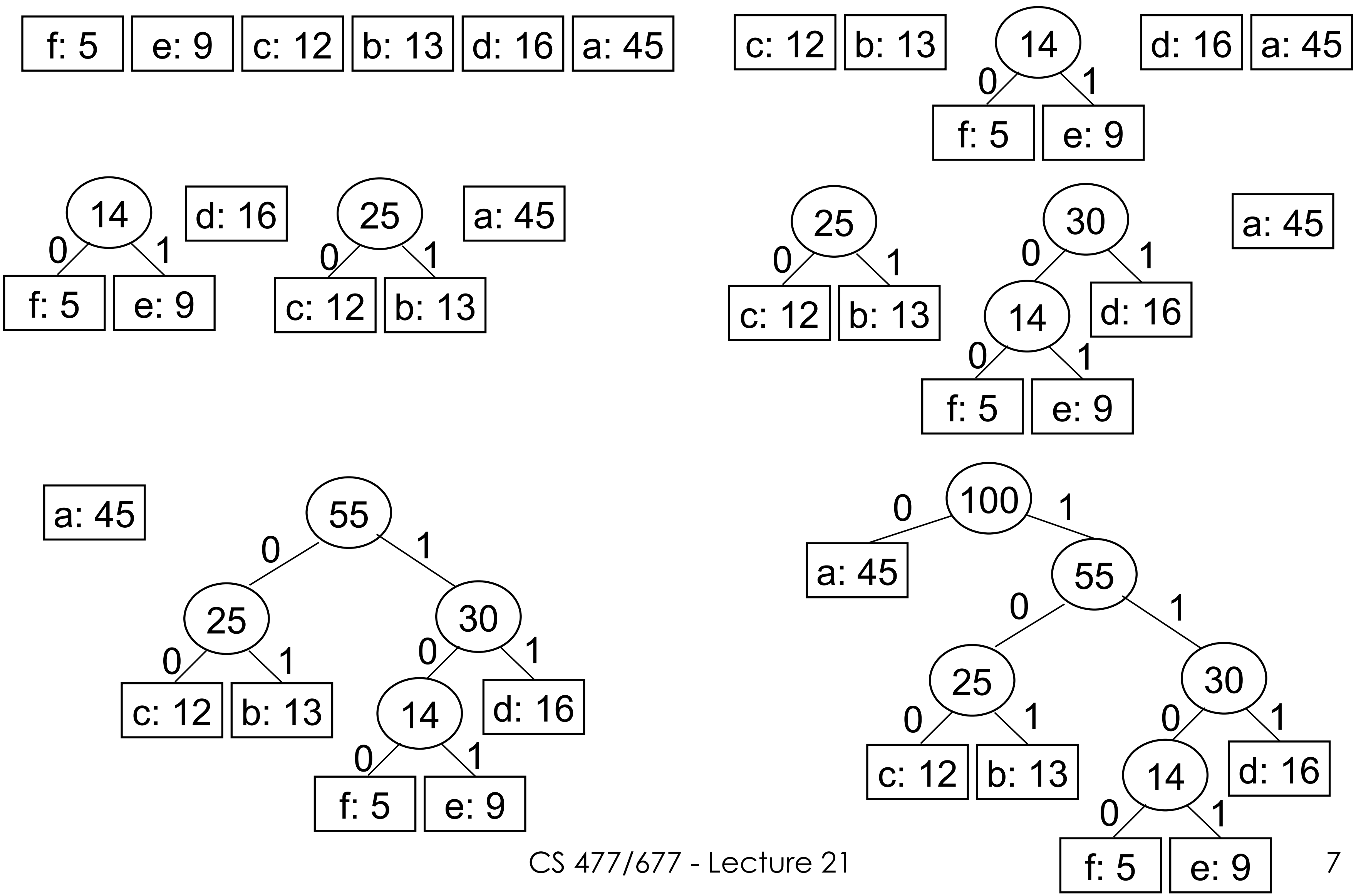
Constructing a Huffman Code

- Let's build a greedy algorithm that constructs an optimal prefix code (called a **Huffman code**)
- Assume that:
  - C is a set of n characters
  - Each character has a frequency $f(c)$
- Idea:

