

Solution:

1

0

A

1

0

C

0

1

G

T

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| word | A | C | T | G |
| Frequency 1 | 100 | 45 | 10 | 20 |
| Frequency 2 | 40 | 30 | 10 | 10 |
| Frequency 3 | 130 | 70 | 40 | 30 |
| Frequency 4 | 15 | 7 | 2 | 3 |

Example for frequency 1

1

0

A

1

0

C

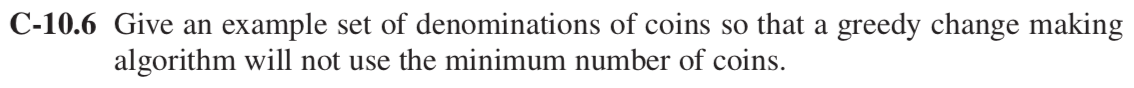
0

1

G

T

Here code of characters A, C, T, G have different length and are assigned in the fashion that most frequently used characters to use fewest number of bits and least used characters to use the most. It satisfies the prefix code, that no code word is a prefix of any other code. As a frequency to appear in a word is more so it has given smallest bit i.e. 0. And least use word G is given the largest bit i.e. 111. This technique helps in achieving the maximum compression.



Solution:

Consider the example:

Suppose value of d1, d2, d3, d4, d5, d6, d7 be 1, 10, 21, 34, 70, 100, 350 respectively.

Now calculate Minimum number of coins to get a

Value = 140 cents

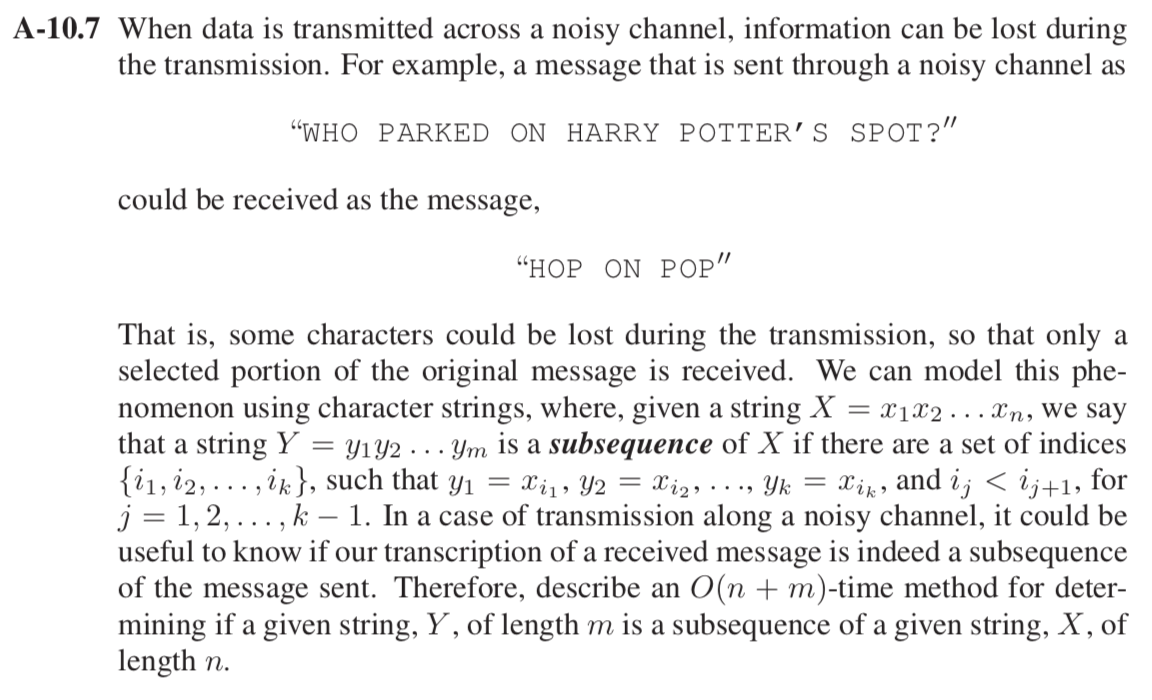
Using **Greedy Approach**, the minimum coin selected

140 cents = 100 + 34 +1 + 1 + 1 + 1 + 1 + 1

But according to **optimal solution**, the minimum coin selected will be

140 cents = 70 + 70

So, greedy change making algorithm will not use minimum number coins.



