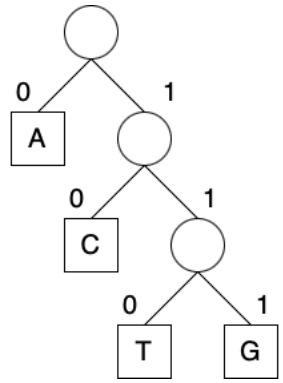
Jay Kalyanbhai Savani
CWID-20009207

EXERCISE | 10.5.7

(a) Fred says that he ran the Huffman coding algorithm for the four characters, **A**, **C**, **G**, and **T**, and it gave him the code words, 0, 10, 111, 110, respectively. Give examples of four frequencies for these characters that could have resulted in these code words or argue why these code words could not possibly have been output by the Huffman coding algorithm.

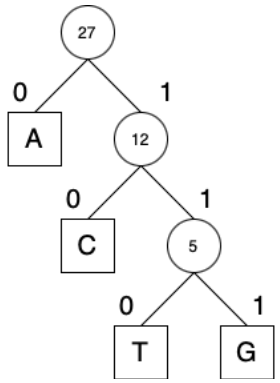
Feedback?

Ans:



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

Example for Frequency 4:



Here, the lengths of the codes for the letters A, C, T, and G vary, and they are given such that the least often used characters come before the most frequently used ones and vice versa. The prefix code, which states that no code is a prefix of any other code, is satisfied.

Since a word's frequency of appearance is higher, it has been given the smallest bit, or 0, or 0.

G is given the biggest bit, or 111, since it is the least frequently used word.

This method aids in obtaining the greatest compression.

EXERCISE | 10.5.16

(a) Give an example set of denominations of coins so that a greedy change making algorithm will not use the minimum number of coins.

Feedback?

Ans: An mathematical paradigm known as the greedy algorithm builds develops a solution piece by piece, constantly selecting the component that provides the most glaringly evident and immediate advantage as the next step. Thus, Greedy fits well with issues where selecting locally optimum also results in a global solution.

It is a strategy for addressing issues by choosing the best alternative at the time. It is unconcerned with whether the most excellent outcome at the moment will provide the final best result. Even if the option was the erroneous one, the algorithm never goes back and changes its mind. It functions in a top-down manner. It's possible that not all issues can be solved using this approach. It does this because it constantly seeks to create the optimal outcome at the local level.

Let's think about the following example:

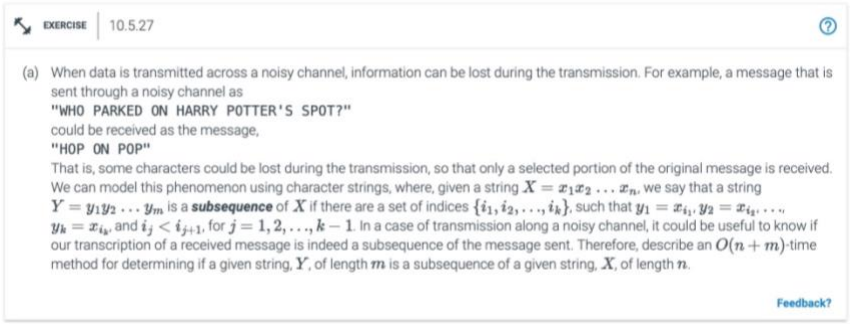
Assume that d1, d2, d3, d4, d5, d6, and d7 have the values 1, 5, 10, 20, 30, 75, 100, and 250.

Now, we figure out how many coins must be present in order to equal $1.50.

The minimum coins chosen, 150 cents, are calculated using the Greedy Method as follows: 100 + 30 + 5 + 1 + 1 + 1 + 1

Yet 150 cents = 75 plus 75 will be the best option.

Hence, the greedy algorithm will not employ the required minimum amount of coins.



EXERCISE | 10.5.27

(a) When data is transmitted across a noisy channel, information can be lost during the transmission. For example, a message that is sent through a noisy channel as
"WHO PARKED ON HARRY POTTER'S SPOT?"
could be received as the message,
"HOP ON POP"
That is, some characters could be lost during the transmission, so that only a selected portion of the original message is received. We can model this phenomenon using character strings, where, given a string $X = x_1x_2 \ldots x_n$, we say that a string $Y = y_1y_2 \ldots y_m$ is a ***subsequence*** of X if there are a set of indices $\{i_1, i_2, \ldots, i_k\}$, such that $y_1 = x_{i_1}$, $y_2 = x_{i_2}, \ldots, y_k = x_{i_k}$, and $i_j < i_{j+1}$, for $j = 1, 2, \ldots, k-1$. In a case of transmission along a noisy channel, it could be useful to know if our transcription of a received message is indeed a subsequence of the message sent. Therefore, describe an $O(n+m)$-time method for determining if a given string, Y, of length m is a subsequence of a given string, X, of length n.

