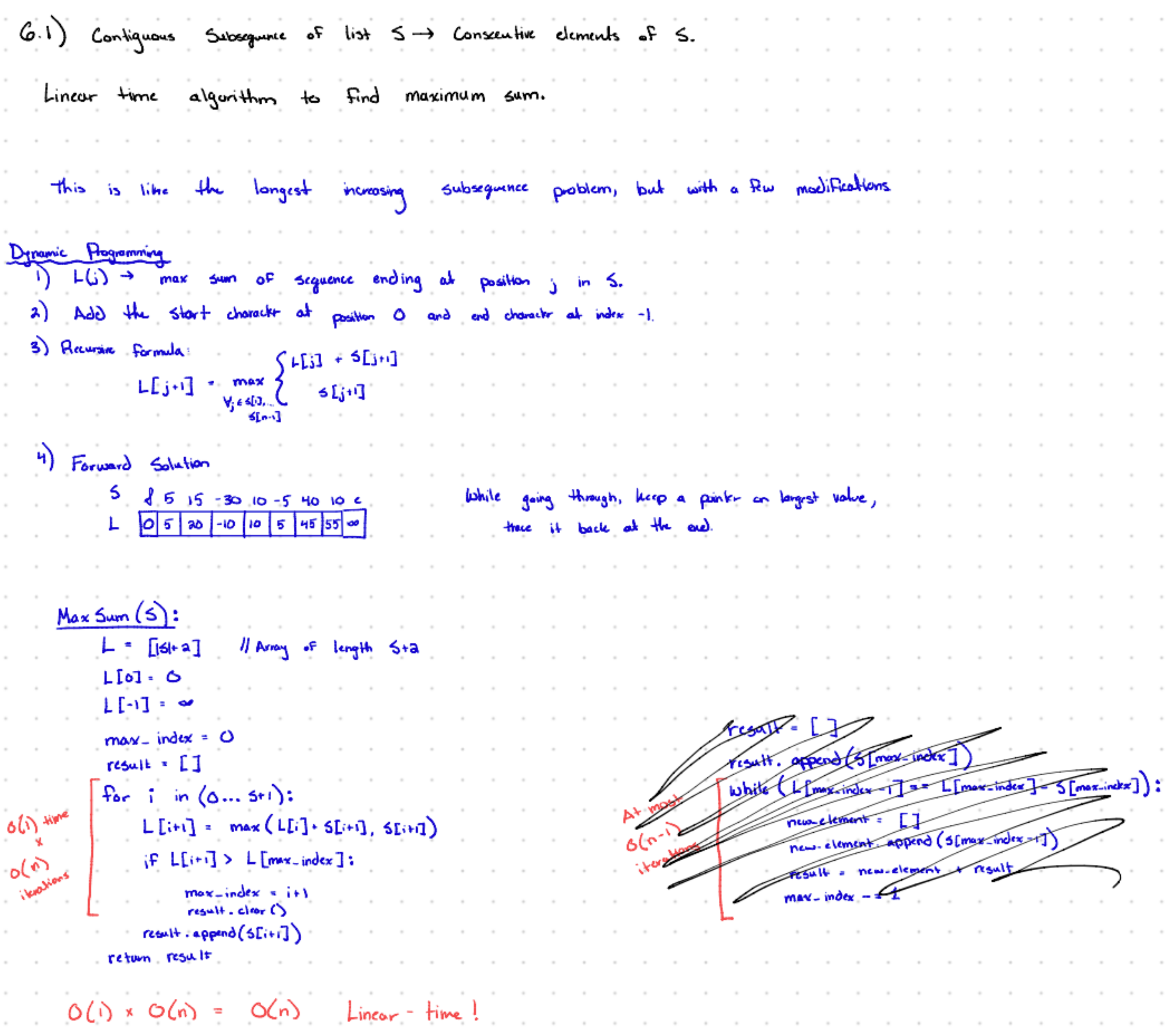
Intro to Algorithms: Homework #9

Due on April 22, 2021

Prof. Zaki

Jared Gridley



6.1) Contiguous Subsequence of list S → Consecutive elements of S.

Linear time algorithm to find maximum sum.

this is like the longest increasing subsequence problem, but with a few modifications

Dynamic Programming

1) L(j) → max sum of sequence ending at position j in S.

