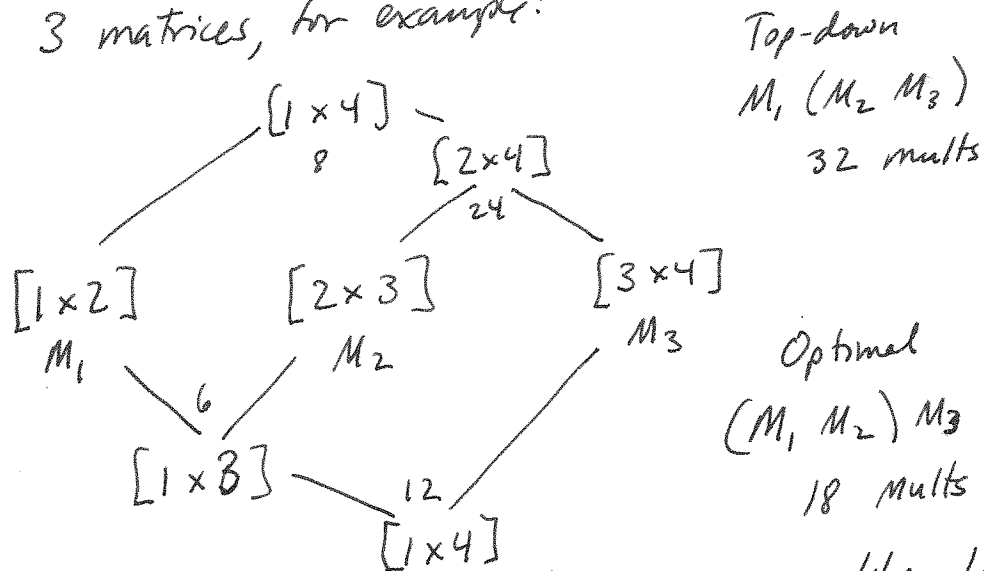


1A. Most divide-and-conquer algorithms split a problem into disjoint subproblems. Memoization is like dynamic programming - it exploits repeated subproblems. If divide-and-conquer subproblems are distinct, there will be no reuse of saved answers. ~~§~~  
 In the case of Freq, if one call is a descendent of another call in the call tree, the former will be on a strict subset of the latter, hence the calls are distinct. If the calls are on different paths, their sets are disjoint and hence distinct.

2B. A number of you showed that the top-down approach failed to get the optimal answer. There are examples with as few as 3 matrices, for example:



2C. This problem can be solved using the LCS algorithm. If  $S$  is the original input, let  $T = \text{sorted}(S)$  and  $U = TTR_1$ , where  $TR_1$  is the reversal of  $T$ , with the largest value removed.  
 Claim: ~~LCS~~  $LCS(S, U)$  will give you the longest bitonic subsequence of  $S$ . Note that  $U$  is bitonic, so that any subsequence of it is also bitonic.\*  
 Further, any bitonic subsequence of  $S$  will be a subsequence of  $U$ , including the longest bitonic subsequence of  $S$ .

Example:  $S = 3 \ 4 \ 6 \ 2 \ 7 \ 1 \ 15 \ 5$   
 $T = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 15$   
 $U = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 15 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1$   
 $LCS(S, U) = 3 \ 4 \ 6 \ 7 \ 15 \ 5$

[\*Note:  $U$  has subsequences such as  $1 \ 2 \ 2 \ 1$  that are not quite bitonic. However, since  $S$  has no duplicates,  $LCS$  will never find one of these.]

Time complexity: Sorting  $S$  is  $O(n \log n)$ . Creating  $U$  is  $O(n)$ .  $LCS(S, U)$  is  $O(\text{len}(S) \cdot \text{len}(U)) = O(n \cdot (2n-1)) = O(n^2)$ . Thus, the overall complexity is  $O(n^2)$ .

2D - see ~~next~~<sup>last</sup> page

2E. A greedy algorithm works here, where activities are scheduled by earliest start time, and we only add a room when all other rooms are already scheduled.

