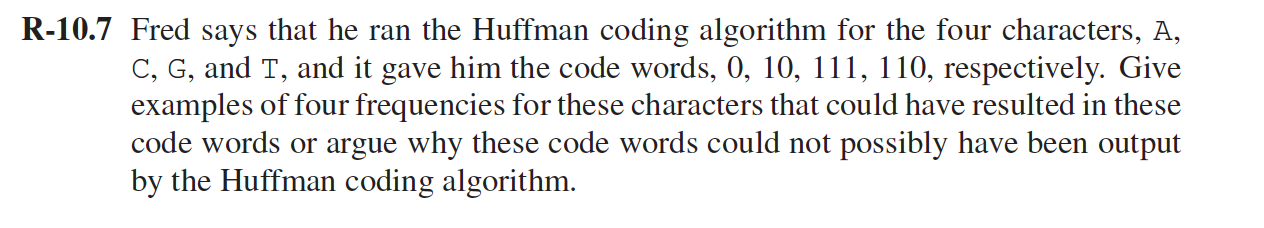
R-10.7,



Consider frequencies of A,C,G,T as follows:

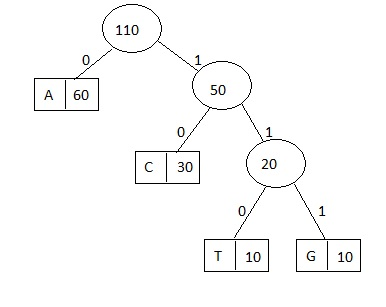
A: 60

C: 30

G: 10

T: 10

The hoff man tree will be:



So, the hoffman coding of the characters are:

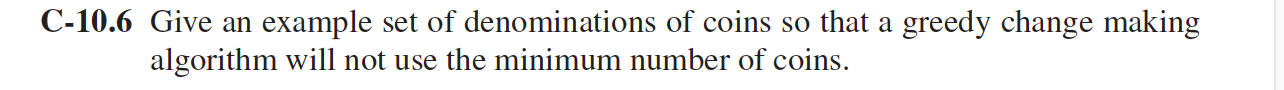
A:0

C:10

G:111

T:110

C-10.6,



In case if we are looking for an approach where we are not looking for minimum number of coins, we can follow the following approach:

We can randomly select denominations instead of going in an order from maximum denomination to the lowest denomination. Because if we follow the order we will be heading for minimum number of coins.

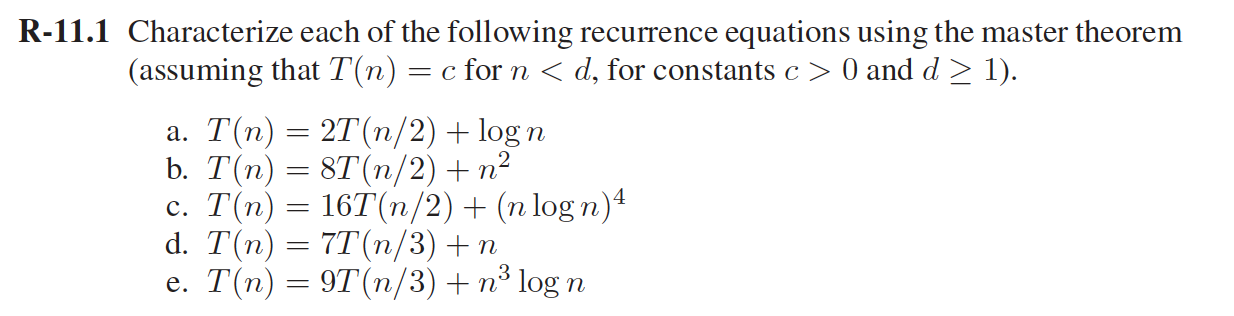
In case of random selection we may end up with minimum or maximum number of coins. So if we have denominations 2,5,7 and the amount is 16. So in case of random selection we may end up 8(2) , or 2 (5), 2(3) , etc.

A-10.7

A greedy solution will give us the solution in linear time.

Start with finding the first letter of the output transmission (hpr) in the original transmission(\*h\*arrypotter)  
Second step is to find the second letter starting at the index of the first letter plus one (arry\*p\*otter), ignoring the letters we have matched already.  
Continue the above two steps till all the letters in the output transmission have been matched, i.e. find the third letter starting at the index of the second letter plus one (otte\*r\*)  
Stop when we run out of letters (MATCH) or till the letters of output are not present in the original.  
Complexity: O(n) n=length of original transmission

R-11.1,



1. a=2,b=2, n^logba=n and f(n) =logn ,so f(n) is small compared to n^logba and

according to the first case of the master theorem,

T(n)=Theta(n^logba)=Theta(n).

1. a=8,b=2, n^logba=n3, and f(n)=n2, Thus case (1) applies since f(n) = n2<= n3-d with d= 1/2, for example.

