Gebze Technical University
Department of Computer Engineering
CSE 321 Introduction to Algorithm Design
Fall 2020
Final Exam (Take-Home)
January 18th 2021-January 22nd 2021

Student ID and Name	Q1 (20)	Q2 (20)	Q3 (20)	Q4 (20)	Q5 (20)	Total
171044098/Akif Kartal						

Read the instructions below carefully

- You need to submit your exam paper to Moodle by January 22nd, 2021 at 23:55 pm as a single PDF file.

- You can submit your paper in any form you like. You may opt to use separate papers for your solutions. If this is the case, then you need to merge the exam paper I submitted and your solutions to a single PDF file such that the exam paper I have given appears first. Your Python codes should be in a separate file. Submit everything as a single zip file. Please include your student ID, your name and your last name both in the name of your file and its contents.

Q1. Suppose that you are given an array of letters and you are asked to find a subarray with maximum length having the property that the subarray remains the same when read forward and backward. Design a dynamic programming algorithm for this problem. Provide the recursive formula of your algorithm and explain the formula. Provide also the pseudocode of your algorithm together with its explanation. Analyze the computational complexity of your algorithm as well. Implement your algorithm as a Python program. **(20 points)**

Q2. Let $A = (x_1, x_2, \ldots, x_n)$ be a list of n numbers, and let $[a_1, b_1], \ldots, [a_n, b_n]$ be n intervals with $1 \le a_i \le b_i \le n$, for all $1 \le i \le n$. Design a divide-and-conquer algorithm such that for every interval $[a_i, b_i]$, all values $m_i = \min\{x_j \mid a_i \le j \le b_i\}$ are simultaneously computed with an overall complexity of $O(n\log(n))$. Express your algorithm as pseudocode and explain your pseudocode. Analyze your algorithm, prove its correctness and its computational complexity. Implement your algorithm using Python. **(20 points)**

Q3. Suppose that you are on a road that is on a line and there are certain places where you can put advertisements and earn money. The possible locations for the ads are x_1, x_2, …, x_n. The length of the road is M kilometers. The money you earn for an ad at location x_i is $r_i > 0$. Your restriction is that you have to place your ads within a distance more than 4 kilometers from each other. Design a dynamic programming algorithm that makes the ad placement such that you maximize your total money earned. Provide the recursive formula of your algorithm and explain the formula. Provide also the pseudocode of your algorithm together with its explanation. Analyze the computational complexity of your algorithm as well. Implement your algorithm as a Python program. **(20 points)**

Q4. A group of people and a group of jobs is given as input. Any person can be assigned any job and a certain cost value is associated with this assignment, for instance depending on the duration of time that the pertinent person finishes the pertinent job. This cost hinges upon the person-job assignment. Propose a polynomial-time algorithm that assigns exactly one person to each job such that the maximum cost among the assignments (not the total cost!) is minimized. Describe your algorithm using pseudocode and implement it using Python. Analyze the best case, worst case, and average-case performance of the running time of your algorithm. **(20 points)**

Q5. Unlike our definition of inversion in class, consider the case where an inversion is a pair $i < j$ such that $x_i > 2\,x_j$ in a given list of numbers $x_1, \ldots, x_n$. Design a divide and conquer algorithm with complexity $O(n \log n)$ and finds the total number of inversions in this case. Express your algorithm as pseudocode and explain your pseudocode. Analyze your algorithm, prove its correctness and its computational complexity. Implement your algorithm using Python. **(20 points)**

171044098
Akif Kartal

CSE 321 Final Exam
Solutions

Akif.

Q1) In this question we need to design a dynamic programming algorithm. Therefore I will apply following step for this question.

=> Dynamic Programming Steps

1) Identify optimal substructure.
2) Find a recursive formulation for the value of the optimal solution.
3) Use dynamic programming to find the value of the optimal solution
4) If needed, keep track of some additional info so that the algorithm from step 3 can find the actual solution.
5) If needed, code this algorithm. (we need this)

If we apply these steps;

Step 1) What is the optimal structure?

In this problem, the idea is very simple. Just compare last character of the array arr[i:j] with its first character. There are two case here.

1) If last character of the array is same as the first character include it in subarray and continue.
2) If they are different return maximum of two values.

Step 2 Find a recursive formula.

By using our optimal structure from step 1 we can easily find following recurrence relation.

$$dp[i][j] = \begin{cases} 0, & (\text{if } i,j == n) \\ dp[i-1][j-1]+1, & (\text{if } arr[i] == rvsArr[j]) \\ \max(dp[i-1][j],\ dp[i][j-1]), & \text{if}(arr[i] \,!= rvsArr[j]) \end{cases}$$

Step 3 Use dynamic programming to find optimal solution.