2) Add the start character at position 0 and end character at index -1.

3) Recursive formula:

$$L[j+1] = \max_{\forall_j \in S[1], \ldots S[n-1]} \begin{cases} L[j] + S[j+1] \\ S[j+1] \end{cases}$$

4) Forward Solution

S	∂	5	15	-30	10	-5	40	10	c
L	0	5	20	-10	10	5	45	55	∞

While going through, keep a pointer on largest value, trace it back at the end.

Max Sum (S):

```
L = [|S|+2]     // Array of length S+2
L[0] = 0
L[-1] = ∞
max_ index = 0
result = []
for i in (0... S+1):
    L[i+1] = max(L[i] + S[i+1], S[i+1])
    if L[i+1] > L[max_index]:
        max_index = i+1
        result.clear()
    result.append(S[i+1])
return result
```

O(1) time × O(n) iterations

~~result = [ ]~~

~~result. append(S[max_index])~~

~~while (L[max_index - 1] == L[max_index] - S[max_index]):~~

~~new_element = [ ]~~

~~new_element. append (S[max_index - 1])~~

~~result = new_element + result~~

~~max_index -= 1~~

At most O(n-1) iterations

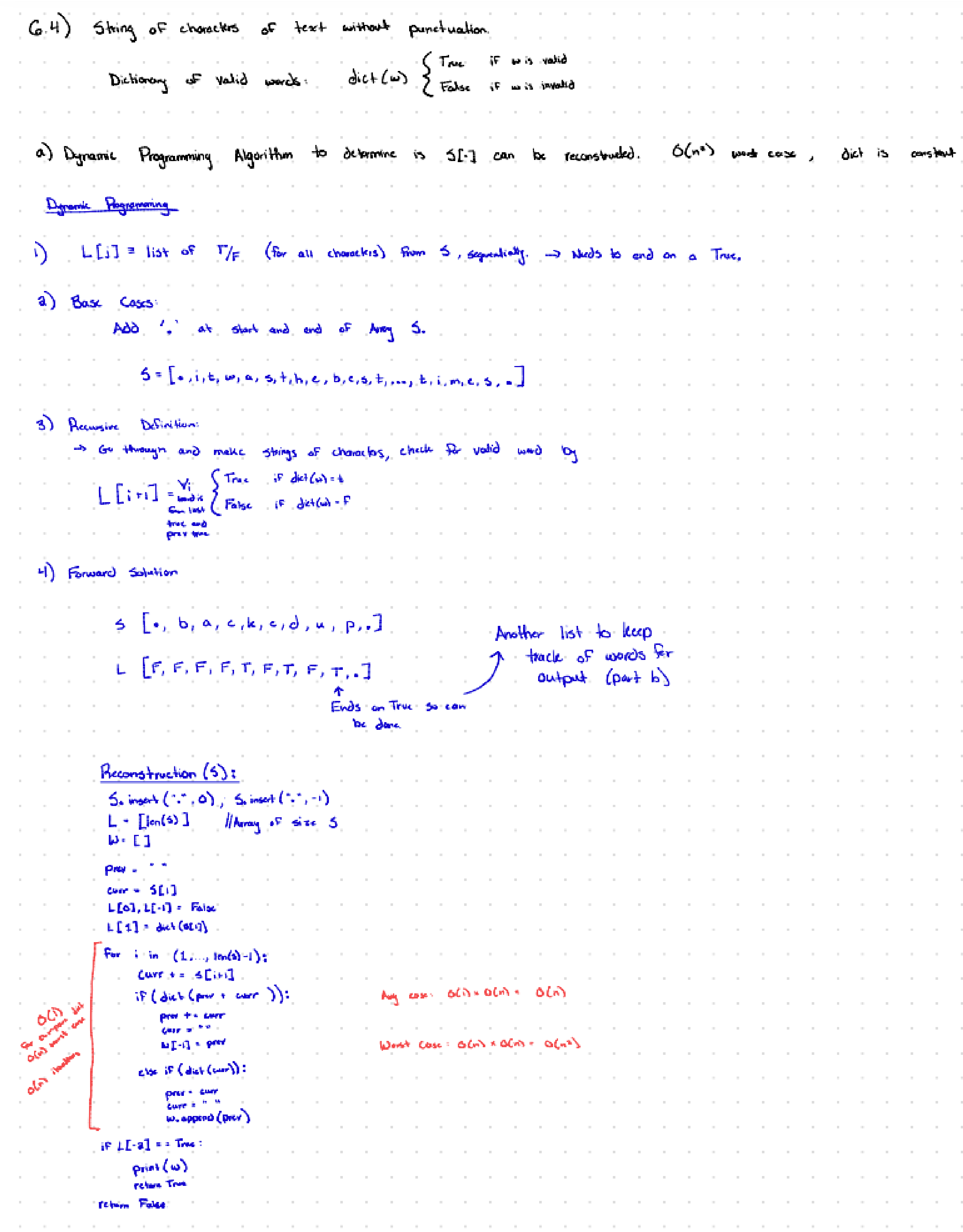
O(1) × O(n) = O(n) Linear - time !

