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

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)**

1)

Pseudocode

```
function isPalRec(st[s:e]):
    if (s==e): then
        return True
    if (st[s] != st[e]): then
        return False
    if (s<e+1):
        return isPalRec(st, s+1, e-1)
    return True.
end-
```

call function $\frac{n}{2}$ times.

```
function isPalindrome(st[0:n])
    if(n==0);then
        return true
    else isPalRec(st[0:n-1])
end
```

call function n times.

$$Total = n \cdot \frac{n}{2} = \frac{n^2}{2} \text{ times.}$$

```
function dynamic_recurrence(liste, st, start, count):
    if (count > length_st) then
        return liste
    end if
    if (start + count <= length_st) then
        if (isPalindrome(st[start : start + count])) then
            liste.append(st[start : start + count])
    else:
        start = -1
        count = count + 1
    endif
    dynamic_recurrence(liste, st, start + 1, count)
end.
```

⇒ dynamic_recurrence(liste, st, 0, 2) ⟹ call function

(2) (n-2+1) ⇒ n-1
(3) (n-3+1) ⇒ n-2
(4) ⟶ n-3
(n) ⟶ 1

$\frac{n \cdot (n-1)}{2}$ times.

dynamic_recurrence function call isPolindrome function ~ $\frac{n.(n-1)}{2}$ times.

isPolindrome function call isPolRec function ~ n times

isPolindrome function call itself $\frac{n}{2}$ times.

Total $\Rightarrow \frac{n.n.n.(n-1)}{4} = n^4$ $O(n^4)$

Explonetion : Checks whether the arrays binary triple, quadruple... etc subarrays (from small to large) are polindrome. If is polindrome, it saves them in liste.

```
    if(start+count<=len(st)):
        print(st[start:start+count])
        if(isPalindrome(st[start:start+count])):
            liste.append(st[start:start+count])
    else:
        start=-1
        count=count+1
    dynamic_recuurence2(liste,st,start+1,count)



st="dcacd"
liste = []
dynamic_recuurence2(liste,st,0,2)
print(str(liste) + " are palindrome")
print(str(liste[-1]) + " is max polindrome")
```

```
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 1.py
dc
ca
ac
cd
dca
cac
acd
dcac
cacd
dcacd
['cac', 'dcacd'] are palindrome
dcacd is max polindrome
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$
```

2)

Pseudocode

```
function partition(arr, low, high):
    i = (low - 1)
    pivot = arr[high]
    for j = low to high : do
        if arr[j] <= pivot : then
            i = i + 1
            temp = arr[i]
            arr[i] = arr[j]
            arr[j] = temp
        end if
    end for
    temp = arr[i+1]
    arr[i+1] = arr[high]
    arr[high] = temp
    return (i+1)
end
function quickSort (arr, low, high):
    if low < high : then
        pi = partition (arr, low, high)
        quickSort (arr, low, high)
        quickSort (arr, pi+1, high)
    end if
end.
```

```
function solve( A[0:n], liste[0,n] )
    quickSort(A, 0, n-1)
    for i:=0 to n
        print ( A[liste[i][0]] )
    end for
end
```

Explanation: Quick sort function divides the function into two parts on the basis of the partition_value and makes two calls for different low and high value which divide the array into subarrays and partition function is used to sort the list A.

Partition function takes the high index value of the list A as the pivot over which sorting is done all the greater elements come after the pivot value and smaller before the pivot and it return (i+1) as result which is the final sorted position of the pivet value in the list

