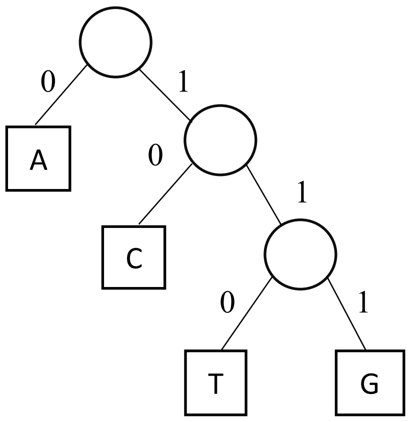
CS 600 HW 5
SHUCHI PARAGBHAI MEHTA
CWID: 20009083

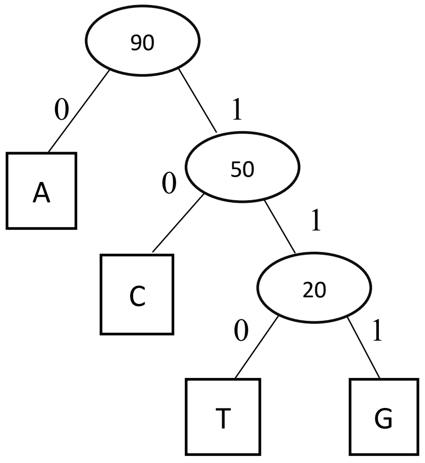
- 10.5.7

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.



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 2,



Here, the length is different for character codes A, C, T and G, which are assigned as, the most frequently used characters to use fewest number of bits and the least used characters to use the greatest number of bits.

It satisfies with the requirement that no code word be the prefix of another code. The smallest bit, 0, has been given because words tend to occur more frequently. And the least used word, G, receives the biggest bit, or 111. By using this method, the maximum compression is possible.

- 10.5.16

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

Let's consider the following example for this,

Suppose value of d_1, d_2, d_3, d_4, d_5, d_6, d_7 be 1, 10, 21, 34, 70, 100, 350 respectively.
Now, Calculating Minimum number of coins to get a Value = 140 cents

According to **Greedy Approach**, the minimum coin selected
140 cents = 100 + 34 +1 + 1 + 1 + 1 + 1 + 1

But if we use **optimal solution**, the minimum coin selected will be 140 cents = 70 + 70
This we can say, greedy change making algorithm will not use minimum number coins.

- 10.5.27

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.

Let's consider a string X that is an original string with length n and a string Y that is a subsequence with length m. In, linear time, we can address this issue using a greedy approach.

a) Finding the same character in the input transmission by starting with the first character of the message that was received at the receiver end.
b) The next step is to locate the second character, "O" which in the original is the third letter of the first word. disregarding characters that have already been matched.
c) Repeating steps, a and b above until every character in the output transmission has been matched.
d) If there are no more characters that match the original string X, the matching is finally done.

The original string X of length n and the subsequence string Y of length m will be matched. Thus, Total comparisons will $O(n+m)$ in size.

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

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 $\varepsilon > 0$

$= O(n^{1-\varepsilon})$ (By 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 $\varepsilon > 0$

$= O(n^{3-\varepsilon})$ (By using case 1 of master method)

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

$= \theta(n^3)$

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

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

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

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

$T(n) = \theta(n^{\log_b a} \log^{k+1} n)$ (By 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 $\varepsilon > 0$

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

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

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

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

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 $\varepsilon = 1$

provided $a.f(n/b) \leq \delta f(n)$ for some $\delta < 1$
$a.f(n/b) = 9(n/3)3 \, log(n/3)$
$= n3/3 \, log(n/3)$
$\leq \delta f(n)$. $(when \; \delta = 1/3 \; and \; n \geq 1)$

$(Using \; case \; 3 \; of \; master \; method)$
$T(n) = \theta(n^3 \, log \, n)$

- 11.6.10

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 $T(n)$.