Figure 1: Page 1

Lab Results:

```
Results:

cox1-protein.fasta
edit distance = 230
alignment:
M-VQRWLYSTNAKDIAVLYFMLAIFSGMAGTAMSLIIRLELAAPGSQYLHGNSQLFNVLVVGHAVLMIFFLVMPALIGGFGNYLLPLMIGATDTA
FPRINNIAFWVLPMGLVCLVTSTLVESGAGTGWTVYPPLSSIQAHSGPSVDLAIFALHLTSISSLLGAINFIVTTLNMRTNGMTMHKLPLFVWSI
FITAFLLLLSLPVLSAGITMLLLDRNFNTSFFEVSGGGDPILYEHLFWFFGHPEVYILIIPGFGIISHVVSTY-S-KKPVFGEISMVYAMASIGL
LGFLVWSHHMYIVGLDADTRAYFTSATMIIAIPTGIKIFSWLATIHGGSIRLATPMLYAIAFLFLFTMGGLTGVALANASLDVAFHDTYYVVGHF
HYVLSMGAIFSLFAGYYYWSPQILGLNYNEKLAQIQFWLIFIGANVIFFPMHFLGINGMPRRIPDYPDAFAGWNYVASIGSFIATLSLFLFIYIL
YDQLVNGLNNKVNNKSVIYNKAPDFVESNTIFNLNTVKSSSIEFLLTSPPAVHSFNTP-AVQS

MFADRWLFSTNHKDIGTLYLLFGAWAGVLGTALSLLIRAELGQPG--NLLGNDHIYNVIVTAHAFVMIFFMVMPIMIGGFGNWLVPLMIGAPDMA
FPRMNNMSFWLLPPSLLLLLASAMVEAGAGTGWTVYPPLAGNYSHPGASVDLTIFSLHLAGVSSILGAINFITTIINMKPPAMTQYQTPLFVWSV
LITAVLLLLSLPVLAAGITMLLTDRNLNTTFFDPAGGGDPILYQHLFWFFGHPEVYILILPGFGMISHIV-TYYSGKKEPFGYMGMVWAMMSIGF
LGFIVWAHHMFTVGMDVDTRAYFTSATMIIAIPTGVKVFSWLATLHGSNMKWSAAVLWALGFIFLFTVGGLTGIVLANSSLDIVLHDTYYVVAHF
HYVLSMGAVFAIMGGFIHWFPLFSGYTLDQTYAKIHFTIMFIGVNLTFFPQHFLGLSGMPRRYSDYPDAYTTWNILSSVGSFI---SL-------
--TAV-ML-----MIFMI-WEA--F-ASKRKVLMVEEPSMNLEWLYGCPPPYHTFEEPVYMKS


cox1-dna.fasta
edit distance = 634
alignment:
AT-GTTC-GCCGA--CCGT-T---G-ACTATT--C--TCT-A-CA-A-A---CCA--CA-AA-GACATTGGAACACTATACCT-ATTATTCGGCG
CAT-GAGCTGGAGTCCTAGGCAC-AGCTCTAAGCCTCCTTA-TTCGAGCCGAGCTGGGCCA-GCCAGGCAACCTTCT-AGGTAACG-ACCACATC
TACAA--C-GTTATCGTCACAGCCCATGC-ATTTGTAATAATCTTCTTCATAGTAATACCCATCATAATCGGAGGCTTTGGCAACTGACTAGTTC
CCCTAATAATCGGTGCCCCCGATATGGCGTTTCCCCGCATAAACAACATAAGCTTCTGACTCTTACCTCCCTCTCTCCTACTCCTGCTCGCAT-C
TGCTATAGTGGAGGCCGG-AGCAGGAACAGGTTGAACAGTCTACCCTCCCTTAGCAGGGAACTA-CTCCCACCCTGGAGCCTCCGTAGACCTAAC
CATCTTCTCCTT-ACACCTAGCAGGTGT-CTCCTCTATCTTAGGGGCCATCAATTTCATCACAACAATTATCAATATAAAACCCCCTGCCATAAC
CCAATACCAAACGCCCCTCTTCGTCTGATCCGTCCTAATCACAGCAGTCCT-ACTTCTCCTATCTCTCCCAGTCCTAGCTGCTGGC-ATCACTAT
ACTACTAACAGACCGCAACCTCAACACCACCTTCTTCGA-CCCCG-CCGGAGGAGGAGACCCCATTCTATACCAACACCTATTCTGATTTTTCGG
TCACCCTGAAGTTTATATTCTTATCCTACCAGGCTT-CGGAATAATCTCCCATATTGTA-ACTT-ACTACTCCGGAAAAAAAGAACCATTTGGAT
ACATAGG-TATGGTCTGAGCTATGATATCAATTGGCTTCCTAGGGTTTATCGTGTGAGCACACCATATATTTACAGTAGGAATAGACGTAGACAC
ACGAGCATATTTCACCTCCGCTACCATAATCATCGCTATCCCCACCGGCGTCAAAGTATTTAGCTGACTCGCCACACTCCACGGAAGCAATATGA