f: 5	e: 9	c: 12	b: 13	d: 16	a: 45

  - The tree T is built in a bottom up manner
  - Start with a set of $|C| = n$ leaves
  - At each step, merge the two least frequent objects: the frequency of the new node = sum of two frequencies
  - Use a min-priority queue Q, keyed on f to identify the two least frequent objects

Example



f: 5 e: 9 c: 12 b: 13 d: 16 a: 45
c: 12 b: 13 14 d: 16 a: 45 0 1 f: 5 e: 9
14 d: 16 25 a: 45 0 1 0 1 f: 5 e: 9 c: 12 b: 13
25 30 a: 45 0 1 0 1 c: 12 b: 13 14 d: 16 0 1 f: 5 e: 9
a: 45 55 0 1 25 30 0 1 0 1 c: 12 b: 13 14 d: 16 0 1 f: 5 e: 9
100 0 1 a: 45 55 0 1 25 30 0 1 0 1 c: 12 b: 13 14 d: 16 0 1 f: 5 e: 9

Building a Huffman Code

Alg.: HUFFMAN(C) Running time: **O(nlgn)**

1. $n \leftarrow |C|$
2. $Q \leftarrow C$ ← O(n)
3. **for** $i \leftarrow 1$ **to** $n - 1$
4. **do** allocate a new node **z**
5. left[z] ← x ← EXTRACT-MIN(Q)
6. right[z] ← y ← EXTRACT-MIN(Q)
7. f[z] ← f[x] + f[y]
8. INSERT (Q, z)
9. **return** EXTRACT-MIN(Q)

O(nlgn)

Greedy Choice Property

Let C be an alphabet in which each character $c \in C$ has frequency $f[c]$. Let x and y be two characters in C having the lowest frequencies.

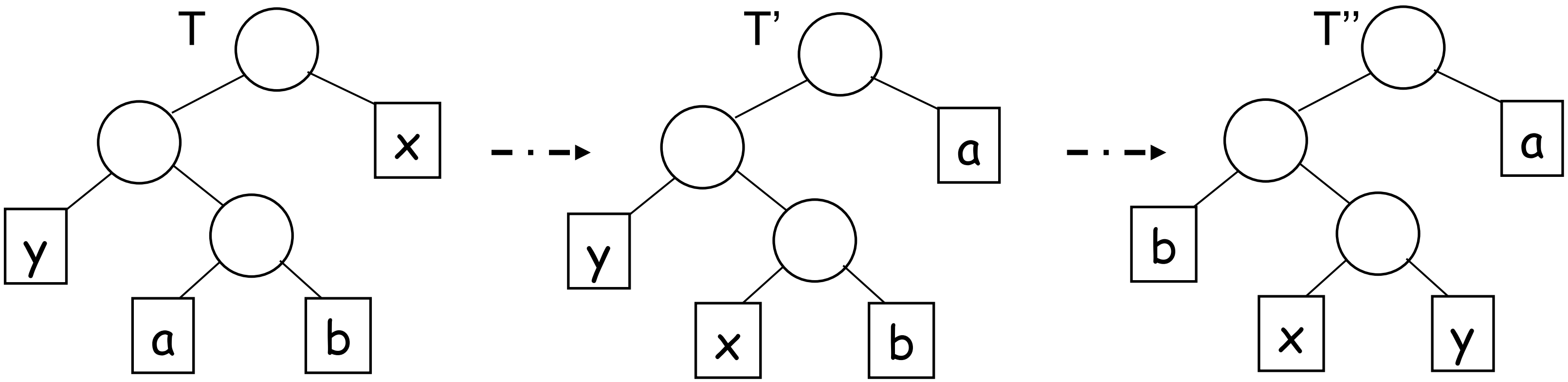
Then, there exists an optimal prefix code for C in which the codewords for x and y have the same (maximum) length and differ only in the last bit.