According to the algorithm, the input is quickly sorted if n = 1 or 2.
Stooge sort is called three times recursively in one loop for n > 3, with each portion having a length of 3n/4.

The recurrence relation for the algorithm described above is,

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

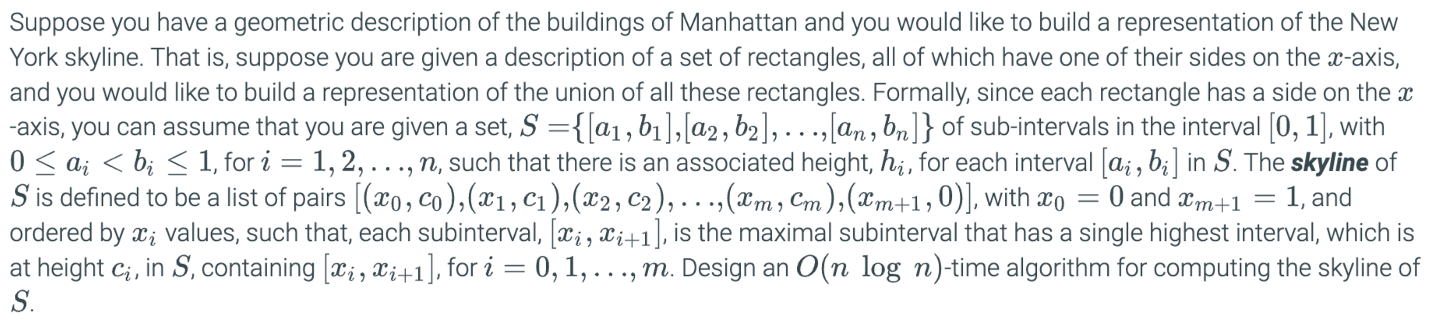
By using master method,

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

$n^{log_b a} = n^{log_3 4} = n1.26$

$n = O(n^{log_b a - \varepsilon})$. $For \; \varepsilon > 0$
$= O(n1.26 - \varepsilon)$ $(Using \; case \; 1 \; of \; master \; method)$
$T(n) = \theta(n^{log_b a})$
$= \theta(n1.26)$

- 11.6.17



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.

As, we are given a set, S, and S = {[a1, b1], [a2, b2], . . ., [an, bn]} of sub- intervals in the interval [0,1], with $0 \leq ai < bi \leq 1$, for i = 1,2,...,n, such that there is an associated height, hi, for each interval $[ai, bi]$ in S. The skyline is made up of a collection of rectangular strips, each of which has a side on the x-axis.

To solve the skyline of S, we can use a divide and conquer approach.

A. Split the set S first into two subsets, S1 and S2, and then divide it again recursively into two halves until only one element remains.
B. Next, begin merging the elements from the bottom up.
C. Merging will proceed similarly to merge sort. Compare the x coordinates of the first strips of the two skylines. Select the strip with the smaller x value, then include it in the result.
D. The new strip's height is calculated as the maximum of the current heights from sets S1 and S2.

The algorithm will generate the recurrence $T(n) = 2\,T(n/2)$ and require $O(log\ n)$ time because it divides the set into two halves every time (as the height of the tree).
The work performed at the depth i nodes and the merging will be completed in $O(n)$ time.
The given algorithm requires $O(n\ log\ n)$ to run completely.

- 12.8.5

Let $S = \{a, b, c, d, e, f, g\}$ be a collection of objects with benefit-weight values, $a:(12,4), b:(10,6), c:(8,5), d:(11,7), e:(14,3), f:(7,1), g:(9,6)$. What is an optimal solution to the 0-1 knapsack problem for S assuming we have a sack that can hold objects with total weight 18? Show your work.