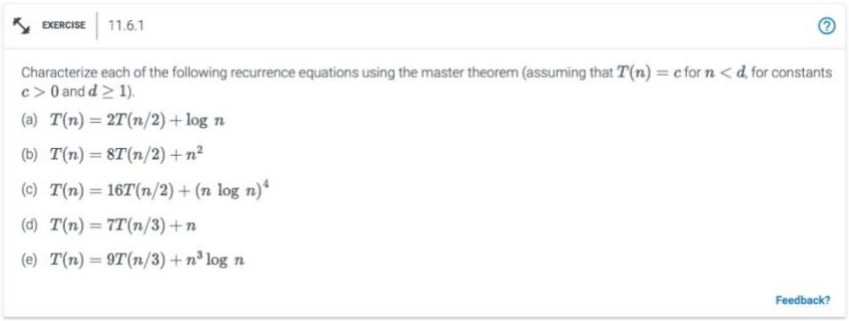
Feedback?

Ans: Think of a string K, which is a string of length n, and a string L, which is a string of length m.

We find a linear time solution to the issue using a greedy strategy.

- We begin by looking for the same character in the input transmission as the first character of the message received at the receiver end.
- Next, disregarding the characters that have previously been matched, we look for the second character, "O," which is the third letter of the first word in the original transmission.
- After all the characters in the output transmission have been received and matched, step c. is repeated until they have, and step d. is reached when there are no more characters remaining to match with the initial string K.

The first string K of length n and the succeeding string L of length m will be used for the matching. As a result, the total number of comparisons will be O(n+m), and this algorithm's time complexity will be O(n+m).



EXERCISE | 11.6.1

Characterize each of the following recurrence equations using the master theorem (assuming that $T(n) = c$ for $n < d$, for constants $c > 0$ and $d \geq 1$).

(a) $T(n) = 2T(n/2) + \log n$

(b) $T(n) = 8T(n/2) + n^2$

(c) $T(n) = 16T(n/2) + (n \log n)^4$

(d) $T(n) = 7T(n/3) + n$

(e) $T(n) = 9T(n/3) + n^3 \log n$

Feedback?

Ans: The right use of the Master Theorem approach, which is highly generic and does not explicitly employ induction, allows one to solve divide-and-conquer recurrence problems. A "cookbook" technique for establishing the asymptotic characterisation of a large range of recurrence equations is the master theorem.

It is specifically used for recurrence equations of the kind

$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d, \end{cases}$$

where $d \geq 1$ is an integer constant, $a \geq 1$, $c > 0$, and $b > 1$ are real constants, and f(n) is a function that is positive for $n \geq d$.

a) $T(n) = 2T(n/2) + \log n$
a= 2, b= 2, f(n) = log n

$n^{\log_b a} = n^{\log_2 2} = n$
$\log n = O(n^{\log_b a - \varepsilon})$. For $\varepsilon > 0$
$= O(n^{1-\varepsilon})$ (Using case 1 of master method)
$T(n) = \theta(n^{\log_b a})$
$= \theta(n)$

b) $T(n) = 8T(n/2) + n^2$
a= 8, b= 2, $f(n) = n^2$

$n^{\log_b a} = n^{\log_2 8} = n^3$

$n^2 = O(n^{\log_b a - \varepsilon})$. For $\varepsilon > 0$
$= O(n^{3-\varepsilon})$ (Using case 1 of master method)
$T(n) = \theta(n^{\log_b a})$
$= \theta(n^3)$

c) $T(n) = 16T(n/2) + (n \log n)^4$

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

$n^{\log_b a} = n^{\log_2 16} = n^4$

$(n \log n)^4 = \theta(n^{\log_b a} \log^k n)$

$T(n) = \theta(n^{\log_b a} \log^{k+1} n)$ (Using case 2 of master method)

$= \theta\, n^4 (\log n)^5$

d) $T(n) = 7T(n/3) + n$

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

$n^{\log_b a} = n^{\log_3 7} = n^{1.77}$

$n = O(n^{\log_b a - \varepsilon})$. For $\varepsilon > 0$

$= O(n^{1.77-\varepsilon})$ (Using case 1 of master method)