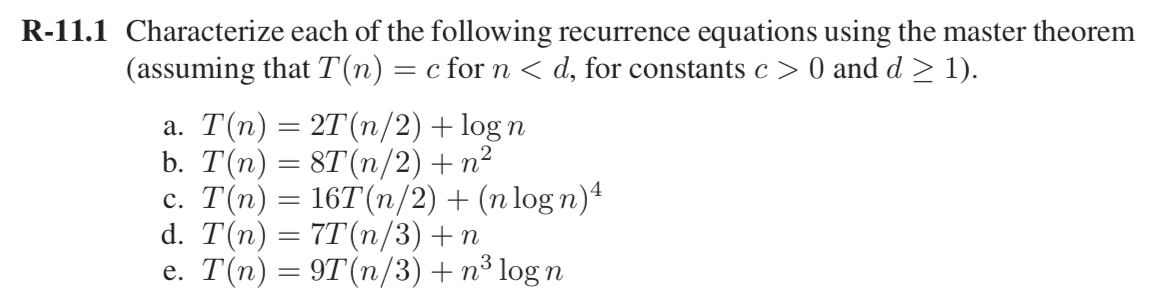
Solution:

Considering a string X that is an original string of length n, and a subsequence string Y of length m.

Using a greedy approach, we can solve this problem in linear time.

1. Initially starting with the first character of the message received at the receiver end and finding the same character in the input transmission.
2. Now second step is to find the second character “O” which is the third of first word in the original. Ignoring the characters which are already matched.
3. Continuously calling above two steps until all the characters in the output transmission have been matched.
4. Finally end the matching if no characters are left to match with the original string X.

Time Complexity: Matching will be done with the original string X of length n and subsequence string Y of length m. Total comparisons made will be O(n+m)



Solution:

1. T(n) = 2T(n/2) + log n

a= 2, b= 2, f(n) = log n

= = n

log n = O(). For

= O(n1-ε) (Using case 1 of master method)

T(n) = θ(

= θ( n)

1. T(n) = 8T(n/2) + n2

a= 8, b= 2, f(n) = n2

= = n3

n2 = O(). For

= O(n3-ε) (Using case 1 of master method)

T(n) = θ(

= θ( n3)

1. T(n) = 16T(n/2) + (n log n)4

a= 16, b= 2, f(n) = (n log n)4

= = n4

(n log n)4 = θ()

T(n) = θ() (Using case 2 of master method)

= θ n4(log n)5

1. T(n) = 7T(n/3) + n

a= 7, b= 3, f(n) = n

= = n1.77

n= O(). For

= O(n1.77-ε) (Using case 1 of master method)

T(n) = θ(

= θ( n1.77)

1. T(n)=9T(n/3) +n3 logn

a= 9, b= 3, f(n) = n3 log n

= = n2

n3 log n = O(). For

provided a.f(n/b) ≤ δ f(n) for some δ < 1

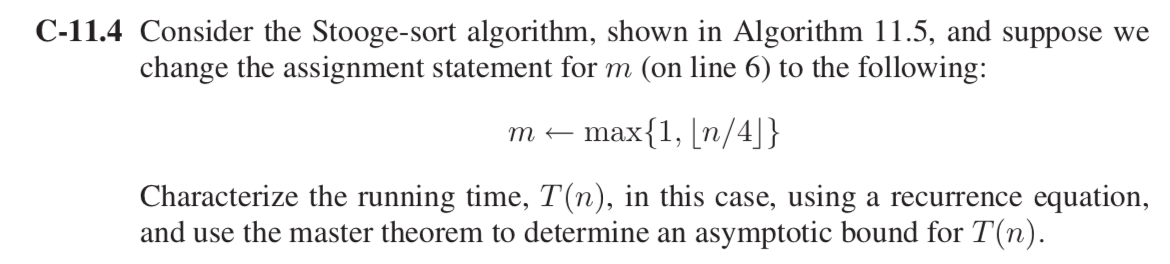
a.f(n/b) = 9(n/3)3 log(n/3)

= n3/3 log(n/3)

≤ δ f(n). (when δ = 1/3 and n≥ 1)

(Using case 3 of master method)

T(n) = θ (n3 log n)



Solution:

Algorithm says: if the input size, n = 1 or 2, then algorithm sorts the input immediately.

