Foundation of Algorithm

Priyanka Dwivedi
Srikant Lakshminarayan

March 8, 2018

Problem - 1

1. The recurrence relation for step ii is
   $T(n) \leq O(1)$ for n = 1
   $T(n) \leq T(n-\alpha) + T(\alpha) + O(n)$ for $n > 1$
   Best Case when α = n/2
   Worst Case when α =1

2. Base Case Complexity is O(nlogn)
   Worst Case Complexity is O(n^2)

3. Yes, It is optimal.The right sub list will have a size which will be less or at most equal to the size of the left sublist. In this algorithm, the size of left sublist becomes smaller and smaller and the right sublist will always have a larger number Suppose there are only 3 or 4 values in the list(Base case). After partition, the left sublist will always have the least two numbers which will have the similar prefix code(same length for the last and 2nd last element starting from right to left). Since the numbers are always being sorted in ascending order, the smaller numbers will be at the start and the larger numbers at the end. Hence the difference will keep getting smaller and the left sublist will have smaller numbers so their prefix code will be of greater length than the right sublist.After partition, the number of elements in left sublist will have to be more or equal than right sublist. i.e the elements in right list will never be more than the left one and as the numbers are getting smaller the optimality will also keep decreasing.

Problem - 2

Algorithm Used : Select K algorithm to find median.

Algorithm:

1. Take co-ordinates of police from user.
2. Find median of x co-ordinate and y co-ordinates.
3. Median for both x and y co-ordinates will form the best co-ordinate to open donut store.Select k algorithm is used for the approach.
   Select k algorithm:

   (a) Take n elements from user.
   (b) Randomly select any value from numbers
   (c) Iterate through the values and partition in three buckets Less,Equal and Greater
      i. Less : This bucket will have all values less than pivot.
      ii. Equal : This bucket will have all values equal to pivot.
      iii. Greater : This bucket will have all values greater than pivot.
   (d) Compare searching index 'k' as follows:
      i. if k is less than length(Less) then element is within Less Bucket and call select with Less and k.
      ii. if k is greater length(Less) and k is less length(Less)+length(Equal) then we have found the element i.e pivot which the median.
      iii. else element is in greater bucket and select only greater bucket as data and new k value i.e k-length(Less)-1
   (e) The result of Select function is median of the values provided.

4. Calculate the sum of distances from each police co-ordinates by calculating summation of $x_{best} - x_i + y_{best} - y_i$ where i is co-ordinates of police from 1 to n and n is number of police. **Correctness:**
   We want to find best location to open donut store where the sum of distance that traffic police have to travel should be minimum. In order to get best location we must place the store in center so that sum of distance is minimum. The algorithm finds the median of x and y co-ordinates which will gives the best location to open donut store.

Data Structure Used:
Python Inbuilt List : Complexity and Description of operation performed:

1. Access element at particular index : O(1)
2. Append Element in list:O(1)
3. Length of list len(n) : O(1)

Overall Complexity of Algorithm

1. Append police co-ordinates from user O(n)

2. Calculate median of x and y co-ordinates using select k algorithm which forms the best co-ordinates O(n)
3. Calculate sum of distance from best co-ordinates O(n)
4. Overall complexity for algorithm **O(n)**

Problem - 3

Algorithm Used : Sorting with scheduling using two pointer method.

Algorithm:

1. Take jobs and there start, finish and employee number from user.
2. Pick jobs from employee 0 and 1 and store in array.
3. Sort the array with the finish time.
4. Consecutively schedule jobs for employee 0 and 1 by using the finish time of job 1 and start time of job 2 such that they don't conflict with each other.
5. Continue scheduling jobs in alternate fashion and update counter as soon as a compatible job if found.

Data Structure Used:
Python Inbuilt List : Complexity and Description of operation performed:

(a) Access element at particular index : O(1)
(b) Length of list len(n) : O(1)
(c) Inbuilt python sort : O(nlogn)

Overall Complexity of Algorithm

(a) Take jobs from user O(n)
(b) Sort job for employee 0 and 1 O(nlogn)
(c) Schedule jobs alternatively O(n)
(d) Overall complexity for algorithm is O(nlogn)

Correctness :
To Solve the Problem we have used two pointer method. Initially we sort jobs with finish time and start scheduling jobs considering finish time of job 1 to be less than finish time of job 2. Since we are sorting both jobs we know we cannot have job lesser than that value.This will always result in optimal choice selection of job. Two pointer system gives complexity of O(n) but pre processing data by sorting gives complexity of O(nlogn) and over all complexity is O(nlogn).