$T(n) = \theta(n^{\log_b a})$

$= \theta(n^{1.77})$

e) $T(n)=9T(n/3) + n^3 \log n$

a= 9, b= 3, $f(n) = n^3 \log n$

$n^{\log_b a} = n^{\log_3 9} = n^2$

$n^3 \log n = O(n^{\log_b a + \varepsilon})$. For $\varepsilon = 1$

Given, $a.f(n/b) \leq \delta\, f(n)$ for some $\delta < 1$

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

$= n^3/3 \log(n/3)$

$\leq \delta\, f(n)$. (when $\delta = 1/3$ and $n \geq 1$)

(Using case 3 of master method)

$T(n) = \theta(n^3 \log n)$

EXERCISE | 11.6.10

(a) Consider the Stooge-sort algorithm, shown in Algorithm 11.6.1, and suppose we change the assignment statement for m (on line 6) to the following:

$$m \leftarrow \max\{1, \lfloor n/4 \rfloor\}$$

Characterize the running time, $T(n)$, in this case, using a recurrence equation, and use the master theorem to determine an asymptotic bound for $T(n)$.

Feedback?

Ans:

Algorithm StoogeSort(A, i, j):
  Input: An array, A, and two indices, i and j, such that $1 \leq i \leq j \leq n$
  Output: Subarray, $A[i..j]$, sorted in nondecreasing order
  $n \leftarrow j - i + 1$ // The size of the subarray we are sorting
  if $n = 2$ **then**
    if $A[i] > A[j]$ **then**
      Swap $A[i]$ and $A[j]$
  else if $n > 2$ **then**
    $m \leftarrow \lfloor n/3 \rfloor$
    StoogeSort$(A, i, j - m)$ // Sort the first part
    StoogeSort$(A, i + m, j)$ // Sort the last part
    StoogeSort$(A, i, j - m)$ // Sort the first part again
  return A

According to the Stooge-Sort Algorithm, the method sorts the input quickly if n = 1 or 2. Nevertheless, if n is more than 3, we execute the Stooge-sort algorithm recursively into segments of length ((3(n)) / 4) and call it three times in a loop.

The following will serve as the algorithm's recurrence relation:
T(n) = 3T ((3(n)) / 4) + cn

Using master method:
a = 3, b = 4, f(n) = n

$n^{\log_b a}$ = $n^{\log_3 4}$ = $n^{1.26}$

n = O($n^{\log_b a - \varepsilon}$). For $\varepsilon > 0$
  = O($n^{1.26-\varepsilon}$) (Using case 1 of master method)

T(n) = θ($n^{\log_b a}$)
  = θ($n^{1.26}$)

EXERCISE | 11.6.17

(a) Suppose you have a geometric description of the buildings of Manhattan and you would like to build a representation of the New York skyline. That is, suppose you are given a description of a set of rectangles, all of which have one of their sides on the x-axis, and you would like to build a representation of the union of all these rectangles. Formally, since each rectangle has a side on the x -axis, you can assume that you are given a set, $S = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$ of sub-intervals in the interval $[0, 1]$, with $0 \leq a_i < b_i \leq 1$, for $i = 1, 2, \ldots, n$, such that there is an associated height, h_i, for each interval $[a_i, b_i]$ in S. The ***skyline*** of S is defined to be a list of pairs $[(x_0, c_0), (x_1, c_1), (x_2, c_2), \ldots, (x_m, c_m), (x_{m+1}, 0)]$, with $x_0 = 0$ and $x_{m+1} = 1$, and ordered by x_i values, such that, each subinterval, $[x_i, x_{i+1}]$, is the maximal subinterval that has a single highest interval, which is at height c_i, in S, containing $[x_i, x_{i+1}]$, for $i = 0, 1, \ldots, m$. Design an $O(n \log n)$-time algorithm for computing the skyline of S.

Feedback?

Ans: We suppose that we are given a set, S = [a1, b1], [a2, b2],..., [an, bn] of sub-intervals in the interval [0, 1], with 0 ai bi 1, for I = 1, 2,..., n, such that there is an associated height, hi, for each range [ai, bi] in S. The skyline is made up of a collection

Here, we may solve the S skyline using a divide and conquer strategy.

1. To begin, we split the set S into two subsets, S1 and S2, and then we continue to split the set into two halves recursively until there is only one member remaining in the set.
2. After that, we begin fusing the components from the bottom up.
3. Merging will proceed similarly to merge sort. Compare the x coordinates of the initial strips of the two skylines. Choose the strip with the lower x coordinate, then include it in the outcome.
4. The height of the additional strip is regarded as the highest height currently possible from sets S1 and S2.

The method will produce the recurrence T(n) = 2 T(n/2) and require O(log n) time since it splits the set into two halves every time (as the height of the tree). The work completed at the depth I nodes and the merging will be completed in O(n) time.

Thus, the aforementioned algorithm's overall running time is O (n log n).