Introduction to Algorithms: Second Assignment

Liu Chengjun, Student ID: 0490303961

**Problem 1:**

For Algorithm A, present general pseudo-code:

|  |
| --- |
| Function AlgorithmA()------  -If  -Solve ------  -Return solved  -Else  -Divide  into  till  with size  -For  from 1 to 5  -Solve  to get ------  -Combine from to ------ (linear time)  -Return solved |

The time of Algorithm A follows: 

By Master Theorem,  is linear,  as  where .

So  between  and .

The running time is estimated to be .

For Algorithm B, present general pseudo-code:

|  |
| --- |
| Function AlgorithmB()------  -If  -Solve ------  -Return solved  -Else  -Divide  into ,  with size  -Solve  to get ------  -Solve  to get ------  -Combine and ------ (constant time)  -Return combined |

The time of Algorithm B follows: 

So 

So the running time is estimated to be exponential .

For Algorithm C, present pseudo-code:

|  |
| --- |
| Function AlgorithmC()------  -If  -Solve ------  -Return  -Else  -Divide C into  till  with size  -For  from 1 to 9  -Solve  subdivision to get  subdivision------  -Combine from  to ------ (quadratic time)  -Return solved |

The time of Algorithm C follows: 

By Master Theorem,  is quadratic,  as  where .

So  between  and .

The running time is estimated to be .

**Problem 2:**

Theoretical discussion:

By Greedy Algorithm, the global optimization is reached based on optimization of local circumstances.

For already selected customer sequence  and remained customer sequence  , to choose one from remained customer sequence and bind it with already selected customer sequence to optimize new selected customer sequence of size  , the optimized solution is to choose customer of minimum time from remained customer sequence, which follows the Greedy Algorithm.

So in global environment, it actually performs the same as sorting method of selection sorting by choosing the minimum, place it one by one and record the corresponding customer number.

That means by using fast sorting method and moving customer numbers the same as movements of sorted time, the algorithm can reach optimization.

|  |
| --- |
| Pseudo-code:  Given two vectors of customer numbers and spending time corresponding to each other:  and  Function MinWaitingTime(,)  -If  - Return  %Only one in C(n)  -Else  - %seeking reference at the end of array S(n)  - %counting from in, tracking the final place that reference x should be  -For  from 1 to  -If  %if finding the spending time smaller than x  -  -Swap  with  %swap it to the left  - Swap  with  %moving the corresponding index together  -Swap  with  %insert the reference  -Swap  with  %insert the index of reference together  - MinWaitingTime(,) %split and sort lower half  - MinWaitingTime(,) %sort upper half  Return |

The pseudo-code above simulates Quick Sort by binding the movement of customer number and spending time.

Because the movements of elements (,) are doubled compared with Quick Sort only moving , the running time is estimated to be  .

The average waiting time of customers will be the minimum of  , where sequence  should be in ascending order. The reason why it is minimum is given by recursion from dual inequality  , further proving it applies to multiple elements.

A more basic way simulating finding the minimum one by one like Greedy Algorithm is as follow:

|  |
| --- |
| Function MinWaitingTime(,)  Order=Array(n)  RemainSn=  RemainCn=C(n)  for i from 1 to n:  Order[i]=RemainSn(FindMinimum(RemainSn)) %finding the index of minimum in the remained array  RemainSn=RemainSn(1:Order[i]-1)+RemainSn(Order[i]+1:n) %keep remaining array  RemainCn=RemainCn(1:Order[i]-1)+RemainCn(Order[i]+1:n) %keep remaining array of index  Return Order  Function j=FindMinimum(RemainSn)  j=1  Minimum=RemainSn(1) %starting point to compare  For i from 2 to n:  if RemainSn(i)<Minimum %if smaller than reference  Minimum=RemainSn(i) %make the compared the minimum  j=I %record the index of minimum  Return j %return the index of minimum |

Obviously the running time of the algorithm is 1+2+…+n=

**Problem 3:**

The Algorithm for finding consecutive LCS is similar to finding LCS.

By running all combinations of comparisons of substrings the same as finding LCS( situations, that is ), the algorithm can be completed.

To find the length of LCS in every sub-situation, the length should follow such recursive rule:



A new rule is that we regard each combination  as a vector of two elements and  . Recursive rule changes the element of a vector and inherit element. When , reset  to be 1 in the vector . When, let .

For example, applying the sequences given in assignment:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **A** | **G** | **C** | **T** | **A** |
| **A** | **1,1** | **1,1** | **1,1** | **1,1** | **1,1** |
| **C** | **1,1** | **1,1** | **1,1** | **1,1** | **1,1** |
| **T** | **1,1** | **1,1** | **1,1** | **2,2** | **2,2** |
| **A** | **1,1** | **1,1** | **1,1** | **2,2** | **3,3** |
| **T** | **1,1** | **1,1** | **1,1** | **2,2** | **3,3** |
| **G** | **1,1** | **1,1** | **1,1** | **2,2** | **3,3** |

**The pseudo-code of algorithm is presented as follow:**

|  |
| --- |
| **Function ConsecutiveLCS() %two string array of X of m size and Y of n size**  **%recording the LCS still counting the length**  **%recording the maximum LCS up to now**  **For  from 2 to**  **For  from 2 to**  **If  %if X=Y and consecutive with the extending LCS in the front**  **%adding the counting length of maximum**  **%copying the maximum from i-1, j-1**  **If  %if the counting LCS exceeds the maximum record**  **%renew the maximum record**  **Else if  %if X=Y but disconnected to previous LCS**  **%restart counting at 1**  **%usually as 1 when starting to count first LCS**  **If  &  %if the maximum on i-1,j is larger**  **%copy the record**  **Else if  &  %if the maximum on I,j-1 is larger**  **%copy the record**  **Else if  %X unequal to Y, the counting breaks.**  **%restart counting**  **%copying maximum record**  **Else**  **%restart counting**  **%copying maximum record**  **Return** |

**The running time is roughly , fitting the requirement.**