AAT-GA-TCTGCT-GCAGTGCTCTGAGCCCTAGGATTCATCTTTCTTTTCACCGTAGGTGGCCTGACTGGCATTGTATTAGCAAACTCATCACTA
GACATCGTACTACACGACACGTACTACGTTGT-AGCCCACTTCCACTATGTCCTATCAATAGGAGCTGTATTTGCCATCAT-A-GGAGGCTTCAT
TCACTGATTTCCCCT-ATTCTCAGGCTACACCCTAGACCAAACCTACGCCAAAATCCA-T-TTCACTATCAT-ATTCATCGGCGTAAATCTAACT
TTCTTCCCACAACACTTTCTCGGCCTATCCGGAATGCCCCG-ACGTTACTCGGACTACCCCGATGCATACACCACATGAAACATCCTATCATCTG
TA-GGCTC-ATTCATTTCTCTAACAGCAGTAATATTAATAATT-TTCATG-ATT-TGAGAAGCCTTCGCTT-CG-AAGCGAAAAGTCCTAAT-AG
TAGAAGAACCCTCCA-TA-AA--CCTG-G-A--GT--GACTA-T-ATGGATGCCCCCCACCCT-ACCA--C--ACATT---CGAAGAACCCGTAT
A-CATAAAATCTAGA

ATTAATCTTTATAAAAAATATCAAGGAGGATTGGCAGTTTGATTAGAGAGATCTAATCATAAAGATATCGGAACTCTTTATTTTATT-TTTGG-A
CTTTGATCTGGTATGGTTGGTACTAGAT-T-TTCTTTATTAATTCGTTTAGAATTAG-CTAAACCAGGTTTTTTTCTTAGG-AATGGAC-AGTTG
TATAATTCAGTTAT--T-ACAGCTCATGCTATTT-TAATAATTTTTTTTATGGTAATACCTACTATAATCGGTGGTTTTGGTAACTGATTATTAC
CACTTATGTTAGGAGCACCTGATATAAGATTTCCACGTTTAAATAATTTAAGATTTTGGTTATTACCTACATCTATATTATTAATTTTAG-ATGC
TTGTTTTGTAGATATAGGTTGTGGGACTAGGT-GAACAGTCTACCCACCTTTA--AG--AACAATGGGGCACCCTGGAAGTAGAGTAGATTTAGC
TAT-TTTTAGTTTACATGCAGCAGG-GTTAAGATCTATCTTAGGTGGTATTAATTTTATGTGTACTACTAAAAATTTACGTAGAAGTTCTATTTC
```

```
ATTAGAACATATAACTTTATTTGTTTGAACTGTATTTGTAACAG-TGTTTTTACTGGTTTTATCTCTACCGGTTTTAGCAG-GGGCTATTACTAT
GTTGTTAACTGATCGTAATTTAAATACTTCATTTTTTGATCCAAGAACTG-G-AGGTAATCCTCTTATTTATCAACATTTGTTTTGATTTTTTGG
TCATCCTGAAGTATATATTTTGATTTTACCAG-CTTTTGGTATTGTCAGACA-AT-CTACACTTTATTTAACAGGAAAAAAAGAAGTTTTTGG-T
GCTTTGGGTATAGTTTATGCAATTTTAAGAATTGGTTTAATTGGTTGTGTAGTATGAGCTCACCATATGTATACAGTAGGTATAGATTTGGATTC
ACGTGCTTATTTTTCGGCTGCTACTATAGTTATTGCAGTGCCAACAGGTGTTAAAGTGTTTAGATGATTGGCTACATTATTTGGTATAAAAATGG
TATTTAATCCACTTTTATTG-T--GAGTATTGGGTTTTATTTTTTTGTTTACTTTAGGTGGGTTGACAGGTGTTGTATTATCTAATTCAAGATTG
GATATTATTTTACATGATACTTATTATGTAGTTAGAC-ATTTTCATTATGTTTTAAGTTTAGGAGCTGTTTTTGGGATTTTCACGGGTGTTACAC
T-A-TGATGAAGATTTATT-ACAGGGTATGTGTTAGA-TAAACTTATGATA-TCTGCAGTATTTA-TTTTATTATTTATTGGGGTAAATTTAACA
TTTTTCCCGCTACATTTTGCAGGACTACACGGGTTCCCACGTAAAT-ATTTAGATTACCCTGATGTTTATTCGGTATGAAATATTATTGCCTCT-
TATGGTTCTATT-ATT---AGAACTGCAGGACTATTCTTATTTATTTATGTATTATTAGA-GTCTT-TCTTTAGTTATCGTTTAGT---AATTAG
-AGATTATTATTCTAATAGAAGACCTGAGTATTGTATGAGTAATTATGTATTTGGTC-ACAGTTATCAGTCTGAGATTTATTTTAGAACTACTAG
ATTAAAAAAT-TAG-
```