Thus, T(n)=Theta(n^logba)=Theta(n3)

1. a=16,b=2, logba=4, and f(n)=n4(logn)4.Thus f(n)=n^ logba(logn)4 and case(2) applies with k=4, T(n)=Theta(n^logbalog1+pn)=Theta(n4log5n)
2. a=7,b=3, logba=1.777…. and f(n) =n . Thus f(n)=n <= n^log37-d for any 0< d<0.77. Hence , according to the first case of the master theorem

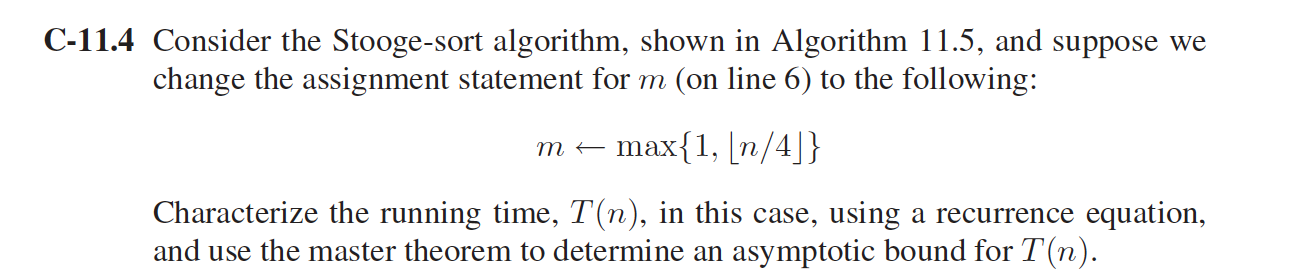
T(n)=Theta(n^logba)=Theta(n^log37)

1. a=9,b=3, logba=2 and f(n)=n3logn so neither case(1) or (2) applies.

Thus, we check if case (3) applies. Since f(n)=n3logn >= n2+d for d =1, for example, and since af(n/b)=9(n/3)3log(n/3) <= ef(n) for e=1/3 and n>=1, case (3) does apply.

Therefore,T(n)=Theta(n3logn)

C-11.4,



Running time using a recurrence equation:

T(n)=T(2n/3) + T(2n/3)+T(2n/3)

The first T(2n/3) is the time required to sort first 2n/3 elements,The Second T(2n/3) is time required to sort second 2n/3 elements. The third T(2n/3) is the time required to sort the first 2n/3 elements again. So that T(n) = 3T(2n/3)+cn

The master’s theorem:

T(n) = 3T(2n/3)+cn

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

So that a=3, b=3/2, and n^logba = n^(log3/log(3/2))~n3 . So f(n) is small compared to n^logba, case 1 of the master theorem applies and T(n) ∼O(n3)

A-11.6

We can find Skyline in Θ(nLogn) time using **Divide and Conquer**. The idea is similar to Merge Sort, divide the given set of buildings in two subsets. Recursively construct skyline for two halves and finally merge the two skylines.

Merge two Skylines is similar to merge of merge sort, start from first strips of two skylines, compare x coordinates. Pick the strip with smaller x coordinate and add it to result. The height of added strip is considered as maximum of current heights from skyline1 and skyline2.

For example：

Height of new Strip is always obtained by takin maximum of following

(a) Current height from skyline1, say 'h1'.

(b) Current height from skyline2, say 'h2'

h1 and h2 are initialized as 0. h1 is updated when a strip from SkyLine1 is added to result and h2 is updated when a strip from SkyLine2 is added.

Skyline1 = {(1, 11), (3, 13), (9, 0), (12, 7), (16, 0)}

Skyline2 = {(14, 3), (19, 18), (22, 3), (23, 13), (29, 0)}

Result = {}

h1 = 0, h2 = 0

1. Compare (1, 11) and (14, 3). Since first strip has smaller left x, add it to result and increment index for Skyline1.

h1 = 11, New Height = max(11, 0)

Result ={(1, 11)}

1. Compare (3, 13) and (14, 3). Since first strip has smaller left x, add it to result and increment index for Skyline1

h1 = 13, New Height = max(13, 0)

Result = {(1, 11), (3, 13)}

1. Similarly (9, 0) and (12, 7) are added.

h1 = 7, New Height = max(7, 0) = 7

Result = {(1, 11), (3, 13), (9, 0), (12, 7)}

1. Compare (16, 0) and (14, 3). Since second strip has smaller left x, it is added to result.

h2 = 3, New Height =max(7, 3) = 7

Result = {(1, 11), (3, 13), (9, 0), (12, 7), (14, 7)}

1. Compare (16, 0) and (19, 18). Since first strip has smaller left x,

it is added to result.

h1 = 0, New Height = max(0, 3) = 3

Result = {(1, 11), (3, 13), (9, 0), (12, 7), (14, 3), (16, 3)}

1. Since Skyline1 has no more items, all remaining items of Skyline2

are added

Result = {(1, 11), (3, 13), (9, 0), (12, 7), (14, 3), (16, 3),

(19, 18), (22, 3), (23, 13), (29, 0)}

1. the strip (16, 3) is redundant, we remove all redundant strips.