Proof of the Greedy Choice

- Idea:
  - Consider a tree **T** representing an arbitrary optimal prefix code
  - Modify **T** to make a tree representing another optimal prefix code in which **x** and **y** will appear as sibling leaves of maximum depth

⇒ The codes of **x** and **y** will have the same length and differ only in the last bit

Proof of the Greedy Choice (cont.)

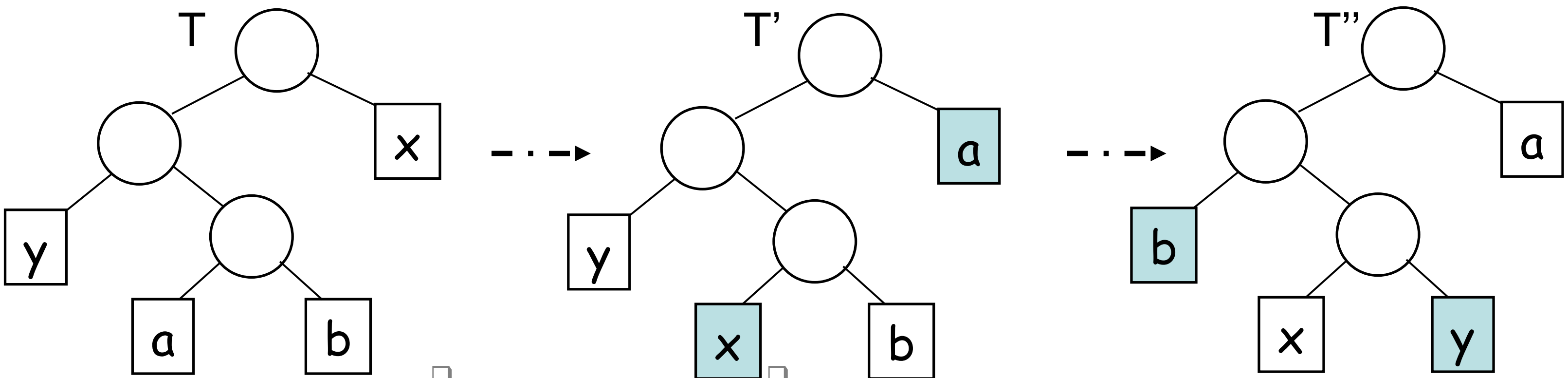


- **a**, **b** – two characters, sibling leaves of max. depth in T
- Assume: f[a] ≤ f[b] and f[x] ≤ f[y]
- f[x] and f[y] are the two lowest leaf frequencies, in order

  ⇒ f[x] ≤ f[a] and f[y] ≤ f[b]

- Exchange the positions of **a** and **x** (T') and of **b** and **y** (T'')

Proof of the Greedy Choice (cont.)



