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
171044077 Mustafa Tokgöz						

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)**

Mustafa Tolgöz 171046077

Answer of Question 1)

pseudocode of algorithm

```
procedure algorithm (str):
    var n= length of str
        if n==0:
            return ""
        end if
    var answer=""
    var dp [n][n] two dimensional array with
        all elemet in it 0.
        max=0
        for j from 0 to n:
            for i from 0 to j+1:
                check=0
                if str[i]==str[j]:
                    check=1
                end if
                if (j-i)>2:
                    dp[i][j]=dp[i+1][j-1]
                        and check
                end if
                else:
                    dp[i][j]=check
                end else
                if dp[i][j]==1 and
                (j-i+1)>max:
```

```
        Max = j - i + 1
        answer = str[i:j+1]
    end if
  end for
end for
  return answer
end procedure
```

Recursive formula of this algorithm =

algorithm = Longest Palindrome

$$\text{Longest Palindrome}(i,j) = \begin{cases} 1 & \text{if } (str[i \text{ to } j] \text{ is palindrome} \\ 0 & \text{if } (str[i \text{ to } j] \text{ is not palindrome} \end{cases}$$

Longest Palindrome(i,j) = Longest Palindrome(i+1, j-1) and str[i] == str[j]

Base Cases = Longest Palindrome(i,j) = 1

= Longest Palindrome(i, i+1) = 1 if (str[i] is equal to s[i+1])

= Longest Palindrome(i, i+1) = 0 if (str[i] is not equal to s[i+1]).

if str[i] is equal to str[j] and dp array[i+1][j-1] is 1 then its longest palindrome.

Computational Complexity = $O(n^2)$ because of nested loop. The others are $O(1)$ time that does not effect the computational complexity.

Analyze and Explanation of Algorithm=

For this question, Firstly I explain my. Firstly I check the string array is empty or not. If it is then it returns empty string. If it is not an empty string, then I assign the empty string to answer variable. After that I create a two dimensional array. I assign 0 to max variable. After that nested loop begin. Outer loop is looping n times from i=0. Inner loop is looping j+1 times from i=0. In inner loop I assign 0 to check variable. After that if statement is checking the ith element of string array is equal to jth element of string array or not. If it is equal then check variable will be 1. After that if (j-i) is bigger than 2, then dp[i][j] will be assigned dp[i+1][j-1] and check. If (j-i) is not bigger than 2, then dp[i][j] will be assigned check variable. Because of bigger than 2 for base cases and for 5 or higher length of str. After that if dp[i][j] is 1 and (j-i+1) is bigger than max variable, then max variable is j-i+1 and answer is string array i to j+1 element. After that the algorithm returns the answer variable. In this algorithm 1 is refers to true in check variable. and dp array,

0 refers to false in check variable and dp array. I first coded the one and two letters palindromes, then finding all palindromes. Dp array is for checking the palindrome length is longer or not. Secondly I explain dynamic programming. Dynamic is an algorithm technique for soluding an optimization problem by breaking it down into simpler subproblems. In this algorithm I use dynamic programming with array as dp. This algorithm solved by using dynamic programming technique.

Tabe for input "MUSTAFA"

	M	U	S	T	A	F	A
M	1						
U	0	1					
S	0	0	1				
T	0	0	0	1			
A	0	0	0	0	1		
F	0	0	0	0	0	1	
A	0	0	0	0	1	0	1

longest palindrome is "AFA"