A

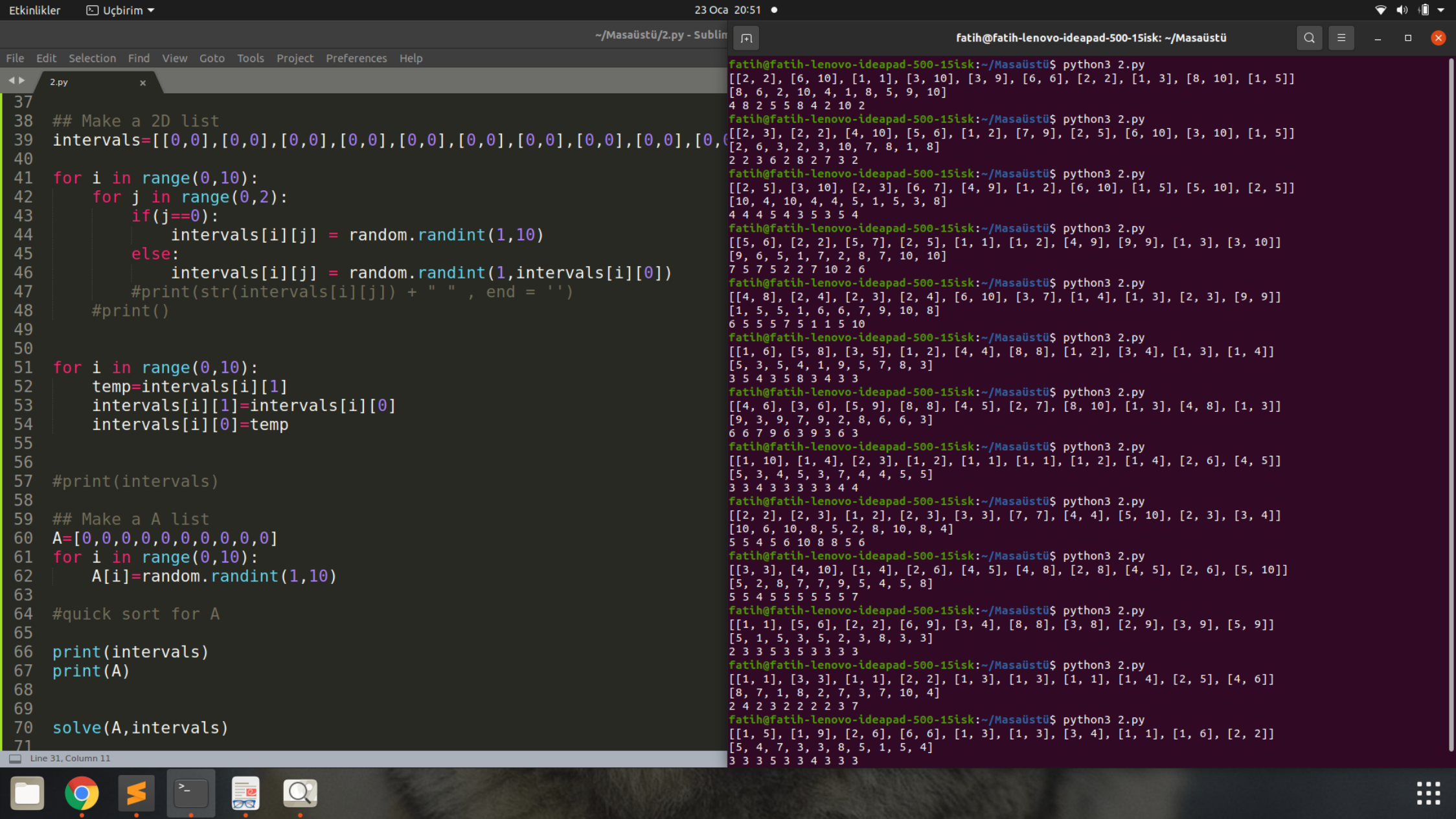
A will be sorted using quicksort algorithm and then traverse through the n intervals where the $a_i \leq j \leq b_i$, as the list is sorted then the a_i index value is the minimum value of the interval. (solve method)

- Time Complexity Quicksort function divides the list into subarray and make two calls for subproblems so it takes O(log(n)) time for dividing the list and each partition call sort the list and traverse element in linear time O(n) so the overall time complexity of sorting is O(nlogn).

Solve function calls QuickSort which has O(nlogn) complexity and traverse through the n intervals that has time complexity O(n).