If we use our recursive formula in our algorithm we will get length of maximum subarray which have the property in the question.

Inputs of algorithm

arr = array of letter
rvsArr = reverse of array (to make comparision) (design choice)
n = length of array
subArr = solution array in recurrence relation. (dp)

```
procedure maxLenght(arr[i:n], rvsArr[i:n], n, subArr[i:n])
    for i = 1 to n do
        for j = 1 to n do
            if arr[i-1] == rvsArr[j-1]
                subArr[i][j] = subArr[i-1][j-1] +1
            else
                subArr[i][j] = max(subArr[i-1][j], subArr[i][j-1])
            end if
        end for
    end for
    return subArr[n][n]
end
```

Actually, I explained this algorithm in step 1.

There is an optimization here which is by using memoization technique we don't compute same subproblems again and again.

Analysing of Algorithm

As you can see from psuedocode algorithm have two nested loop like in selection sort Thus

$\boxed{T(n) \in \Theta(n^2)}$

Note that in this algorithm we only find length of subarray. I will print this subarray in real code python since it is not relevant the question.

Also, in this algorithm I am finding any subarray which have the property such that letters don't have to be one after the other (successive) which mean if there is a subarray have property in array in straight order I will find lenght of it by using this algorithm.

For example;

Let arr = [a, b, b, d, c, a, c, b]

The subarray which have property will be:

subArr = [b, c, a, c, b] with lenght of 5.

As you see sub array contain elements such that some of them is not successive in original arrray but have the property.

SOLUTION OF Q2

Q2) In order to solve this question, if we observe the question we will see that we have "n" interval and to solve question with overall complexity of $O(n\log n)$ we need to find minimum of each interval $O(\log n)$ time without destroy total complexity.

1) How can we find minumum of an array in $O(\log n)$?

This question leads us to binary search trees and if we make a deep search on this question we will find a tree called "segment tree".

Now, by using segment tree structure we can easily find min(a,b) in $O(\log n)$ because segment tree has $O(\log n)$ levels and we move one level higher at each step by using divide and conquer methodology.

2) Second question is What will be the cost of constracting the tree? Will it damage total cost?

Answer is easy, creating segment tree will take $O(n)$ time in total. I will explain this in pseudocode.

No, it will not damage total cost.

Now, totaly we have;

$$\Rightarrow T(n) \in \underbrace{O(n)}_{\substack{\text{create segment} \\ \text{tree}}} + \underbrace{O(n\log n)}_{\substack{\text{find each min value} \\ \text{for } n \text{ interval}}}$$

$$\Rightarrow \boxed{T(n) \in O(n\log n)}$$

pseudocode

```
procedure createSegmentTree(tree[1:2n], list[1:n], n)
    for i=1 to n do
        tree[n+i] = list[i]
    end for
    for i=n-1 to 0 do
        if (tree[2*i] < tree[2*i+1])
            tree[i] = tree[2*i]
        else
            tree[i] = tree[2*i+1]
        end if
    end for
end
procedure find-min(tree, a, b, n)
    a = a+n
    b = b+n
    min = ∞
    while (a < b)
        if (a % 2 == 1)
            min = min(min, tree[a])
            a = a+1
        end if
        if (b % 2 == 1)
            b = b-1
            min = min(min, tree[b])
        end if
        a = a/2
        b = b/2
end    return min
```

Pseudocode of main algorithm

```
procude find-all-min (list[1:n], n)
        tree = [2*n] // tree array
        createSegmentTree (tree, list, n) → O(n)
n ← for i 0 to n
            a = random(0, n)
            b = random(a, n)                              } → O(nlogn)
logn ←      print("min:", find-min (tree, a, b, n))
        end for
end
```

Analysis of the algorithm

As you can see the from code creating segment tree take $O(n)$ time. Also, finding minumum minumum on this tree takes $O(\log n)$ time by power of divide and conquer methodology. Because segment tree has $O(\log n)$ levels and we move one higher level at each step.