Result = {(1, 11), (3, 13), (9, 0), (12, 7), (14, 3), (19, 18),

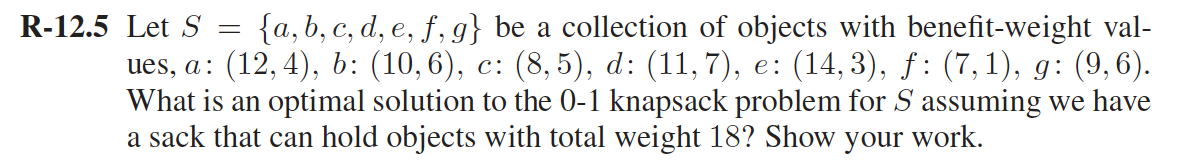
(22, 3), (23, 13), (29, 0)}

Time complexity of above recursive implementation is same as Merge Sort.

T(n) = T(n/2) + Θ(n)

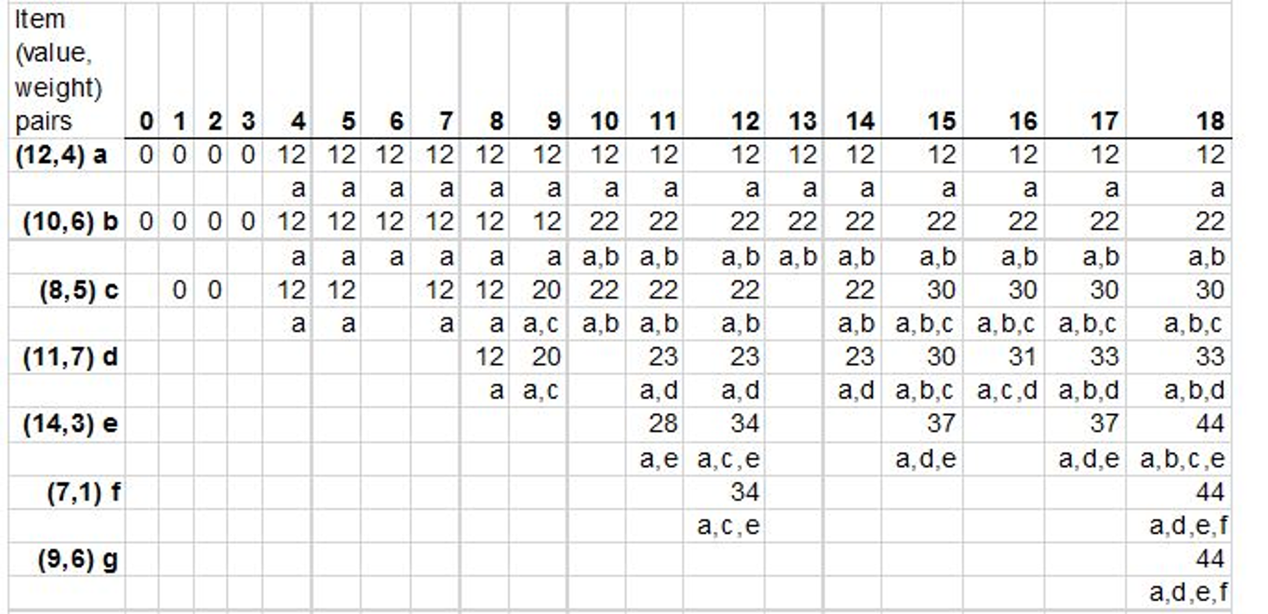
Solution of above recurrence is Θ(nLogn)

R-12.5,



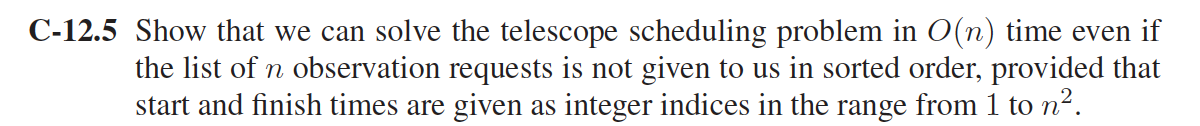
Answer：

The table fills only those values which are necessary to obtain the final answer to the problem, in (g,18).



The solution to maximum total weight 18, and items a,b,c,d,e,f,g, the best way to choose the items is to choose (a,d,e,f), and get a benefit of 44.

C-12.5,



Use an backtrack method, we simply need to trace backward in B from

last point. During the trace, if B[i] = B[i − 1], then we can assume observation

i is not included and move next to consider observation i − 1. Otherwise, if

B[i] = B[P[i]] + bi, then we can assume observation i is included and move next

to consider observation P[i]. It is easy to see that the running time of this algorithm is O(n).

A-12.3

The most efficient was of breaking the input into a sequence of English words would be using Dynamic programming. The method is described below:

* We maintain a dp array dp[ i ].
* At the beginning, initialize dp[0]=true.
* Let dp[ i ] = true when there is a j, such that j < i, dp[ j ] = true and valid(string S[ j : i ])==true.
* Break the input at all indices ***i*** where dp[ i ] = true.

There would be two loops involved, one of ***i*** and an inner loop of ***j***. The total running time would be O(n^2) because we use O(1) about n^2 times to check if dp[ i ] is true.