Overall time complexity of the solution =

$O(n\log(n)) + O(n) = O(n\log(n))$ as greater complexity is taken.




```
## Make a 2D list
intervals=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,(

for i in range(0,10):
    for j in range(0,2):
        if(j==0):
            intervals[i][j] = random.randint(1,10)
        else:
            intervals[i][j] = random.randint(1,intervals[i][0])
        #print(str(intervals[i][j]) + " " , end = '')
    #print()


for i in range(0,10):
    temp=intervals[i][1]
    intervals[i][1]=intervals[i][0]
    intervals[i][0]=temp


#print(intervals)

## Make a A list
A=[0,0,0,0,0,0,0,0,0,0]
for i in range(0,10):
    A[i]=random.randint(1,10)

#quick sort for A

print(intervals)
print(A)


solve(A,intervals)
```

```
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[2, 2], [6, 10], [1, 1], [3, 10], [3, 9], [6, 6], [2, 2], [1, 3], [8, 10], [1, 5]]
[8, 6, 2, 10, 4, 1, 8, 5, 9, 10]
4 8 2 5 5 8 4 2 10 2
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[2, 3], [2, 2], [4, 10], [5, 6], [1, 2], [7, 9], [2, 5], [6, 10], [3, 10], [1, 5]]
[2, 6, 3, 2, 3, 10, 7, 8, 1, 8]
2 2 3 6 2 8 2 7 3 2
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[2, 5], [3, 10], [2, 3], [6, 7], [4, 9], [1, 2], [6, 10], [1, 5], [5, 10], [2, 5]]
[10, 4, 10, 4, 4, 5, 1, 5, 3, 8]
4 4 4 5 4 3 5 3 5 4
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[5, 6], [2, 2], [5, 7], [2, 5], [1, 1], [1, 2], [4, 9], [9, 9], [1, 3], [3, 10]]
[9, 6, 5, 1, 7, 2, 8, 7, 10, 10]
7 5 7 5 2 2 7 10 2 6
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[4, 8], [2, 4], [2, 3], [2, 4], [6, 10], [3, 7], [1, 4], [1, 3], [2, 3], [9, 9]]
[1, 5, 5, 1, 6, 6, 7, 9, 10, 8]
6 5 5 5 7 5 1 1 5 10
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[1, 6], [5, 8], [3, 5], [1, 2], [4, 4], [8, 8], [1, 2], [3, 4], [1, 3], [1, 4]]
[5, 3, 5, 4, 1, 9, 5, 7, 8, 3]
3 5 4 3 5 8 3 4 3 3
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[4, 6], [3, 6], [5, 9], [8, 8], [4, 5], [2, 7], [8, 10], [1, 3], [4, 8], [1, 3]]
[9, 3, 9, 7, 9, 2, 8, 6, 6, 3]
6 6 7 9 6 3 9 3 6 3
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[1, 10], [1, 4], [2, 3], [1, 2], [1, 1], [1, 1], [1, 2], [1, 4], [2, 6], [4, 5]]
[5, 3, 4, 5, 3, 7, 4, 4, 5, 5]
3 3 4 3 3 3 3 3 4 4
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[2, 2], [2, 3], [1, 2], [2, 3], [3, 3], [7, 7], [4, 4], [5, 10], [2, 3], [3, 4]]
[10, 6, 10, 8, 5, 2, 8, 10, 8, 4]
5 5 4 5 6 10 8 8 5 6
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[3, 3], [4, 10], [1, 4], [2, 6], [4, 5], [4, 8], [2, 8], [4, 5], [2, 6], [5, 10]]
[5, 2, 8, 7, 7, 9, 5, 4, 5, 8]
5 5 4 5 5 5 5 5 5 7
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[1, 1], [5, 6], [2, 2], [6, 9], [3, 4], [8, 8], [3, 8], [2, 9], [3, 9], [5, 9]]
[5, 1, 5, 3, 5, 2, 3, 8, 3, 3]
2 3 3 5 3 5 3 3 3 3
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[1, 1], [3, 3], [1, 1], [2, 2], [1, 3], [1, 3], [1, 1], [1, 4], [2, 5], [4, 6]]
[8, 7, 1, 8, 2, 7, 3, 7, 10, 4]
2 4 2 3 2 2 2 2 3 7
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 2.py
[[1, 5], [1, 9], [2, 6], [6, 6], [1, 3], [1, 3], [3, 4], [1, 1], [1, 6], [2, 2]]
[5, 4, 7, 3, 3, 8, 5, 1, 5, 4]
3 3 3 5 3 3 4 3 3 3
```

3) Pseudocode

M km ← distances ↑ money ↑ array size ↑ for check 4 km

```
function knapSack(W, wt, val, n, flag):
basecase => if (n==0 or W==0) then
              return 0
            end if
            if (wt[n-1] > W) then
              return knapSack(W, wt, val, n-1, flag)
            else:
              if (wt[n-1] - flag >= 4) then
                flag = wt[n-1]
                return max(val[n-1] + knapSack(W - wt[n-1],
                           wt, val, n-1, flag),
                           knapSack(W, wt, val, n-1, flag))
              else:
                return knapSack(W, wt, val, n-1, flag)
              end if
            end if
end
```

The distance between two advertisement places cannot be greater than total distance(M)

at least 4 kms from the previous advertisement / check value.