In order to get good time complexity, we can use a priority queue to organize both the activities and the finish of an activity in a room. So the queue will have two kinds of events:

SCHED( $a, s, e$ ): Activity  $a$  that runs from time  $s$  to  $e$  needs to be scheduled.

AVAIL( $r, e$ ): Room  $r$  will be available next at time  $e$ .

In the queue, items are ordered smallest-first by  $SCHED.s$  and  $AVAIL.e$ , with an  $AVAIL$  item coming before any  $SCHED$  item with the same sort key. (C)

We also will maintain a list  $OPEN$  of currently open rooms.

The schedule will be in the form of items of the form  $ASSIGN(a, r)$ , meaning activity  $a$  is assigned to room  $r$ .

Let  $A = A_1, A_2, \dots, A_n$  be the initial list of activities, and let  $PQ$  be the priority queue.

min Rooms(A)

Add  $A$  to  $PQ$ ;  $OPEN \leftarrow nil$

While  $PQ$  not empty

$m \leftarrow EXTRACT-MIN(PQ)$

if case  $m = SCHED(a, s, e)$

if  $OPEN = nil$  then  $add(OPEN, newRoom())$

$r \leftarrow remove(OPEN)$

output  $ASSIGN(a, r)$

$INSERT(AVAIL(r, e), PQ)$

case  $m = AVAIL(r, e)$

$add(OPEN, r)$

Time complexity:  $PQ$  initially has  $n$  events, and never gets larger, since any  $INSERT$  follows an ~~extract~~  $EXTRACT-MIN$ .

If we implement  $PQ$  as a heap, then each operation will be  $O(\lg n)$ , plus one  $O(n)$  for initialization.  $OPEN$  can be a stack, for  $O(1)$  cost per operation. Each activity is in  $PQ$  twice, as  $SCHED$  and  $AVAIL$ . So the

while -loop ~~executes~~ executes  $2n$  times, each time doing  $O(\lg n)$  work. So over all, we have  $O(n + 2n \cdot \lg n) = O(n \cdot \lg n)$ .

Example:  $A = [a_1, 1, 3], [a_2, 6, 8], [a_3, 5, 10], [a_4, 7, 11], [a_5, 4, 7], [a_6, 2, 5], [a_7, 4, 8], [a_8, 1, 2]$ .

Let  $S$  be SCHED and  $A$  be AVAIL is PQ

Start:  $PQ = S(a_1), S(a_8), S(a_6), S(a_5), S(a_7), S(a_3), S(a_2), S(a_4)$

1.  $m = S(a_1), OPEN = r1, \boxed{ASSIGN(a_1, r1)}$ , insert  $A(r1, 3)$   
 $PQ = S(a_8), S(a_6), A(r1, 3), S(a_5), S(a_7), S(a_3), S(a_2), S(a_4)$
2.  $m = S(a_8), OPEN = r2, \boxed{ASSIGN(a_8, r2)}$ , insert  $A(r2, 2)$ .  
 $PQ = A(r2, 2), S(a_6), A(r1, 3), S(a_5), S(a_7), S(a_3), S(a_2), S(a_4)$
3.  $m = A(r2, 2), OPEN = r2$   
 $PQ = S(a_6), A(r1, 3), S(a_5), S(a_7), S(a_3), S(a_2), S(a_4)$
4.  $m = S(a_6), OPEN = r2, \boxed{ASSIGN(a_6, r2)}$ , insert  $A(r2, 5)$   
 $PQ = A(r1, 3), S(a_5), S(a_7), A(r2, 5), S(a_3), S(a_2), S(a_4)$
5.  $m = A(r1, 3), OPEN = r1$   
 $PQ = S(a_5), S(a_7), A(r2, 5), S(a_3), S(a_2), S(a_4)$
6.  $m = S(a_5), OPEN = r1, \boxed{ASSIGN(a_5, r1)}$ , insert  $A(r1, 7)$   
 $PQ = S(a_7), A(r2, 5), S(a_3), S(a_2), A(r1, 7), S(a_4)$
7.  $m = S(a_7), OPEN = r3, \boxed{ASSIGN(a_7, r3)}$ , insert  $A(r3, 8)$   
 $PQ = A(r2, 5), S(a_3), S(a_2), A(r1, 7), S(a_4), A(r3, 8)$
8.  $m = A(r2, 5), OPEN = r2$
9.  $PQ = S(a_3), S(a_2), A(r1, 7), S(a_4), A(r3, 8)$