$$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$$

$$= f[x]d_T(x) + f[a]d_T(a) - f[x]d_{T'}(x) - f[a]d_{T'}(a)$$

$$= f[x]d_T(x) + f[a]d_T(a) - f[x]d_T(a) - f[a]d_T(x)$$

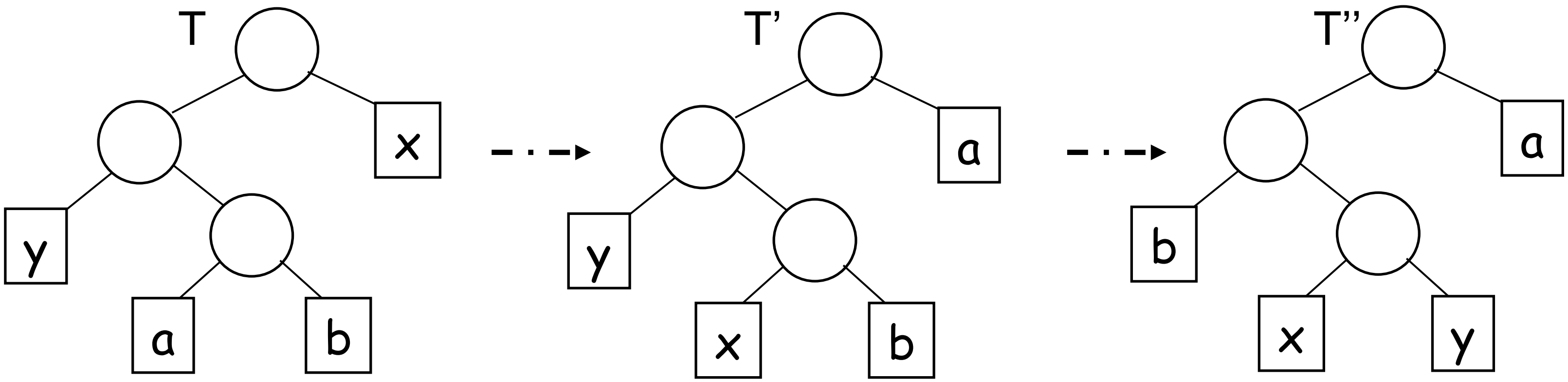
$$= \underbrace{(f[a] - f[x])}_{\geq 0} \underbrace{(d_T(a) - d_T(x))}_{\geq 0}$$

x is a minimum frequency leaf

a is a leaf of maximum depth

≥ 0

Proof of the Greedy Choice (cont.)



B(T) – B(T') ≥ 0

Similarly, exchanging **y** and **b** does not increase the cost:

B(T') – B(T'') ≥ 0

⇒ B(T'') ≤ B(T). Also, since T is optimal, B(T) ≤ B(T'')

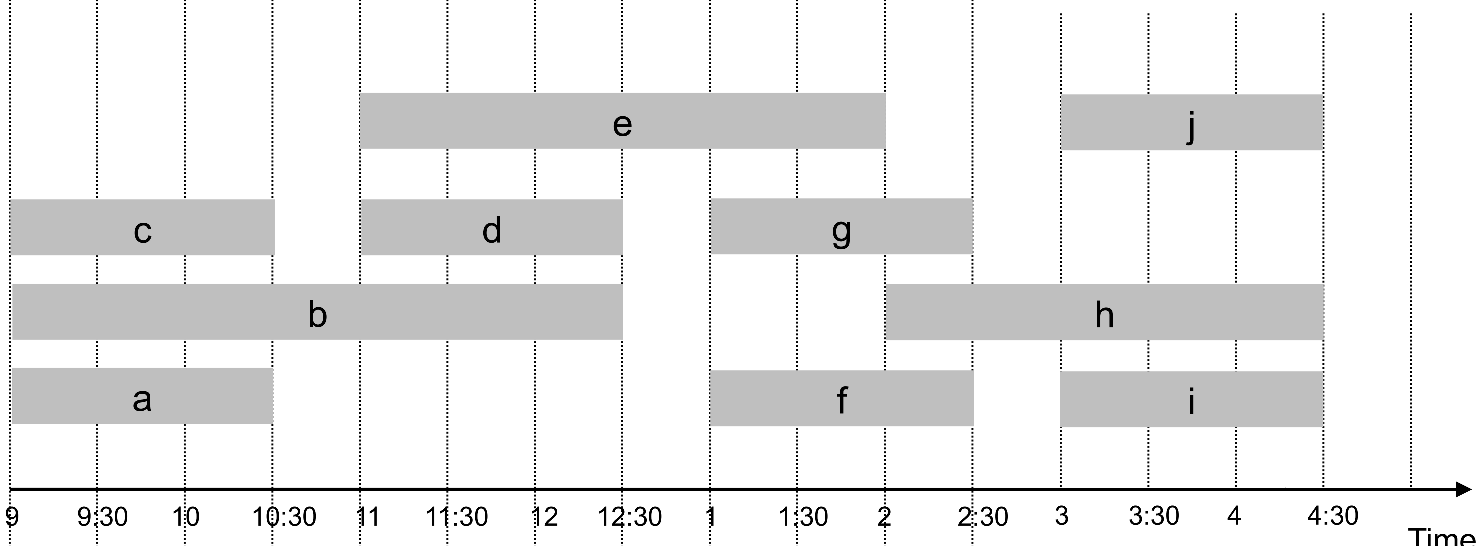
Therefore, B(T) = B(T'') ⇒ T'' is an optimal tree, in which x and y are sibling leaves of maximum depth