Computation Complexity : 2^n

Explanation: The condition is very similar to the knapsack problem, where the difference between two weights is at least 4, rather than crossing the total weight limit.

After writing the knapsack problem in a recurrence way, I assign the last received weight to the variable named flag.

When we subtract the weight of the new future from this flag, the result should be greater than 4. If this check happens, a new weight will be added. So the new advertising fee in our problem

wt => distances of ad places

val => money

~/Masaüstü/3.py - Sublin

File Edit Selection Find View Goto Tools Project Preferences Help

1.py | 3.py | untitled

```python
def knapSack(W, wt, val, n,flag):

    if n == 0 or W == 0:
        return 0


    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1,flag)
    else:
        print("wt  " + str(wt[n-1] )+ " flag " + str(flag))
        if(wt[n-1]-flag>4):
            flag=wt[n-1]
            return max(
                val[n-1] + knapSack(
                    W-wt[n-1], wt, val, n-1,flag),
                knapSack(W, wt, val, n-1,flag))
        else:
            return knapSack(W, wt, val, n-1,flag)




val = [2,5,100,12,120,62]
wt = [39,34, 29,22,17,12]
W = 99999999999
n = len(val)
flag=0
print (knapSack(W, wt, val, n,flag) )

```

Line 26, Column 9

fatih@fatih-lenovo-ideapad-500-15isk: ~/Masaüstü

```
wt  39 flag 34
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  22 flag 17
wt  29 flag 22
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  29 flag 22
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  17 flag 12
wt  22 flag 17
wt  29 flag 22
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  29 flag 22
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  22 flag 17
wt  29 flag 22
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  29 flag 22
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
wt  34 flag 29
wt  39 flag 34
wt  39 flag 34
301
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$
```

```python
def knapSack(W, wt, val, n,flag):

    if n == 0 or W == 0:
        return 0


    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1,flag)
    else:
        print("wt  " + str(wt[n-1] )+ " flag " + str(flag))
        if(wt[n-1]-flag>4):
            flag=wt[n-1]
            return max(
                val[n-1] + knapSack(
                    W-wt[n-1], wt, val, n-1,flag),
                knapSack(W, wt, val, n-1,flag))
        else:
            return knapSack(W, wt, val, n-1,flag)




val = [2,5,100,12,120,62]
wt = [32, 11, 10,6,5,1]
W = 99999999999
n = len(val)
flag=0
print (knapSack(W, wt, val, n,flag) )

```

```
prosedure removeAssign (arr[1:n][1:n], num)
    for J=0 to n do
        if (arr[1][j] ==num) then
            arr[1][j]=-1
        end if
    end for
end
prosedure JobPeople (Job[1:n], people[1:n])
    flap=1
    while flap!=0 do
        flap=0
        for i=0 to n do
            for j=0 to n do
                if (people[1][j] < job[0][i]) then
                    if (people[0][j] > Job[1][i]) then
                        removeAssign (Job, people[0][j])
                        removeAssign (paple, job[0][i])
                        flap=1
                    end if
                end if
            end for
        end for
    end while
end
```

Analysis of RemoveAssign

removeAssign similar to a linearSearch.

If num = arr[1][0], then the best case occurens.

$B(n) = 1 \in \Omega(1)$

Worst case if arr[1][n] = num or num does not exist in arr, then the worst case occurs.

$W(n) = n \in \Omega(n)$

$$A(n) = \sum_{i=1}^{W(n)} i \cdot p_i$$

$$= \left[\sum_{i=1}^{n-1} i \cdot \frac{p}{n} \right] + n\left(\frac{p}{n} + (1-p)\right)$$

$$= \frac{p}{n} \cdot \frac{(n-1)\cdot n}{2} + (p + n\cdot(1-p))$$

$$= \frac{p\cdot(n-1)}{2} + p + n - np$$

$$= \frac{pn}{2} - \frac{p}{2} + p + n - np$$

$$= \frac{p}{2} - \frac{pn}{2} + n = n\left(1 - \frac{p}{2}\right) + \frac{p}{2}$$

$$= n \in \Omega(n)$$

Analysis of JobPeople

$T_{if} \rightarrow B_{T_{if}}(n) = \Theta(1) \quad W_{T_{if}}(n) = A_{T_{if}}(n) = \Theta(n)$