For n > 3, calling stooge sort recursively into parts each of length 3n/4, and in one loop we call stooge sort 3 times recursively

Recurrence Relation for the above algorithm will be:

T(n) = 3T(3n/4) + cn

Using master method

a= 3, b= 4, f(n) = n

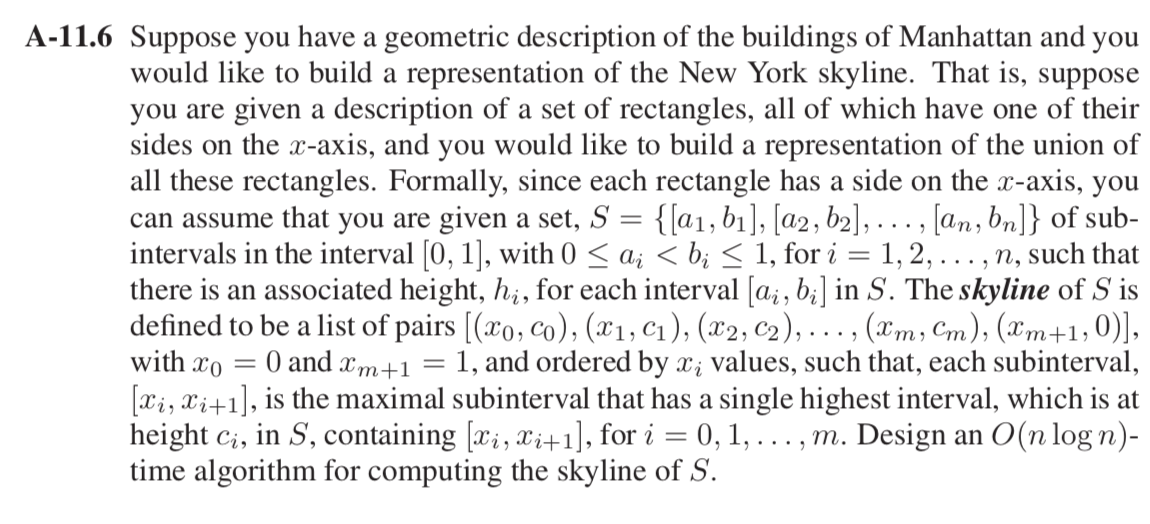
= = n1.26

n= O(). For

= O(n1.26-ε) (Using case 1 of master method)

T(n) = θ(

= θ( n1.26)



Solution:

The skyline is a collection of rectangular strips, each rectangle has a side on the x-axis, we can assume that you are given a set, S = {[a1, b1], [a2, b2], . . . , [an, bn]} of sub- intervals in the interval [0,1],with0 ≤ ai < bi ≤ 1,fori = 1,2,...,n, such that there is an associated height, hi, for each interval [ai, bi] in S.

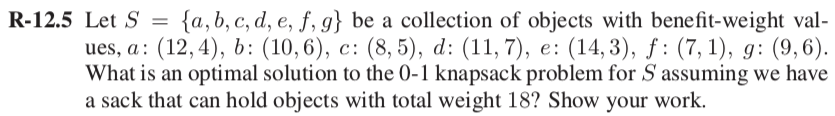
We can use divide and conquer approach to solve the skyline of S.

* 1. First divide the set **S** into two subsets S1 and S2, then divide further recursively in two halves until single element is left in the set.
  2. Then start merging the elements in bottom up fashion.
  3. Merging will be done similar to merge sort. Start from the first strips of two skyline, compare their ***x*** coordinates. Pick the strip which has smaller ***x*** coordinate and add it to the result.
  4. The height of added strip is considered as maximum of current heights from set S1 and S2.

**Time Complexity**: As the algorithm divides the set into two halves every time, it will form the recurrence T(n) = 2 T(n/2) and takes O(log n) time (as the height of the tree).

Merging will be done in O(n) time as the work done at the nodes of depth i.

Total running time for the above algorithm is O(n log n).