Total we have;

$$\Rightarrow T(n) \in O(n) + O(n\log n)$$

$$\Rightarrow \boxed{T(n) \in O(n\log n)}$$

Scanned with CamScanner

Proof of logn complexity while finding minimum

if we observe find-min algorithm we will see that in while loop each iteration we are dividing condition elements by two. Thus our recurrence relation is this from algorithm;

$$T(n) = T(n/2) + 1$$

Note = +1 comes from constant operations such as return.

if we solve this recurrence by using master theorem:

$$f(n) = \theta(1) \Rightarrow \theta(n^0 \log^0 n) \Rightarrow k = 0$$

$$a = 1, \quad b = 2 \Rightarrow \log_b a = \log_2 1 = 0$$

$$\log_b a = k \Rightarrow 0 = 0 \quad \text{Thus,}$$

$$T(n) \in \theta(n^k \log^{p+1} n)$$

$$T(n) \in \theta(n^0 \log^{0+1} n)$$

$$\boxed{T(n) \in \theta(\log n)}$$

SOLUTION OF Q3

Q3) If we observe this question, we see that this question is similar to the knapsack problem which is the one of the most popular problem in dynamic programming. Thus;
In order to solve this question with dynamic programming we need to start with finding recursive formula of algorithm.

Recursive formula

When we observe the question we have two choices at each step; one of them is put an ads or don't put an ads at x_i.
If we put an ads we will ignore the ads in previous 4 miles, and add the money of the ads.

if we don't put an ads we will ignore this ads. By using these information our relation will be;

Let $dp[i]$ be the optimal money that earned then,

$$dp[i] = \begin{cases} dp[i-1] \\ \max(dp[i-1], money[i]) \\ \max(dp[i-1], dp[i-5]+money[i]) \\ dp[i-1] + money[i] \end{cases}$$

We have 4 cases in recursive formula.

Before writing the algorithm we need to know these;

In this question we have;

M = total kilometer for road

$X[1:n]$ = list of possible location for ads.

money$[1:n]$ = list of money for ads in x_i place.

n = number of ads locations.

We will use this inputs to solve the problem.
These inputs comes from question.
Next, by using our recursion formula we will implement our algorithm.

```
procedure max_money(M, x[1:n], Money[1:n], n)

    dp = [M+1] //solution array
    nextAds = 0 // index for next advertisement
    for i = 1 to M+1
        if (nextAds >= n)
            dp[i] = dp[i-1]
        else
            if (i != x[nextAds])
                dp[i] = dp[i-1]
            else
                if (4 >= i)
                    dp[i] = max(dp[i], money[nextAds])
                else if
                    dp[i] = max(dp[i-1], dp[i-5] +
                                          money[nextAds])
                else
                    dp[i] = dp[i-1] + money[nextAds])
                end if
                nextAds = nextAds + 1
            end if
        end if
    end for
end
```

Analysis of the algorithm

As you can see from pseudocode in algorithm we have only one loop and in this loop by using previous recursive formula we are solving the problem. Thus, time complexity is;

$$\boxed{T(n) \in O(M)} \text{ (linear time)}$$

SOLUTION OF Q4

If we observe this question, we will see that this question is NP-Hard problem.

In order to solve this question we can use a linear programming algorithm which is a special case of mathmetical programming. But because of time constraint I tried to find minimitied maximum cost but I couldn't do this.

Next, you will see my midterm solution for this question.

Note that my algorithm procudes results such that very close to answer. Even, most of the time correct answer.

Q4) In this question since we have 2 array (people and Jobs) and also we have costs hinges upon this person-Job assignment for this costs we need to costs Matrix as an input with arrays.

For instance;

people

Job	0	1	2
0	10	5	12
1	40	80	1
2	30	15	22

⟹ Costs for person-Job assignment

$n \times n \Rightarrow$ n by n matrix

Also we have;

people — n

Job — n

(n is person and Job size)

Question says minimized maximum cost while assigning job to people.

To solve this question, my Algorithm is;

1) Check costs for each Job, if all maximum costs has same person then Just assing the Jobs in any order in this case number of maximum cost is will be 1.

2) if maximum costs are distrubuted on people then assing Jobs to people such that number of maximum cost will be 0 (zero).

Pseudocode

```
procedure Job-assingment(People, Jobs, cost_matrix, n)
    // First find maximum cost indexes for each row in matrix.
        max-cost-indexes[]
O(1) ←  max-index = -1
        all_max_has-same-person = true
      for i = 0 to n do
          max = -1
          for j = 0 to n do
              if max < cost[i][j]
                  max = cost[i][j]
                  max_index = j
              end if
O(n²) ←   end for
          max_cost_indexes[i] = max-index
          if i != 0 and max-cost-indexes[i-1] != max_index
                  all-max-has-same-person = false
          end if
      end for
      if all-max-has-same-person
          // number of max cost is 1
          // assing jobs in any orde
          for i = 0 to n do
O(n) ←        Jobs[i] = i
              people[i] = i
          end for
      else
O(n³) ←       assing-Jobs(People, Jobs, max-cost-indexes, n)
      end if
end
```