$T_{For} \rightarrow B_{T_{For}}(n) = \Theta(1) \cdot \Theta(n^2) = \Theta(n^2)$

$W_{T_{For}}(n) = \Theta(n) \cdot \Theta(n^2) = \Theta(n^3)$

$A_{T_{For}}(n) = \Theta(n) \cdot \Theta(n^2) = \Theta(n^3)$

flop (while)

worst case

1 2 3 4

0 7 8 9

6, 7, 8, 9, -1

6, 7, 8, -1

6, 7, -1

6 -1

6

n times

flop(while) $B(n) = \Theta(1)$

$W(n) = \Theta(n)$

$A(n) = \Theta(n)$ → similar to linear seach.

so JobPeople ⇒ $B(n) = \Theta(1) \cdot \Theta(n^2) \cdot \Theta(1) = \Theta(n^2)$

$W(n) = \Theta(n) \cdot \Theta(n^2) \cdot \Theta(n) = \Theta(n^4)$

$A(n) = \Theta(n) \cdot \Theta(n^2) \cdot \Theta(n) = \Theta(n^4)$

```
jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, -1, 8, 17, -1, -1, 85, 11]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [-1, 0, 0, 89, 84, 0, 75, 74]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, -1, 8, 17, 7, -1, 85, 11]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [78, 0, 0, 89, 84, 0, 75, 74]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, -1, 8, 17, 11, -1, 85, -1]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [-1, 0, 0, 89, 84, 0, 78, 74]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, -1, 8, 17, 11, 7, 85, -1]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [18, 0, 0, 89, 84, 0, 78, 74]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, -1, 8, 17, 11, -1, 85, 7]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [75, 0, 0, 89, 84, 0, 78, 74]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, -1, -1, 17, 11, -1, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [-1, 0, 0, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [7, -1, -1, 17, 11, -1, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [15, 0, 0, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [7, 3, -1, 17, 11, -1, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [15, 13, 0, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [7, 5, -1, 17, 11, -1, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [15, -1, 13, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, 5, 7, 17, 11, -1, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [74, -1, 13, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, 5, 7, 17, 11, 3, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [74, 18, 13, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [-1, -1, 7, 17, 11, 5, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [74, -1, 18, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [3, -1, 7, 17, 11, 5, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [74, 15, 18, 89, 84, 0, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [4, -1, 7, 17, 11, 5, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [74, -1, 18, 89, 84, 15, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [4, 3, 7, 17, 11, 5, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [74, 13, 18, 89, 84, 15, 78, 75]]

jobs [[15, 13, 74, 84, 78, 18, 89, 75], [4, 3, 7, 17, 11, 5, 85, 8]]
people [[7, 3, 5, 85, 17, 4, 11, 8], [74, 13, 18, 89, 84, 15, 78, 75]]

jobs [15, 13, 74, 84, 78, 18, 89, 75]
people [4, 3, 7, 17, 11, 5, 85, 8]
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$
```

5) Pseudocode

```
function mergeSort(arr[0:n])
    temp_arr = [0]*n
    return _mergeSort(arr, temp_arr, 0, n-1)
end
function _mergeSort(arr, temp_arr, left, right)
    inv_count = 0
    if (left < right) then
        mid = (left+right) // 2
        inv_count = inv_count + _mergeSort(arr, temp_arr, left, mid)
        inv_count = inv_count + _mergeSort(arr, temp_arr, mid+1, right)
        inv_count = inv_count + merge(arr, temp_arr, left, mid, right)
    end_if
    return inv_count
end
function merge(arr, temp_arr, left, mid, right):
    i = left   j = mid+1   k = left   inv_count = 0
    while (i <= mid and j <= right) do
        if (arr[i] <= 2 * arr[j]) then
            temp_arr[k] = arr[i]
            k = k+1   i = i+1
        else:
            temp_arr[k] = arr[j]
            inv_count = inv_count + (mid - i + 1)
            k = k+1   j = j+1
        end if
    end_while
```