Discussion

- Greedy choice property:
  - Building an optimal tree by mergers can begin with the greedy choice: merging the two characters with the lowest frequencies
  - The cost of each merger is the sum of frequencies of the two items being merged
  - Of all possible mergers, HUFFMAN chooses the one that incurs the least cost

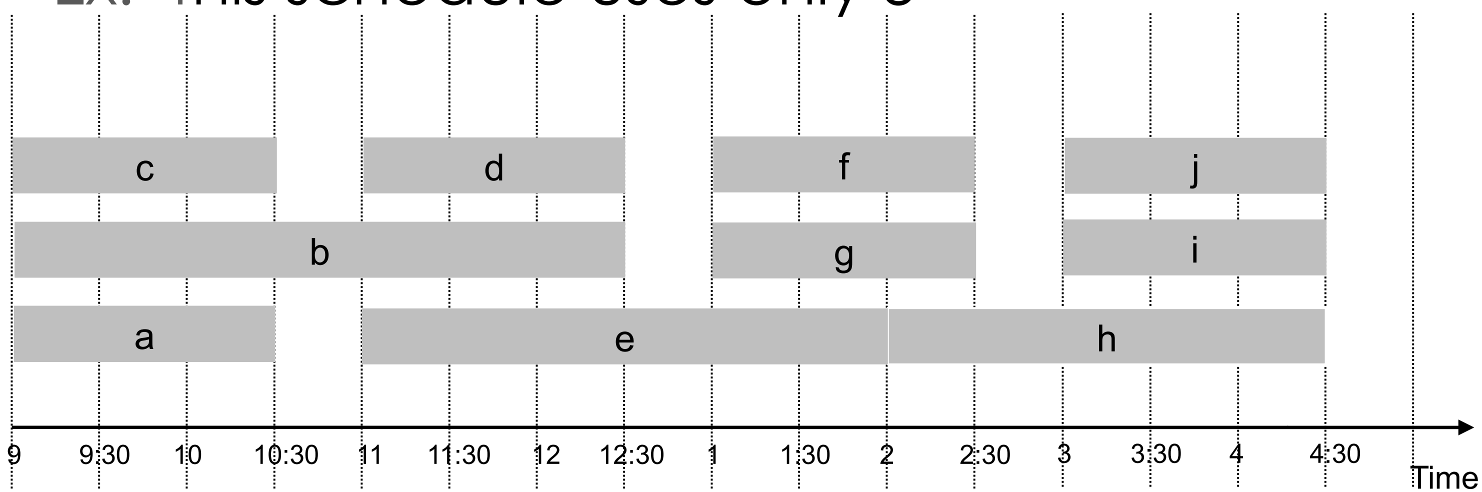
Interval Partitioning

- Lecture j starts at s_j and finishes at f_j
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room
  - Ex: this schedule uses 4 classrooms to schedule 10 lectures



Interval Partitioning

- Lecture j starts at s_j and finishes at f_j
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room
  - Ex: this schedule uses only 3