Scanned with CamScanner

pseudocode cont'd

```
procedure assing-Jobs(People, Jobs, max-cost-indexes, n)
    for i = 0 to n do
        if i == n-1 // if last index
O(n^3) <-   O(n^2) <- Jobs[i] = find-person(Job, n, max-cost-indexes[i])
        else
            is-found = false
            for k = i to n do
                if (max-cost-indexes[k] != max-cost-indexes[i])
                O(1) <- if not is-found
O(n^3) <- O(n^2) <- O(n) <- O(n) <- if not Jobs.contain(max-cost-indexes[k])
                            Jobs[i] = max-cost-indexes[k]
                    O(1) <- is-found = true
                        end if
                    end if
                end if
            end for
            if not is-found
O(n^3) <-   O(n^2) <- Jobs[i] = find-person(Job, n, max-cost-indexes[i])
            end if
        end if
        O(1) <- People[Jobs[i]] = i
    end for
end
```

⇒ Note that number of maximum cost is O(zero) in this case of algorithm.

Total time = $T(n) \in \Theta(n^3)$ (for this function)

~~pse~~udocode cont'd

```
procedure find-person(Jobs, n, current-index)

O(1)   <- person = -1
          for i=0 to n do
O(n^2) <- O(n) <- if (not Jobs.contain(i)) and (i != current-index)
                          person = i
                  end if
          end for
O(1)   <- return person
end
```

Note = contain method works $O(n)$ time since it makes search on array.

Total time $\Rightarrow T(n) \in \Theta(n^2)$ (for this function)

General Analysis

Best Case

If same person has all maximum costs we will have $\boxed{T(n) \in \Theta(n^2)}$ (to find maximum cost indexes in matrix. # of max cost is 1.)

Worst Case

If maximum costs are distributed on people we will have $\boxed{T(n) \in \Theta(n^3)}$ time to assing Jobs on people with minimited maximum cost such that # of maximum cost is O(zero).

Scanned with CamScanner

Average case

In average case, since the possibility of same person has all maximum cost is very low, we will have $\boxed{T(n) \in \Theta(n^3)}$ running time.

SOLUTION OF Q5

Q5) In this question to find number of the total number of inversions I will use following divide and conquer algorithm.

Divide: seperate list into two piece

Conquer: recursively count inversions in each half

Combine: count inversions where x_i and x_j are in different halves, and return sum of these three step.

As you can see this process actually very similar to the merge sort algorithm.

Next, we will apply this algoritm on pseudocode.

```
procedure getTotalInversion(L[1:n])
    if list L has one element
        return 0

    divide the list into two halves A and B
     r_A <- getTotalInversion(A) --> T(n/2)
     r_B <- getTotalInversion(B) --> T(n/2)
     r <- combine(A,B) --> n
     return r_A + r_B + r
end
```

Note that in this algorithm while we sorting the list according to new condition which is $x_i > 2x_j$ we are also finding the total number of inversions according to this condition.

Next, I will write combine algorithm of combining sorted two halves of list.

```
Procedure combine(L[1:n], Extra[1:n], low, mid, high)
    k = low , i = low , j = mid + 1 , inversion = 0
    while(i <= mid and j <= high)
        if(L[i] > 2*(L[j])) // x_i > 2x_j
            inversion = inversion + 1
            Extra[k] = L[j]
            j = j + 1
        else
            Extra[k] = L[i]
            i = i + 1
        end if
        k = k + 1
    end while
    while(i <= mid)
        Extra[k] = L[i]
        k = k + 1
        i = i + 1
    end while
    for i = low to high + 1 do
        L[i] = Extra[i]
    end for
    return inversion
end
```

Analysis of Algorithm

This algorithm is;

1) Divide and conquer algorithm
2) Recursive
3) Stable
4) Not in-place
5) $O(n\log n)$ time complexity.
6) $O(n)$ space complexity.

Proof of Time complexity

From the pseudocode we have;

$$T(n) \in \begin{cases} 1, & \text{if } n=1 \\ 2T(n/2)+n, & \text{if } n>1 \end{cases}$$

By using backward substution method;

$$T(n) = 2\{2T(n/2)+\frac{n}{2}\}+n$$
$$= 4T(n/4)+2n$$
$$= 4\{2T(n/8)+\frac{n}{4}\}+2n$$
$$= 8T(n/8)+3n$$
$$\vdots \quad \vdots \quad \vdots$$
$$= 2^k T(n/2^k)+kn$$

cont'd

$$\frac{n}{2^k} = 1 = 2^k = n \quad , \quad k = \log_2 n$$

$$= 2^{\log_2 n} T(1) + \log_2 n \cdot n$$

$$= n + n.\log n$$

$$= \boxed{\Theta(n\log n)}$$

Scanned with CamScanner