6.4) String of characters of text without punctuation.

Dictionary of valid words: $\text{dict}(w) \begin{cases} \text{True} & \text{if } w \text{ is valid} \\ \text{False} & \text{if } w \text{ is invalid} \end{cases}$

a) Dynamic Programming Algorithm to determine is S[·] can be reconstructed. $O(n^2)$ worst case, dict is constant

Dynamic Programming

1) L[i] = list of T/F (for all characters) from S, sequentially. → Needs to end on a True.

2) Base Cases:

Add '.' at start and end of Array S.

S = [., i, t, w, a, s, t, h, e, b, e, s, t, ..., t, i, m, e, s, .]

3) Recursive Definition:

→ Go through and make strings of characters, check for valid word by

$$L[i+1] = \underset{\substack{\text{word is} \\ \text{from last} \\ \text{true and} \\ \text{prev true}}}{\forall_i} \begin{cases} \text{True} & \text{if } \text{dict}(w) = t \\ \text{False} & \text{if } \text{dict}(w) = F \end{cases}$$

4) Forward Solution

S [., b, a, c, k, e, d, u, p, .]

L [F, F, F, F, T, F, T, F, T, .]

Ends on True so can be done.

Another list to keep track of words for output (part b)

Reconstruction (S):

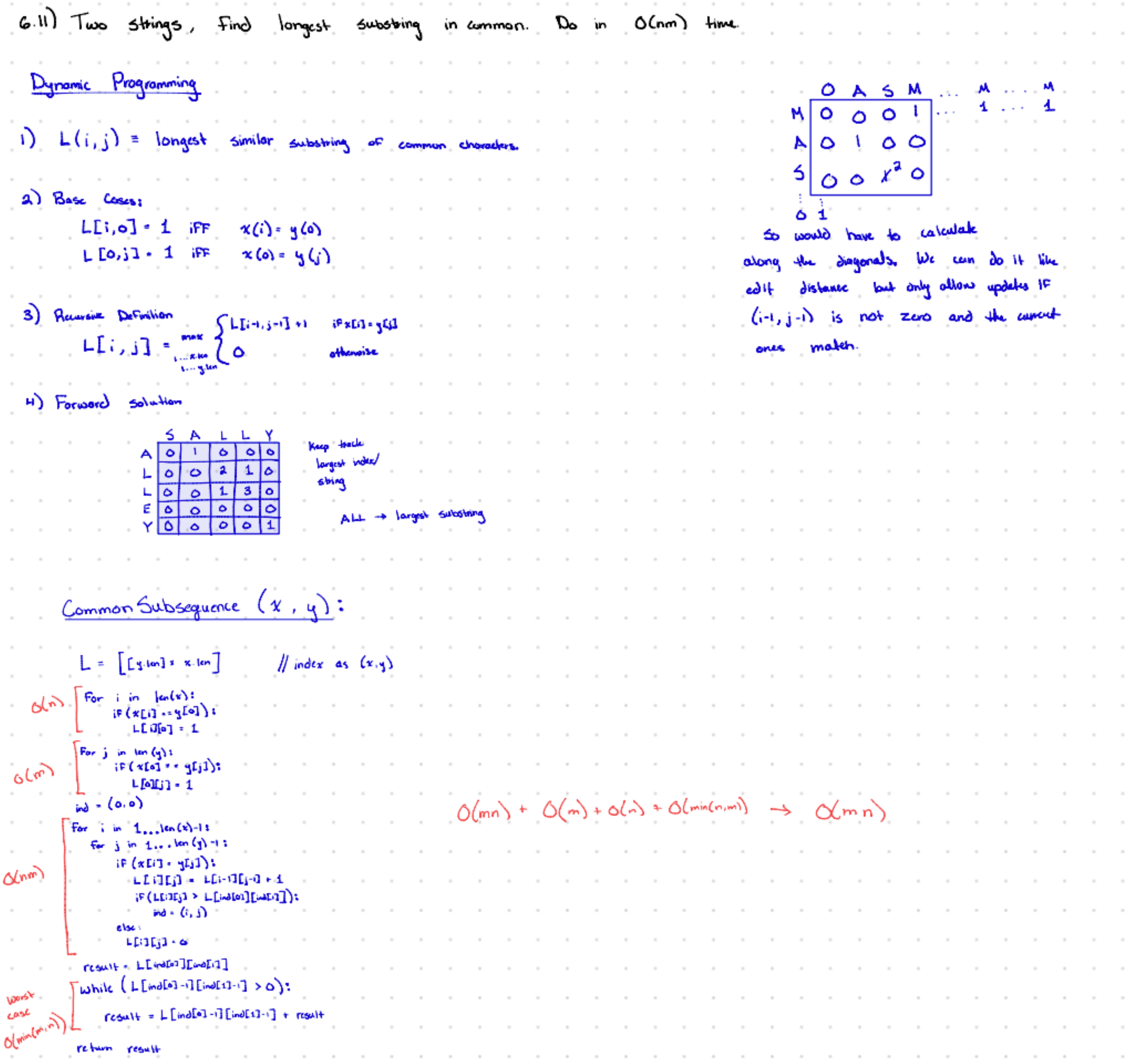
```
S.insert(".", 0); S.insert(".", -1)
L = [len(S)]        //Array of size S
W = []

prev = " "
curr = S[1]
L[0], L[-1] = False
L[1] = dict(S[1])

For i in (1, ..., len(S)-1):
    curr += S[i+1]
    if (dict(prev + curr)):
        prev += curr
        curr = " "
        W[-1] = prev

    else if (dict(curr)):
        prev = curr
        curr = " "
        w.append(prev)
if L[-2] == True:
    print(w)
    return True
return False
```

O(1) but for average
O(n) worst case
O(n) iterations

Avg case: O(1) × O(n) = O(n)

Worst Case: O(n) × O(n) = $O(n^2)$

Figure 2: Page 2



6.11) Two strings, Find longest substring in common. Do in O(nm) time.