Answer of Question 2)

pseudo code of algorithm

```
global var checklist[100][100]

class Interval:
    constructor(left, right):
        this.L = left
        this.R = right
    end constructor
end class

procedure fill_the_checklist(arr, n):
    for i from 0 to n
        checklist[i][0] = i
    end for
    J = 1
    while (2^J <= n):
        i = 0
        while (i + 2^J - 1) < n:
            if (arr[checklist[i][J-1]] <
                arr[checklist[i + 2^(J-1)]
                [J-1]]):
                checklist[i][J] = checklist
                [i][J-1]
            end if
            else:
                checklist[i][J] =
                    checklist[i + 2^(J-1)][J-1]
            end else
```

```
        i = i + 1
    end while
    J = J + 1
end while
end procedure.

procedure findMin(arr, L, R):
    J = log base 2 (R - L + 1)
    if (arr[checklist[L][J]] <= arr[checklist[R - 2^J + 1]
                                    ][J]:
        return arr[checklist[L][J]]
    end if
    else:
        return arr[checklist[R - 2^J + 1][J]]

end procedure

procedure findMinimums(arr, n, I, m):
    fill_the_check(arr, n)
    for i from 0 to m:
        L = I[i].L
        R = I[i].R
        print("Minimum of interval [L, R] is
              , findMin(arr, L, R)).
    end for
end procedure.
```

procedure main:

A= [2,4,5,3,6,5,9,2]

n=length of A

I =[Interval(0,3), Interval(3,5), Interval(5,∞)]

m=length of I

findMinimums(A,n,I,m)

end procedure

Explanation and Analyze of this algorithm=

For this question, Firstly I construct a class that is Interval and this class has left an right variables. I create a global list table that is checklist. After that in fill_the_table method, I fill the checklist global variable with respect to that if checklist[i][j-1] th element of array is less or equal to array[i+2^{j-1}][j-1] then checklist[i][j] becomes checklist[i][j-1]. Else, checklist[i][j] becomes checklist[i+2^{j-1}][j-1]. This technique fills the checklist with respect to this formula. This fill is in while loop. After that, findmin method takes array and left and right variables. After that I set j variable as log base 2 (R-L+1). After that if arr[checklist[L][j]] is less or equal to arr[checklist(R-(2^j)+1)[j]] then it returns checklist[L][j] else it returns checklist[R-(2^j)+1][j]] then it returns checklist[L][j] else it returns checklist[R-2^j+1][j]. After that the beginning of the algoritm method is findMinimums method. This method takes array and

length of array and intervals and length of intervals. I call the fill_the_checklist method here. In this method there is a loop from 0 to length interval. In this loop, it divides the interval as left and right. After that it calls findMin method to find the minimums. This method is divide and conquer algorithm. The idea is to compute a minimum of subarrays of size 2^j from 0 to log n.

Computational Complexity:

$O(n \log n) + O(1)$ → FindMinimums for loop

↳ find min method because while loops takes n log n time.

$\boxed{O(n \log n)}$

Correctness:

Inputs:

A = [2, 4, 5, 3, 8, 5, 9, 2]

I = [Interval(0,3), Interval(3,5), Interval(5,8)]

Outputs:

[0,3] = [2, 4, 5, 3] => Minimum of interval [0,3] is 2

[3,5] = [3, 8, 5] => Minimum of interval [3,5] is 3

[5,7] = [5, 9, 2] => Minimum of interval [5,7] is 2

Answer of Question 3)

pseudocode of algorithm

```
procedure algorithm-question3(X, revenue, M, n):
    MaxR[M+1] array
    nextad = 0
    for i from 1 to M+1
        if nextad < n:
            if X[nextad] != i:
                MaxR[i] = MaxR[i-1]
            end if
        else:
            if i<=4:
                if MaxR[i-1] > revenue
                    [nextad]:
                    MaxR[i]=MaxR[i-1]
                end if
                else:
                    MaxR[i]=revenue
                        [nextad]
                end else
            end if
            else:
                if MaxR[i-5] +
            revenue[nextad] > MaxR[i-1]:
            MaxR[i] = Max[i-5] + revenue
                [nextad]
            end if
```

```
    else:
        MaxR[i] = MaxR[i-1]
    end else
    nextad = nextad + 1
end if
else:
    MaxR[i] = MaxR[i-1]
end else
return MaxR[M]
end procedure
```

Recursive Formula of this algorithm=

$$Max[i] = \text{Maximum of } [MaxR[i-1], MaxR[i-5] + revenue[i]]$$

ith element of MaxR array is maximum of MaxR[i-1] and MaxR[i-5] + revenue[i] because of that we want to find the maximum revenue. Why it adds MaxR[i-5] with revenue is there will be more than 4 Em. So 5 comes there.