Object can hold Total weight of 18,

As given,

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

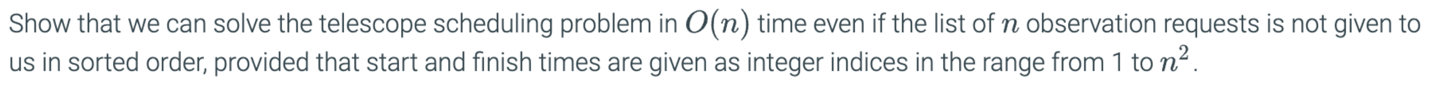
Filling values according to the formula:
According to formula, the values,

$B[k, w] = B[k-1, w]$ $if \quad w_k > w$
$max\{B[k-1, w], B[k-1, w - w_k] + b_k\}$ else.

As found, maximum benefit is 44.

So, the optimal solution can be retrieved from {(12,4), (10,6), (8,5), (14,3)}

- 12.8.14


Show that we can solve the telescope scheduling problem in $O(n)$ time even if the list of n observation requests is not given to us in sorted order, provided that start and finish times are given as integer indices in the range from 1 to n^2.

Algorithm for telescope scheduling problem:

By sorting requests according to their completion times and filling up the array P so that $P[i] = pred(i)$,

```
$B[0] \leftarrow 0$

$for\ i = 1\ to\ n\ do$

$B[i] \leftarrow max\{B[i - 1], B[P\ [i]] + b_i\}\ A$
```

If the list of n observations is not sorted, sorting will take $O(n\ log\ n)$ time, and the for loop in the algorithm will run $O(n)$ times. As a result, the algorithm's total running time will be $O(n\ log\ n + n)$. When n is large, we can think about the value $O(n)$, ignoring $log\ n$.

It is simple to sort n observations in $O(n)$ time using only $O(n)$ extra storage. Thus, it takes $O(n)$ time to compute the complete array B. We can use dynamic programming to determine the best solution for inputs 1 to n^2. However, the brute force algorithm performs better when the size varies by 2^n.

- 12.8.30

The comedian, Demetri Martin, is the author of a 224-word palindrome poem. That is, like any ***palindrome***, the letters of this poem are the same whether they are read forward or backward. At some level, this is no great achievement, because there is a palindrome inside every poem, which we explore in this exercise. Describe an efficient algorithm for taking any character string, S, of length n, and finding the longest subsequence of S that is a palindrome. For instance, the string, "I EAT GUITAR MAGAZINES" has "EATITAE" and "IAGAGAI" as palindrome subsequences. Your algorithm should run in time $O(n^2)$.

Algorithm to find the longest subsequence of S which is a palindrome.

Input: Consider a palindrome string S of length $[0..n-1]$.
Output: The length of a palindrome's longest common sequence, $L[i,j]$, which is $S[0..n-1]$.

```
$\boldsymbol{for}\ i \leftarrow 1\ to\ n\ \boldsymbol{do}$
        $L[i,i] \leftarrow 1$
$\boldsymbol{for}\ i \leftarrow 2\ to\ n\ \boldsymbol{do}$
        $L[i,i-1] \leftarrow 0$
$\boldsymbol{for}\ i \leftarrow n-1\ to\ 2\ \boldsymbol{do}$
        $\boldsymbol{for}\ j \leftarrow i\ +\ 1\ to\ n\ \boldsymbol{do}$
                $\boldsymbol{if}\ \ L[i]\ =\ L[j]\ \boldsymbol{then}$
                    $L[i,j]\ =\ L\ [i+1, j-1]\ +\ 2$
                $\boldsymbol{else}$
                    $L[i,j]\ =\ max\ \{L[i+1,j],\ L[i,j-1]\}$
$\boldsymbol{return}\ array\ L$
```

Check the sequence's first and last characters first. There are two outcomes from this: either they are the same or not.
If the outputs are similar, remove both characters, add 2, and store the result in the array L. If not, determine the remainder of the subsequence while solving the problem with the initial character in mind. In order to solve it, go back to the previous stage and select the final character. The best of both outcomes will be used.

Since, the algorithm uses two nested for-loops, one inside the other, both of which run n times. Additionally, the loop's if statement and allocation take up $O(1)$ time. Thus, the algorithm takes $O(n^2)$ time to complete.