Dynamic Programming

1) $L(i,j)$ = longest similar substring of common characters.

2) Base Cases:

$L[i,0] = 1$ iFF $x(i) = y(0)$

$L[0,j] = 1$ iFF $x(0) = y(j)$

3) Recursive Definition

$$L[i,j] = \max_{\substack{1 \dots x.len \\ 1 \dots y.len}} \begin{cases} L[i-1,j-1]+1 & \text{if } x[i]=y[j] \\ 0 & \text{otherwise} \end{cases}$$

4) Forward solution

	S	A	L	L	Y
A	0	1	0	0	0
L	0	0	2	1	0
L	0	0	1	3	0
E	0	0	0	0	0
Y	0	0	0	0	1

Keep track largest index/ string

ALL → largest substring

	O	A	S	M	...	M	...	M
M	0	0	0	1	...	1	...	1
A	0	1	0	0				
S	0	0	$\cancel{2}$	0				
⋮	⋮							
0	1							

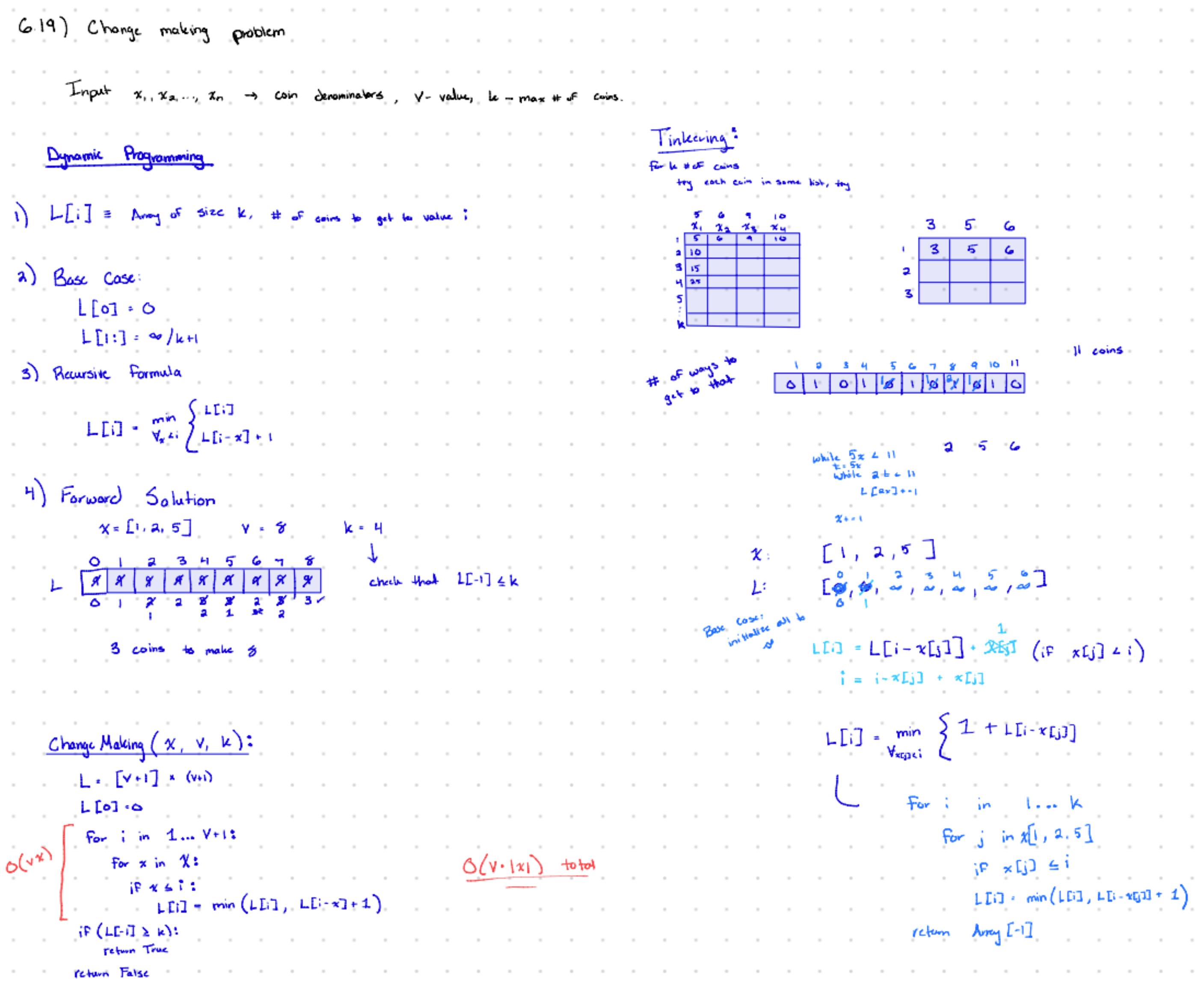
So would have to calculate along the diagonals. We can do it like edit distance but only allow updates IF (i-1, j-1) is not zero and the current ones match.