Computational Complexity=

$O(n)$ because of for loop and the others O(1) time that does not affect the complexity. n is actually M. Because for loop is looping M+1 -1 = M times.

Explanation and Analyze of Algorithm=

For this question, Firstly Dynamic is an algorithm tekhnique for solving an optimization problem by breaking it down into simpler subproblems. In this algorithm, I use dynamic programming with list MaxR. This algorithm solved by using dynamic programming technique. Secondly I explain my algorithm to solve this question. Firstly I create an array as MaxR to be dynamic and to find the maximum. I assign 0 to nextad variable. After that in for loop, I check nextad is less than length of the X array or not X array stores the possible locations. Because it checks if there is a possible location or not. If it is then it nextad th element of X array is not equal to current i in for loop, then i th element of MaxR becomes i-1 th element of MaxR array. Else then if i is less or equal than restriction (that is 4) then if i-1 th element of MaxR is bigger than nextad th element of reverse then ith element of MaxR becomes i-1 th element of MaxR array, else ith element of MaxR becomes nextad th element of reverse array. Because we want to find the maximum reverse. So we check element is bigger or not. After that if i is bigger than restriction (that is 4) then i-5 th element of MaxR + nextad th element of reverse array is bigger than i-1 th element of MaxR array then ith element of MaxR array becomes i-5 th element of MaxR + nextad th element of reverse array, else ith element of MaxR becomes i-1 th element of MaxR array. Because of ignore the ad. Then nextad

variables increases 1. If in loop, nextad variable is bigger than length of array X then ith element of MaxR becomes $i-1$ th element of MaxR array. After that algorithm returns Mth element of MaxR array. M is length of the road.

Answer of Question 4)

pseudocode of my algorithm

```
procedure find_min_index (array):
    var min_index=0
    if array[0]==-1:
        min_index=1
    end if
    for i to length array
        if array[i] != -1 && array[i] <
                                array[min_index]
            min index =i
        end if
    end for
    return min_index
end procedure

procedure find_max_index (array):
    var max_index=0
    if array[0] == -1:
        max_index=1
    end if
    for i to length array
        if array[i] != -1 && array[i] >
                                array[max_index]
            max_index =i
        end if
    end for
    return max_index
end procedure
```

```
procedure clone (a):
    var new = deepcopy of a
    return new
end procedure

procedure algorithm (array):
        var count_max=0
        var arr2= clone(array)
        var result
        var last_result

    for i to length array:
            count_max2=count_max
            count_max=0
            if i==1:
                last_result= result
            end if
            for j to length array:
                min = find_min_index (arr2[j])
                max = find_max_index (arr2[j])
                if max==min:
                    count_max +=1
                end if
                result[j]= arr2[j][min]
                if i!=(length array)-1:
                    for k to length array:
                        arr2[k][min] =-1
                    end for
                end if
            end for
```

```
        if count_max <= count_max2:
            last_result = result

        end if
        arr2 = clone(array)
        mini = find-min-index(array[i])
        arr2[i][mini] = -1
        array[i][mini] = -1

    end for

    return last_result

end procedure
```

Best Case = $\Omega(n^3)$

Average Case = $\Theta(n^3)$

Worst Case = $O(n^3)$

Explanation and Analysis of my algorithm:

In my algorithm, I traverse array with for loop then keeping the maximum of the result and the inner loop, I find minimum index of that assignment and maximum index of the that assignment, then I check if the min is equal to max or not. If it is then I increase the count of max Because I need to know the number of the maximum in the result to choose minimum of maximum. After that I assign minimum number to result. Then I assign that column -1 not to choose that column. Then it goes inner for loop.