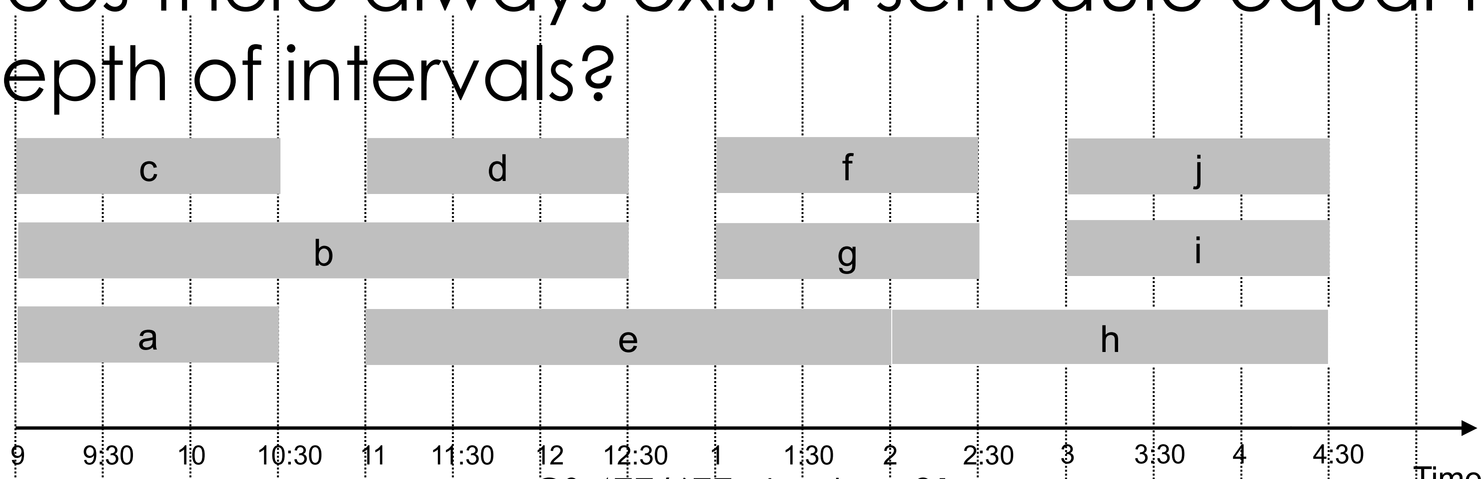


Interval Partitioning: Lower Bound on Optimal Solution

- The **depth** of a set of open intervals is the maximum number that contain any given time
- Key observation:
  - The number of classrooms needed ≥ depth
- Ex: Depth of schedule below = 3 ⇒ schedule below is optimal

a, b, c all contain 9:30

- Does there always exist a schedule equal to depth of intervals?



Greedy Strategy

- Consider lectures in increasing order of start time: assign lecture to any compatible classroom
  - Labels set {1, 2, 3, …, d}, where d is the depth of the set of intervals
  - Overlapping intervals are given different labels
  - Assign a label that has not been assigned to any previous interval that overlaps it