Common Subsequence (x, y):

```
        L = [[y.len] * x.len]        // index as (x,y)
O(n)    For i in len(x):
            iF (x[i] == y[0]):
                L[i][0] = 1
O(m)    For j in len(y):
            iF (x[0] == y[j]):
                L[0][j] = 1
        ind = (0,0)
O(nm)   For i in 1...len(x)-1:
            For j in 1...len(y)-1:
                iF (x[i] = y[j]):
                    L[i][j] = L[i-1][j-1] + 1
                    iF (L[i][j] > L[ind[0]][ind[1]]):
                        ind = (i, j)
                else:
                    L[i][j] = 0
        result = L[ind[0]][ind[1]]
worst   while (L[ind[0]-1][ind[1]-1] > 0):
case
O(min(m,n))  result = L[ind[0]-1][ind[1]-1] + result
        return result
```

$O(mn) + O(m) + O(n) + O(\min(n,m)) \rightarrow O(mn)$

Figure 3: Page 3



6.19) Change making problem

Input $x_1, x_2, \ldots, x_n$ → coin denominators , V- value, k – max # of coins.

Dynamic Programming

1) L[i] ≡ Array of size k, # of coins to get to value i

2) Base Case:

L[0] = 0

L[1:] = ∞ / k+1

3) Recursive Formula

$$L[i] = \min_{\forall_x < i} \begin{cases} L[i] \\ L[i-x]+1 \end{cases}$$

4) Forward Solution

x = [1, 2, 5] V = 8 k = 4

	0	1	2	3	4	5	6	7	8
L	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
	0	1	~~2~~ 1	2	~~3~~ 2	~~3~~ 1	2 ~~3~~	~~3~~ 2	3 ✓

↓ check that L[-1] ≤ k

3 coins to make 8

```
Change Making (x, v, k):
    L = [v+1] * (v+1)
    L[0] = 0
    For i in 1... V+1:
        For x in X:
            if x ≤ i:
                L[i] = min (L[i], L[i-x]+1)
    if (L[-1] ≥ k):
        return True
    return False
```

O(vx)

O(V·|x|) total

Tinkering:

for k # of coins

try each coin in some list, try

	5 x_1	6 x_2	9 x_3	10 x_4
1	5	6	9	10
2	10			
3	15			
4	25			
5 ⋮				
k				

	3	5	6
1	3	5	6
2			
3			

11 coins

# of ways to get to that

1	2	3	4	5	6	7	8	9	10	11
0	1	0	1	1 ~~0~~	1	1 ~~0~~	2 ~~1~~	1 ~~0~~	1	0

2 5 6

```
while 5x < 11
  t = 5x
  while 2t < 11
      L[2x] += 1

  x += 1
```

x: [1, 2, 5]

L: [~~0~~ 0, ~~∞~~ 1, ∞, ∞, ∞, ∞, ∞] (indices 0, 1, 2, 3, 4, 5, 6)

Base case: initialize all to ∞

$L[i] = L[i - x[j]] + \cancel{x[j]}\ 1$ (if $x[j] < i$)

$i = i - x[j] + x[j]$

$$L[i] = \min_{\forall x[j] < i} \begin{cases} 1 + L[i-x[j]] \\ \end{cases}$$

```
For i in 1... k
    For j in x[1, 2, 5]
        if x[j] ≤ i
        L[i] = min(L[i], L[i-x[j]] + 1)
    return Array[-1]
```

Figure 4: Page 3