9.  $m = S(a_3)$ ,  $OPEN = r_2$ ,  $ASSIGN(a_3, r_2)$ , insert  $A(a_3, 10)$  |E|

$PQ = S(a_2), A(r_1, 7), S(a_4), A(a_2, 8), A(a_3, 10)$

10.  $m = S(a_2)$ ,  $OPEN = r_4$ ,  $ASSIGN(a_2, r_4)$ , insert  $A(a_2, 8)$

\*  $PQ = A(r_1, 7), S(a_4), A(a_2, 8), A(r_3, 8), A(r_2, 10)$

11.  $m = A(r_1, 7)$ ,  $OPEN = r_1$

$PQ = S(a_4), A(r_4, 8), A(r_3, 8), A(r_2, 10)$

12.  $m = S(a_4)$ ,  $OPEN = r_1$ ,  $ASSIGN(a_4, r_1)$ , insert  $(r_1, 11)$

$PQ = A(r_4, 8), A(r_3, 8), A(r_2, 10), A(r_1, 11)$

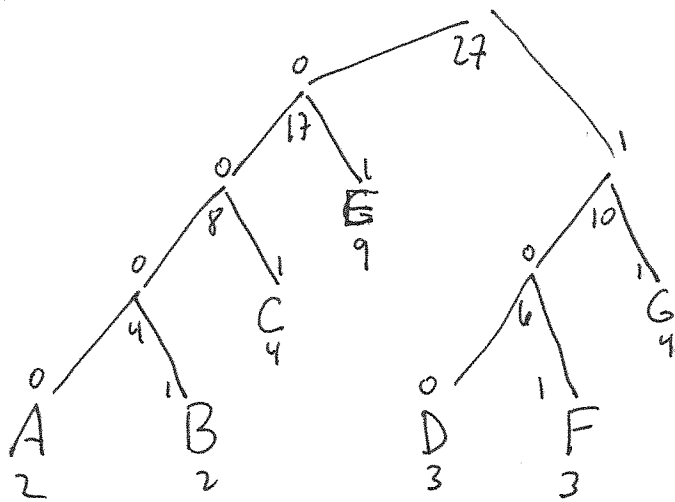
13.-16. We remove all the A's,  $OPEN = r_1, r_2, r_3, r_4$

Minimality: Note that if  $A(r_1, i)$  and  $A(r_2, j)$  are simultaneously in  $PQ$ , then they correspond to two activities  $A_i$  and  $A_j$  that overlap in time. Also note that we don't add room  $r_k$  unless rooms  $r_1 - r_{k-1}$  are in  $PQ$ . So whenever we add room  $r_k$ , we must have  $k$  mutually overlapping events, as at (\*). So no fewer than  $k$  rooms will suffice.

Correctness: We never assign an activity to an occupied room. If we are processing  $S(a_m)$ , then all rooms with a next available time after the start time of  $a_m$  are behind  $S(a_m)$  in  $PQ$ , and are not candidates for assignment.