Problem - 4

Algorithm Used : Dynamic Programming Approach

Heart of the Algorithm:

(a) Description of DP array : The DP array is 2D array i.e it has two components weight for bag 1 and bag 2. Representation - knapsack[W1][W2] where W1 and W2 are the weights of bag 1 and bag 2. Dp array stores the maximum cost that will be asssociated to fill the bag with weight w.

(b) Final Solution : k[W1][W2]

(c) Recurrence :

$$k[w1][w2] = \max \begin{cases} \text{w1 and w2} > \text{v:} & (k[w1][w2], k[w1-v][w2]+c, k[w1][w2-v]+c); \\ \text{w1} > \text{v:} & k[w1][w2], k[w1-v][w2]+c; \\ \text{w2} > \text{v:} & k[w1][w2], k[w1][w2-v]+c; \end{cases} \tag{1}$$

where v = weight of j^{th} item and c is cost of the j^{th} item and k is the knapsack dp array.

Data Structure Used:
Python Inbuilt List : Complexity and Description of operation performed:

(a) Access element at particular index : O(1)

(b) Length of list len(n) : O(1)

(c) To find maximum of n elements : O(n)

Overall Complexity of Algorithm

(a)	Take n items along with weight and cost	O(n)
(b)	Initialize DP array i.e knapsack with 0	O(W1W2)
(c)	Compute the cost for filling the bags	O(nW1W2)
(d)	Final solution	O(1)
(e)	Overall complexity	**O(nW1W2)**

Correctness
The algorithm takes n items and fill DP array by giving the maximum cost associated to fill the bag with that weight. For each item it considers the previous weight cost along with including or excluding the item so that the previous cost is also taken into consideration and we do not lose track of previous item taken into consideration.

Problem - 5 a

(a) Algorithm Used : Dynamic Approach.
Heart of the Algorithm:

i. Verbal Description of DP array: The Dp array is 1D array which stores the length of increasing subsequence for every given element 1,2...,k and a_k is part of last element of LIS.
ii. Solution: Maximum in the dparray.
iii. Recurrence : dparray = 1 + dparray[j] or include that element where j is the index that maximises as subject to j < k and $a_j < a_k$

Data Structure Used:
Python Inbuilt List : Complexity and Description of operation performed:

i. Access element at particular index : O(1)
ii. Length of list len(n) : O(1)
iii. To find maximum of n elements : O()

Overall Complexity of Algorithm

i.	Initialize dp array as 1 for sequence length 'n'	O(n)

ii. For every element fill dp array by checking whether any element less than that exist if yes what is the dparray value of that element. $O(n^2)$

iii. Overall complexity $O(n^2)$

Problem - 5 b

1. Algorithm Used : Recursive Approach.
   Algorithm :

   (a) Base case if current position is end of sequence return 0

   (b) Recursive Case:

   (c) Start from index 0 and find lis for considering that element and lis for not considering that element.

   (d) Maximum of both these will be returned and longest increasing sequence till jth element will be found

   Data Structure Used:
   Python Inbuilt List : Complexity and Description of operation performed:

   (a) Access element at particular index : O(1)

   (b) Length of list len(n) : O(1)

   (c) To find maximum of n elements : O(n)

   Overall Complexity of Algorithm

   (a) Recurrence Relation : $T(n) \leq \ 2T(n-1) + O(n)$

   (b) Overall Complexity : $O(n2^n)$

   Comparison between Dynamic and Recursive Approach:
   Note : Time taken are in seconds.

Input Size	Dynamic Approach	Recursive Approach
10	2.9087066650390625e-05	0.0001556873321533203
30	0.00011801719665527344	0.011667966842651367
60	0.00038886070251464844	2.0883491039276123
100	0.001071929931640625	64.04404520988464
120	0.002432107925415039	617.9677040576935
200	0.0035860538482666016	more than 10 mins
400	0.014435052871704102	more than 10 mins
800	0.053948163986206055	more than 10 mins
1200	0.13698720932006836	more than 10 mins
5000	2.2889530658721924	more than 10 mins
10000	8.47981309890747	more than 10 mins

Observation :

(a) Recursive Algorithm take more time as we increase size of input from 120

(b) Our algorithm could handle input size up to 120.

(c) Dynamic approach is fast compared to recursive approach.