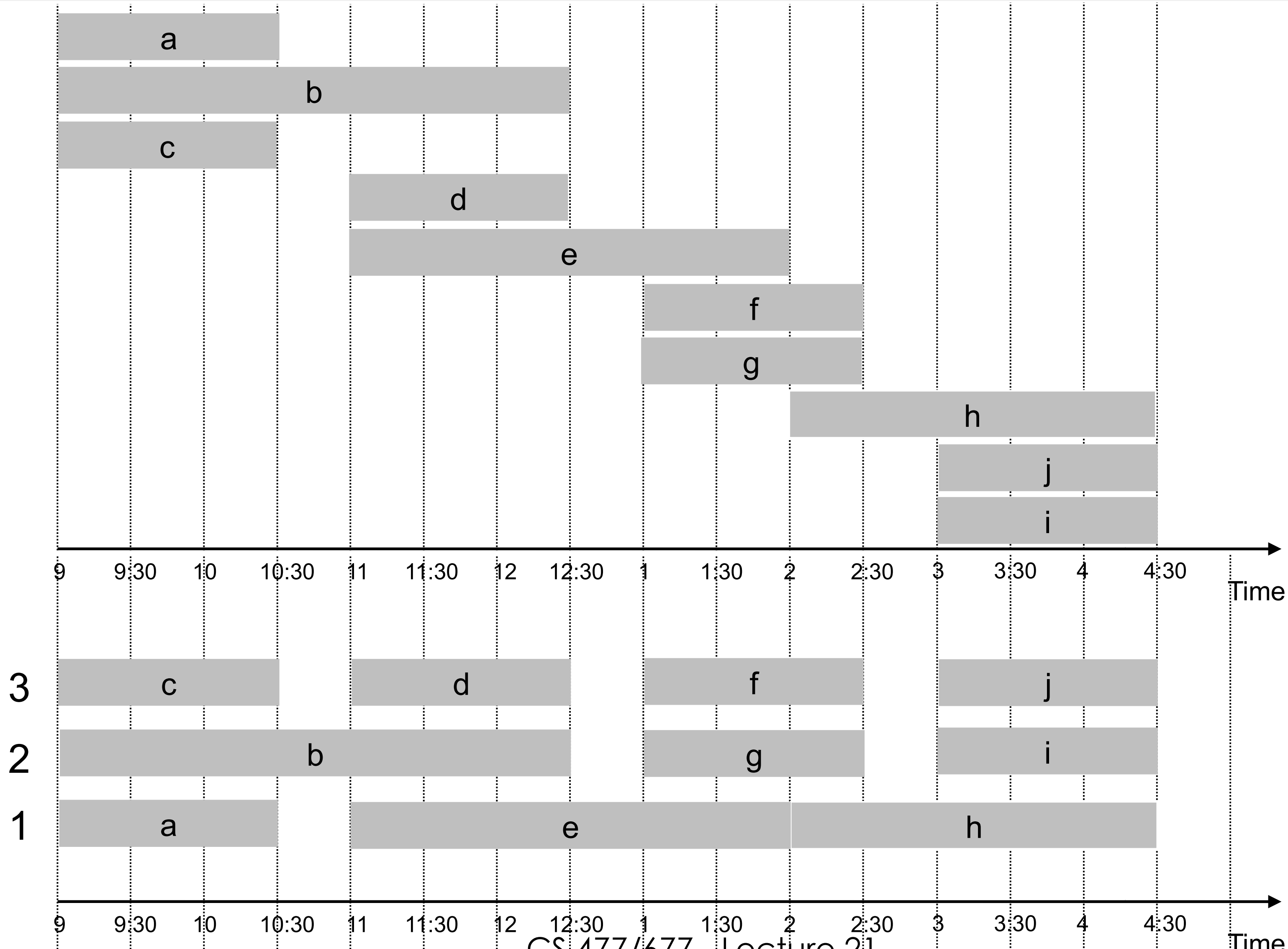
Greedy Algorithm

1. Sort intervals by start times, such that $s_1 \le s_2 \le \ldots \le s_n$

   (let $I_1, I_2, .., I_n$ denote the intervals in this order)

2. **for** $j = 1$ **to** n
3. Exclude from set $\{1, 2, \ldots, d\}$ the labels of preceding and overlapping intervals I_i from consideration for I_j
4. **if** there is any label from $\{1, 2, \ldots, d\}$ that was not excluded assign that label to I_j
5. **else**
6. leave I_j unlabeled

Example



a
b
c
d
e
f
g
h
j
i
9
9:30
10
10:30
11
11:30
12
12:30
1
1:30
2
2:30
3
3:30
4
4:30
Time
3
2
1
c
d
f
j
b
g
i
a
e
h
9
9:30
10
10:30
11
11:30
12
12:30
1
1:30
2
2:30
3
3:30
4
4:30
Time