Solution:

Total weight object can hold = 18

a: (12, 4), b: (10, 6), c: (8, 5), d: (11, 7), e: (14, 3), f: (7, 1), g: (9, 6)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| B, W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12, 4 | 1 | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| 10, 6 | 2 | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 |
| 8, 5 | 3 | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 20 | 22 | 22 | 22 | 22 | 22 | 30 | 30 | 30 | 30 |
| 11, 7 | 4 | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 20 | 22 | 23 | 23 | 23 | 23 | 30 | 31 | 33 | 33 |
| 14, 3 | 5 | 0 | 0 | 0 | 14 | 14 | 14 | 14 | 26 | 26 | 26 | 26 | 26 | 34 | 36 | 37 | 37 | 37 | 37 | 44 |
| 7,1 | 6 | 0 | 0 | 0 | 14 | 14 | 14 | 14 | 26 | 26 | 26 | 26 | 26 | 34 | 36 | 37 | 37 | 37 | 37 | 44 |
| 9,6 | 7 | 0 | 0 | 0 | 14 | 14 | 14 | 14 | 26 | 26 | 26 | 26 | 26 | 34 | 36 | 37 | 37 | 37 | 37 | 44 |

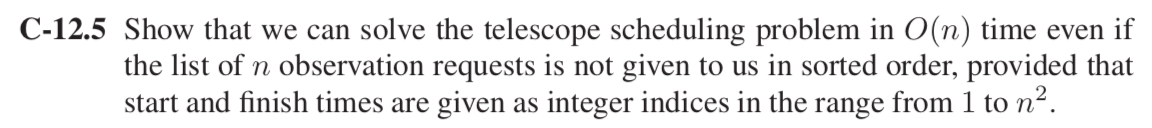
Filling values according to the formula:

B[k, w] = B[k−1,w] if wk >w

max{B[k−1, w], B[k−1, w−wk]+bk} else.

Maximum benefit is 44.

The optimal solution will be got from {(12,4), (10,6), (8,5), (14,3)}



Solution:

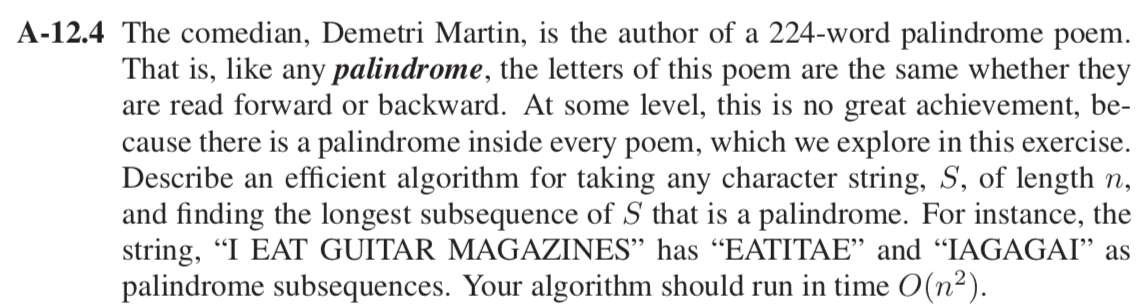
Algorithm for telescope scheduling:

Ordering of requests by finish times and an array, P, so that P[i] = pred(i), then we can fill in the array, B

B[0] ← 0  
for i = 1 to n do

B[i] ← max{B[i − 1], B[P [i]] + bi} A

If the list of n observations is not sorted than it will take O(n log n) time to sort and O(n) times for the *for* loop given in the algorithm. So the total running time for the algorithm will be O(n log n + n). For large value of n we can consider the value O(n) neglecting log n. Sorting n observations can be easily done (using only O(n) extra storage) in O(n) time. So computing the entire array B in O(n) time. For input 1 to n2 we can implement dynamic programming to find the optimal solution. But if the size varies to 2n then brute force algorithm works better.



Solution:

Algorithm for finding the longest subsequence of S that is a palindrome.

Input: Consider a palindrome string S of length [0..n-1].

Output: the length L[i, j] of longest common sequence of a palindrome S[0..n-1]

**for** i ← 1 to n **do**

L[i, i] ← 1

**for** i ← 2 to n **do**

L[i, i-1] ← 0

**for** i ← n-1 to 2 **do**

**for** j ← i + 1 to n **do**

**if** L[i] = L[j] **then**

L[i, j] = L [i+1, j-1] + 2

**else**

L[i, j] = max {L[i+1, j], L [i, j-1]}

**return** array L

**Explanation**: Initially check the first and last character of the sequence. This leads to two possibilities that they are same or not.

If both characters are same, add 2 to the result and remove both the characters and put it in the array L. Else solve the problem considering the first character and compute the remaining subsequence. Then repeat the previous step by choosing the last character and solve it. Maximum of both the results will be taken.

**Time Complexity**: as the algorithm uses two dominated nested for-loops, with both inner and outer iterating n times. And if statement and assignment inside the loop requires O(1) time. The total running time of the algorithm is O(n2).