Then Outter loop I control the result has less maximum tha last result or not. Then if it is, I assigns. Then I turn the array original but assigns -1 to ith row minimum element And at the end of function, last result has the solution of the algorithm. After that I make it return last result. And the complexity of this algorithm is $O(n^3)$ because $O(1) * O(n) * (O(1) + O(1) + O(1)) + O(1) = O(n^3)$

$O(1)$: find_min_index, $O(1)$: find_max_index, $O(1)$: for loop, $O(1)$: clone, $O(n)$: for loop, $O(n)$: for loop

And the others $O(1)$ time that does not effects running time. The best case is $\Omega(n^3)$. The worst case is $O(n^3)$ and average case is $O(n^3)$.

Therefore the algorithm complexity $O(n^3)$

Answer of Question 5)

pseudocode of algorithm=

```
procedure InvCount(list, n):
    temp[n+1] array
    return InvCount2(list, temp, 0, n-1)
end procedure

procedure InvCount2(list, temp, left, right):
    count = 0
    if left < right:
        mid = (left + right)/2
        count = count + InvCount2(list, temp, left, mid)
        count = count + InvCount2(list, temp, mid+1, right)
        count = count + InvCount3(list, temp, left, mid,
            right)
    end if
    return count
end procedure

procedure InvCount3(list, temp, left, mid, right):
    i = left, j = mid+1, k = left, count = 0
    while i <= mid and j <= right:
        if list[i] <= 2 * list[j]:
            temp[k] = list[i]
            k = k+1
            i = i+1
        end if
```

```
    else:
        temp[k] = list[J]
        count = count + (mid - i + 1)
        k = k + 1
        J = J + 1
    end else
end while

    while i <= mid:
        temp[k] = list[i]
        k = k + 1
        i = i + 1
    end while

    while J <= right:
        temp[k] = list[J]
        k = k + 1
        J = J + 1
    end while

    for m from left to right + 1
        list[m] = temp[m]
    end for
end procedure

procedure main:
        list[]
        n length of list
        InvCount (list, n)
end procedure
```

Explanation and Analyze of algorithm:

For this question, I explain my algorithm. Firstly I create a temp array for not changing the original list when compare x[i] > 2x[j]. After that I call the InvCount2 method because of recursion. In InvCount2 method takes 4 parameter, 2 are list and 2 are indexes. Then return it. In InvCount2, after that I assign 0 to count variable. Then I check if left variable index is less than right like in mergesort. After that I assign (left+right)/2 to middle index like in merge sort. After that I set count variable as count + InvCount2 with left and mid variables to deuide the list and again set count + InvCount2 with mid+1 and right variables and then again set with count + InvCount3 with mid and right variables like mergesort but the only difference is count. Then InvCount2 returns the count. In InvCount3, i variable is left and j is mid+1 and k is left and count is 0. Then when i is small or equal than mid and j is small or equal than right, then if i.th element of list is small or equal than 2^j th element of list then there is no inversion and I assign ith element of list to kth element of temp list. Else there are an inversion and I set count variable as count+mid-i+1 because left and right sub list are sorted. Then merge the temp array, after that I set all elements of temp list to original list. After that InvCount3 returns count. Count variable is sum of the first half and second half. I actually change the merge sort to find count of inversion number. This algorithm is divide and conquer algorithm.

Computational Complexity=

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

this algorithm devides that length is $n/2$ and merge them $O(n)$ time. and assigns $O(n)$ time.

I use master teorem

$a = 2 \quad b = 2 \quad f(n) = n$

$a > 0 \quad b > 1 \quad d = 1$

if $a > b^d \Rightarrow$

$2 > 1^1 \Rightarrow T(n) = O(n^{\log_b a}) = \boxed{O(n \log n)}$

Correctness=

input= [1, 30, 6, 4]

output= Total number of inversion is 2

Because $30 > 2 * 6$ and $30 > 2 * 4$

result array = [1, 6, 4, 30]

input=[1, 30, 6, 4, 5]

output = Total number of inversion is 3

Because $30 > 2 * 6$ and $30 > 2 * 4$ and $30 > 2 * 5$

result array= [1, 6, 4, 5, 30]