1. (20 pts) Complete the tables for dynamic programming for the Longest Common Sub-sequence (LCS) Problem on $X = CGATAC$ and $Y = ACGCTAC$.

   Let $c(i, j)$ represent the length of LCS(X_i, Y_j), where $X_i = (x_1, x_2, \ldots, x_i)$ contains the first i letters of X and $Y_j = (y_1, y_2, \ldots, y_j)$ contains the first j letters of Y.

   Fill in the missing values of the entries.

c	•	A	C	G	C	T	A	C
•	0	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1	1
G	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
T	0	1	1	2	2	3	3	3
A	0	1	1	2	2	3	4	4
C	0	1	2	2	3	3	4	5

In the following, each entry $b(i, j) \in \{\uparrow, \nwarrow, \leftarrow\}$ represents the direction of recursion for $c(i, j)$. Specifically,$b(i, j) = \nwarrow$ if $x_i = y_j$; otherwise, $b(i, j) = \uparrow$ if $c(i-1, j) \geq c(i, j-1)$; $b(i, j) = \leftarrow$ if $c(i-1, j) < c(i, j-1)$.

Fill in the missing values of the entries.

b	•	A	C	G	C	T	A	C
•	0	0	0	0	0	0	0	0
C	0	↑	↖	←	↖	←	←	↖
G	0	↑	↑	↖	←	←	←	←
A	0	↖	↑	↑	↑	↑	↖	←
T	0	↑	↑	↑	↑	↖	←	←
A	0	↖	↑	↑	↑	↑	↖	←
C	0	↑	↖	↑	↖	↑	↑	↖

What is the LCS of X and Y? C G T A C

2. (20 pts) Complete the tables for dynamic programming for the Knapsack Problem with $C = 12$, on items $\{(0.74, 6), (0.84, 5), (0.37, 4), (0.57, 3), (0.25, 2), (0.10, 1)\}$, where each item i is represented by (value v_i, size s_i).

Let $f(i, b)$ represent the optimal objective on items $\{1, 2, \ldots, i\}$ and with capacity b. We have the following recursion for computing the value of $f(i, b)$ (suppose $b \geq s_i$):

$$f(i, b) = \max\{f(i-1, \underline{b - s_i}) + \underline{v_i}, f(i-1, \underline{b})\}.$$

Fill in the missing values of the entries.

f	1	2	3	4	5	6	7	8	9	10	11	12
1 (0.74, 6)	0.00	0.00	0.00	0.00	0.00	0.74	0.74	0.74	0.74	0.74	0.74	0.74
2 (0.84, 5)	0.00	0.00	0.00	0.00	0.84	0.84	0.84	0.84	0.84	0.84	1.58	1.58
3 (0.37, 4)	0.00	0.00	0.00	0.37	0.84	0.84	0.84	0.84	1.21	1.21	1.58	1.58
4 (0.57, 3)	0	0	0.57	0.57	0.84	0.84	0.84	1.41	1.41	1.41	1.58	1.78
5 (0.25, 2)	0	0.25	0.57	0.57	0.84	0.84	1.09	1.41	1.41	1.66	1.66	1.78
6 (0.10, 1)	0.1	0.25	0.57	0.57	0.84	0.94	1.09	1.41	1.51	1.66	1.76	1.78

In the following, each entry $d(i, b) \in \{Y, N\}$ represents the direction of recursion for $f(i, b)$. Specifically, $d(i, b) = Y$ if $f(i, b)$ is obtained by taking item i into the knapsack. Fill in the missing values of the entries.

d	1	2	3	4	5	6	7	8	9	10	11	12
1 (0.74, 6)	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y
2 (0.84, 5)	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
3 (0.37, 4)	N	N	N	Y	N	N	N	N	Y	Y	N	N
4 (0.57, 3)	N	N	Y	Y	N	N	Y	Y	Y	Y	N	Y
5 (0.25, 2)	N	Y	N	N	N	N	Y	N	N	Y	Y	N
6 (0.10, 1)	Y	N	N	N	N	Y	N	N	Y	N	Y	N

What is the optimal solution (set of items) of the problem instance? 1.78

What is the solution returned by the Greedy algorithm that

- picks items with maximum value first:
- picks items with minimum size first:
- picks items with maximum density first:

(1) Maximum value first

(0.84, 5), (0.74, 6), (0.1, 1) → 1.68

(2) minimum size first

(0.10, 1) (0.25, 2) (0.57, 3) (0.37, 4) → 1.29

(3) maximum density

$\frac{0.74}{6} = 0.123$ $\frac{0.84}{5} = 0.168$ $\frac{0.37}{4} = 0.0925$

$\frac{0.57}{3} = 0.19$ $\frac{0.5}{2} = 0.25$ $\frac{0.1}{1} = 0.1$

(0.37,3) (0.84,5) (0.25,2) (0.1,1)

→ 1.76

3. (10 pts) Given n intervals $\{I_1, I_2, \ldots, I_n\}$, where each interval $I_i = (s_i, f_i)$ has start time s_i and finish time f_i, both of which are integers and $f_i > s_i > 0$. The intervals are sorted in ascending order of their finish times, i.e., we have $f_1 \leq f_2 \leq \ldots \leq f_n$. Additionally, each interval is associated with a value $v_i \geq 0$. The objective of the problem is to pick a subset of non-overlapping intervals with maximum total value.

   - (5 pts) For each interval i, let $k(i) \leq i - 1$ be the maximum index such that $f_{k(i)} \leq s_i$. If $f_1 > s_i$ then we define $k(i) = 0$.
     Show that we can compute $k(1), k(2), \ldots, k(n)$ in $O(n \log n)$ total time.

We use binary search to find the lower bound f_j of each s_i and we will get $k(i) = j$

Each binary search is $O(\log n)$

To computer $k(1)$ to $k(n)$ is in $O(n \log n)$

- (5 pts) Let $A[i]$ be the value of the optimal solution of the problem with intervals $\{1, 2, \ldots, i\}$, where $i \in \{0, 1, \ldots, n\}$. In other words, $A[i]$ is the maximum total value of a subset of non-overlapping intervals in $\{I_1, \ldots, I_i\}$. Derive a recursive formula for $A[i]$, e.g., express $A[i]$ using $A[1], \ldots, A[i-1]$ and v_i, and use it to give an $O(n)$ time algorithm to compute the value of the optimal solution.
  You can assume that you already have the values $k(1), k(2), \ldots, k(n)$.

$A[0] = 0$

$A[i] = A[k(i)] + v_i$

Algorithm:
max = -1
for i in 1 to n:
A[i] = A[k[i]] + v[i]
if A[i] > max:
max = A[i]

output = max

4. (10 pts) Compute a table representing the failure function in the Knuth-Morris-Pratt (KMP) algorithm, for the pattern string "CGTCGCGTCGTAC".

   Recall that given a string P with m characters, the failure function f is defined as follows: for all $i \in \{0, 1, \ldots, m-1\}$, $f(i) \in \{0, 1 \ldots, i\}$ is the length of the longest prefix (excluding $P[0, i]$ itself) of $P[0, i]$ that is also a suffix of $P[0, i]$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12
P[i]	C	G	T	C	G	C	G	T	C	G	T	A	C
f(i)	0	0	0	1	2	1	2	3	4	5	3	0	1

5. (10 pts) Let T be a text of length n, and let P be a pattern of length m. Design an algorithm for finding the longest prefix of P that is a sub-string of T and analyze its running time. You get 5 pts if your algorithm runs in $O(nm)$ time; 8 pts if it runs in $O(n \log m)$ time; 10 pts if it runs in $O(n + m)$ time.

```
#include <iostream>
#include <vector>
using namespace std;
void computeLPSArray(string &p,int m,vector<int>& lps){
        int len = 0;
        int i = 1;
        while( i < m){
                if(p[i] == p[len]){
                        lps[i] = len + 1;
                        len++;
                        i++;
                }else{
                        if( len != 0)
                        len = lps[len-1];
                        else{
                                lps[i] = 0;
                                i++;
                        }
                }
        }
}
string solution(string &t,string &p){
        int n = t.length();
        int m = p.length();
        vector<int> lps(m,0);
        computeLPSArray(p,m,lps);
        int i = 0;
        int j = 0;
```

O(m)

```
        while (i < n)
        {
                if(t[i] == p[j]){
                        i++;
                        j++;
                }else{
                        if( j != 0)
                        j = lps[j-1];
                        else
                        i++;
                }
                if(j == m)
                break;
        }
        return p.substr(0,j+1);
}
int main(){
        string T = "onions";
        string P = "onion";
        cout<<solution(T,P);
        return 0;
}
```

O(m)

O(n+m) → whole