CSE417_HW7_Yao-Chung Liang_1826630

2.

I used the convention of slides to index the OPT table which starts from the up-left position and ends on up-right place with the highest OPT value.

Algorithm	Runtime
Def OPT(start, end, seq):	
If start >= (end-4):	O(1)
Return 0	
Else:	
Value1=table[start][end-1]	O(1)
For t = (start) to (end-4):	O(n)
If seq[t] + seq[end] ==0:	O(1)
comparer += [1 + table[start][t-1] + table[t+1][end-1]]	O(1)
Else:	
comparer.append(0)	O(1)
Value2 = max (comparer)	O(n)
Return max(Value1, Value2)	

Algorithm	Runtime
Def traceback(start,end,pairs,OPT,sequence):	
If end<=start :	O(1)
return	
elif OPT[start][end] == OPT[start][end-1]:	O(1)
traceback(start,end-1,pairs,OPT,sequence)	T(n-1)
else:	
for k in [x for x in range(start,end-4) if check_pair(b,end)]	O(n)
if k-1 < 0:	O(1)
if OPT[start][end] == OPT[k+1][end-1] +1:	O(1)
pairs.append([k, end])	O(1)
traceback(k+1,end-1,pairs,OPT,sequence)	T(n-1)
break	
elif OPT[start][end] == OPT[start][k-1] + OPT[k+1][end-1] +1 :	O(1)
pairs.append([k,end])	O(1)
traceback(start, k-1, pairs, OPT, sequence)	T(n-1)
traceback(k+1, end-1, pairs, OPT, sequence)	T(n-1)
break	

5.

In my traceback algorithm,

First case is " if j<=i,   then return", which means in this case I've gone through the whole sequence and no need to do anything more.

Second case is "if OPT[i][j] == OPT[i][j-1] , then traceback(i,j-1)", which means if j is unpaired, then there will be no change in score when we take it out, so we just recurse to next index.

Third case is to try to pair j with a matching index k to the left. Thus, in this case, it will consider if there are some substructure or say sub pairs inside i and j. The implementation here is to check if the score at OPT[i][j] is the result of adding 1 from pairing (j,k) and whatever score comes from the substructure to its left OPT[i][k-1] and to its right OPT[k+1][j-1].

And I also consider some tricky problems like when k is 0, means it is in the leftist edge of the OPT table. In this situation, I also make it to only trace back to it's right OPT[k+1][j-1] when OPT[i][j] is equal to OPT[k+1][j-1]+1 which means there maybe a way back.

Thus, After I consider all these cases, I can find all possible pairs and pick up those may result in the next position in OPT table and use this way to trace back the most pairs.

6.

Time complexity:

For constructing OPT table:

In the beginning I'll loop through i from 0 to n and j from 0 to n which is O(n)*O(n) and put into OPT function to assign value to the certain position in the table which is (O(1)+O(1)+O(n)*(O(1)+O(1)+O(1))+O(n)).

Thus, after I multiply them, it's O(n^3)

O(n)*O(n)*(O(1)+O(1)+O(n)*(O(1)+O(1)+O(1))+O(n))=O(n^3)

For traceback:

In my trace back algorithm, between the first conditional statements, since "else" part is even worse than "if" part, so I picked this part as worst case part. And in for k in [x for x in range(start,end-4) if check_pair(b,end)], the worst situation is there are almost (n-4)/2 pairs inside, thus this line would be O(n) in worst situation. Under the "if k-1<0", the main parts is just another trackceback function which is T(n-1).

T(n) = O(n)+T(n-1)

T(n-1)=O(n-1)+T(n-2)

T(n-2)=O(n-2)+T(n-3)

…..

T(1)=O(1)+T(0)

=

0+1+2+3+….+n = n(n-1)/2

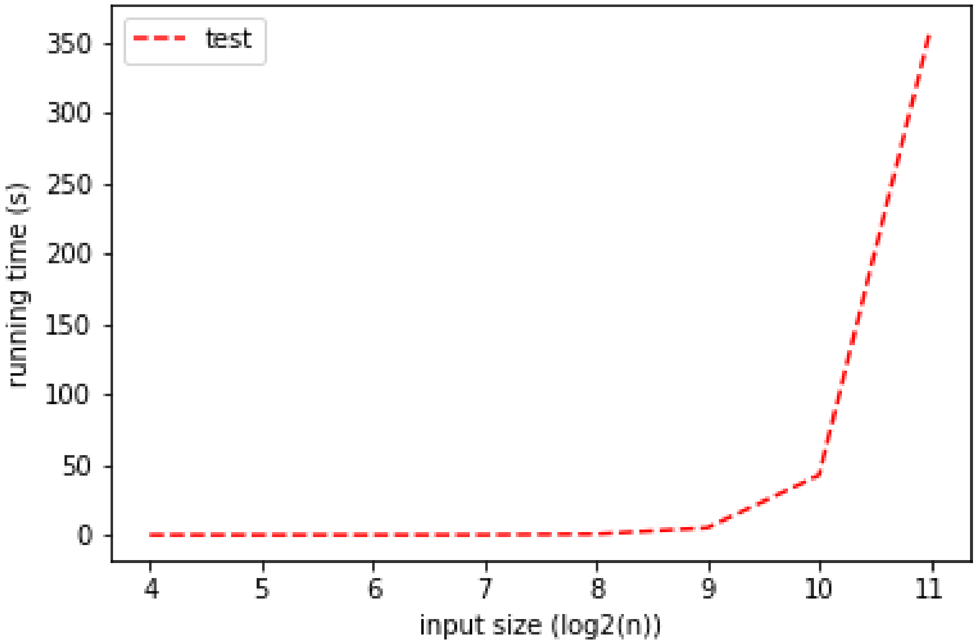
= O(n^2)

For total time complexity:

Because O(n^3) dominated O(n^2), the total time complexity is O(n^3).

7.

Random sample runtime analysis:



Actual runtime:

Input size n	Running Time (s)
16	0.0010884650000662077
32	0.0011580290001802496
64	0.009628757999962545
128	0.0764865280000322
256	0.6229025499999352
512	5.165956714999993
1024	42.50242388499987
2048	359.5303070289999
4096	3233.9193488320007

My theoretical performance is O(n^3) and my real runtimes are a little bit more than that, which is because there are O(n^2) and O(n) in O(n^3). So from the form, you can clearly see that after size =32, every time it doubles the input size, the running time will be a little bit more than 8 times than before.

And before size = 32, there is no big change I think is because in that small size, there are not much chance to have a subpair or make it go into the recursion.

And the last column for input size = 4096 is additional one so I didn't put into my graph together.