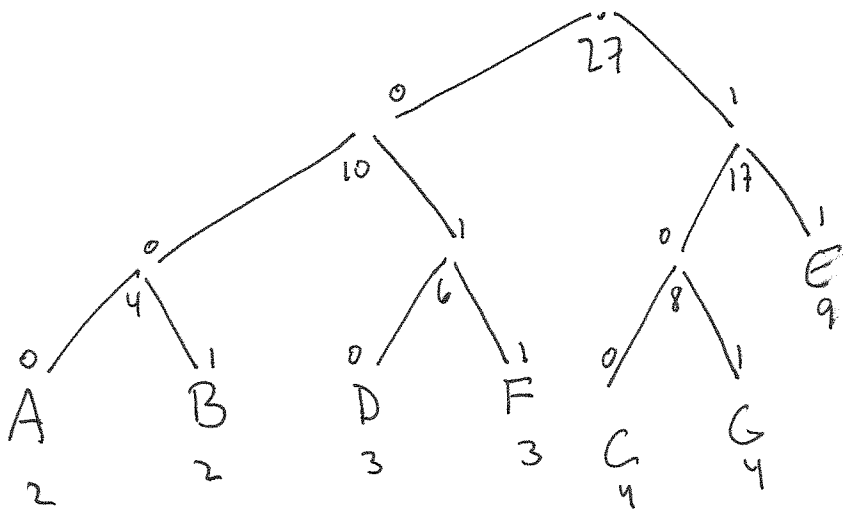
2F. There are choices in whether to merge DF with AB, C or G. (Plus there are left-right choices.)

F



code (ACEBAG) =  
0000, 001, 01, 0001, 0000, 11

19 bits \*



code (AEBAG) =

~~0000, 100, 11, 0001, 0000, 101~~

~~20 bits \*~~

000, 100, 11, 001, 000, 101

17 bits \*

\* Why does the difference in length ~~not~~ not represent a contradiction to the minimality of the second code?



### 584 Only

7. **Dynamic Programming** (25 points) We have  $n$  people with integer weights  $w_1, w_2, \dots, w_n$ . There are two elevators, and we want to know if we can divide the people into two groups that will put equal weights on each elevator. (For example, with weights 3, 4, 6, 2, 11, 4 we can split them into Group 1 = 4, 11 and Group 2 = 3, 6, 2, 4, each with total weight 15).

a. Describe a method to solve the problem in  $O(n^2W)$  time, where  $W$  is the maximum weight of any person. If you give pseudocode, also provide an English explanation.

Let  $S = \sum w_i$ . The goal is to see if some subset of the weights adds up to  $S/2$ . Then the remain weights must add up to  $S/2$  as well. (If  $S/2$  is not an integer, report "no solution")

Let  $B$  be a two dimensional array:  $B[0..n, 0..S/2]$   
 Entry  $B[i, v] = 1$  if some subset of  $w_1, \dots, w_i$  can add up to  $v$ .  
 Note that  $B[i, v] = 1$  if either  $B[i-1, v] = 1$  (leave out  $w_i$ ) or  $B[i-1, v-w_i] = 1$  (add  $w_i$  to the subset of  $w_1, \dots, w_{i-1}$  that sums to  $v-w_i$ ). Initialize  $B[0, v] = B[i, 0] = 0$ .  
 $B[i, 0] \leftarrow 1, B[0, v] \leftarrow 0$  for  $v > 0$

b. Illustrate your method with the weights 2, 3, 2, 7, 2. Here  $S=16$  so  $S/2=8$

$w_i$	$i$	0	1	2	3	4	5	6	7	8
	0	1	0	0	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0	0	0
3	2	1	0	1	1	0	1	0	0	0
2	3	1	0	1	1	1	1	0	1	0
7	4	1	0	1	1	1	1	0	1	0
2	5	1	0	1	1	1	1	1	1	0

This value is 0, so even split not possible.

There is also a formulation that just keeps a list of the 1's in each row.

for  $i = 1$  to  $n$   
 for  $v = 1$  to  $S/2$   
 $B[i, v] \leftarrow B[i-1, v]$  or  $B[i-1, v-w_i]$

Test if  $B[n, S/2] = 1$   
 since  $S/2 < n \cdot W$ ,  
 this is  $O(n \cdot S/2) \subseteq O(n \cdot n \cdot W)$