Claim

- Every interval will be assigned a label
  - For interval I_j, assume there are t intervals earlier in the sorted order that overlap it
  - We have $t + 1$ intervals that pass over a common point on the timeline
  - $t + 1 \le d$, thus $t \le d - 1$
  - At least one of the d labels is not excluded by this set of t intervals, which we can assign to I_j

Claim

- No two overlapping intervals are assigned the same label
  - Consider I and I’ that overlap, and I precedes I’ in the sorted order
  - When I’ is considered, the label for I is excluded from consideration
  - Thus, the algorithm will assign a different label to I

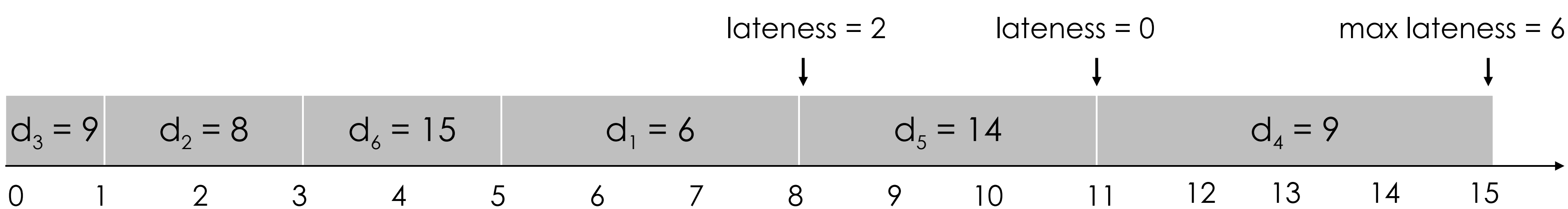
Greedy Choice Property

- The greedy algorithm schedules every interval on a resource, using a number of resources equal to the depth of the set of intervals. This is the optimal number of resources needed.
- Proof:
  - Follows from previous claims
- Structural proof
  - Discover a simple "structural" bound asserting that every possible solution must have a certain value
  - Then show that your algorithm always achieves this bound

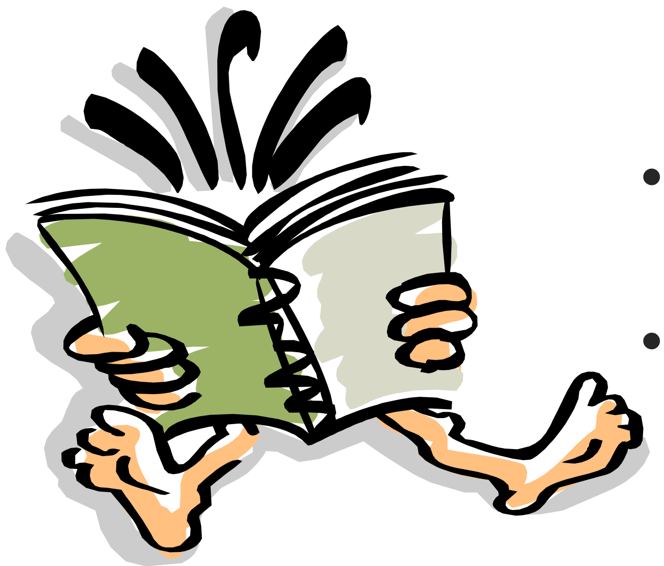
Scheduling to Minimizing Lateness

- Single resource processes one job at a time
- Job j requires t_j units of processing time, is due at time d_j
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$
- Lateness: $\ell_j = \max\{0, f_j - d_j\}$
- Goal: schedule all jobs to minimize **maximum** lateness $L = \max \ell_j$
- Example:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Readings



- For this lecture
  - Chapter 15
- Coming next
  - Chapter 15