i < j
$X_i > 2X_j$

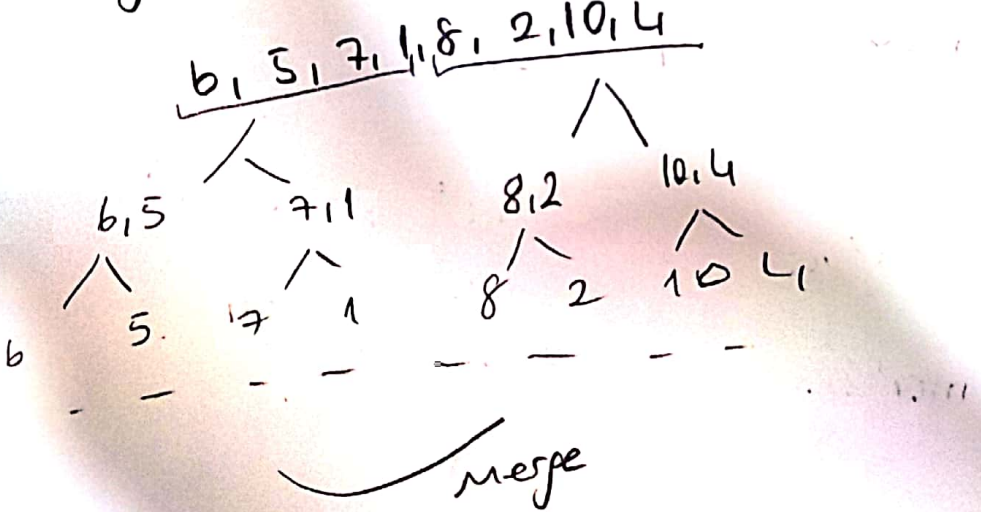
```
while (i <= mid) do
    temp_arr[k] = arr[i]
    k=k+1   i=i+1
end_while
while (j <= right) do
    temp_arr[k] = arr[j]
    k=k+1   j=j+1
end_while
for z = left to right+1 do
    arr[z] = temp_arr[z]
end_for
return inv_count
end
```

Explanation : The idea similar to merge sort, divide the array into two equal or almost equal halves in each step until the base case is reached.
Create a function merge that counts the number of inversions when two halves of the array are merged, create two indices i and j, i is the index for first half and j is an index of the second half. If a[i] is greater than 2 a[j], then there are (mid-i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray a[i+1], a[i+2], .. a[mid] will be greater than a[j].

Create a recursive function to divide the array into halves and find the answer by summing the number of inversions is the first half, number of inversion in the second half and the number of inversions by merging the two. The base case of recursion is when there is only one element in the given half.

Print the answer.

Calculation: The algorithm's main operation is merge sort. Since the inversion count calculation is inside the merge sort operations.



$C(n) = 2.C.(\frac{n}{2}) + C_{merge}(n)$

⇓

At each step, exactly 1 comparison is needed. In the worst case, neither of the 2 arrays become empty before the other one contains just 1 element. Hence

$C_{merge}(n) = n-1$ and we have $C_{worst}(n) = 2.C_{worst}(\frac{n}{2}) + n-1$

$C_{worst} = n.\log_2 n - n + 1$

$= n.\log n$

~/Masaüstü/5.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

5.py

```
            k += 1
            j += 1

        for loop_var in range(left, right + 1):
            arr[loop_var] = temp_arr[loop_var]




        return inv_count

arr = [1, 20, 6, 4,23, 5]
n = len(arr)
print("before " + str(arr))
result = mergeSort(arr, n)
print("Number of inversions are", result)
print("array " + str(arr))

```

Line 59, Column 26 Tab Size: 4 Python

fatih@fatih-lenovo-ideapad-500-15isk: ~/Masaüstü

```
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$ python3 5.py
before [1, 20, 6, 4, 23, 5]
ar [1, 20, 6, 4, 23, 5]
ar [1, 20, 6, 4, 23, 5]
ar [1, 20, 6, 4, 23, 5]
ar [1, 20, 6, 4, 23, 5]
ar [1, 6, 20, 4, 23, 5]
ar [1, 6, 20, 4, 23, 5]
ar [1, 6, 20, 4, 23, 5]
ar [1, 6, 20, 4, 23, 5]
ar [1, 6, 20, 4, 23, 5]
ar [1, 6, 20, 4, 5, 23]
ar [1, 6, 4, 5, 20, 23]
Number of inversions are 4
array [1, 6, 4, 5, 20, 23]
fatih@fatih-lenovo-ideapad-500-15isk:~/Masaüstü$
```