Name: Pourna Sengupta

ID: 109086577

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

Final/dont_panic.png

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Honor code: On my honor as a University of Colorado at Boulder student, I have neither given nor sought unauthorized assistance in this work

Initials PS

Date 7/25/2020

If you violate the CU **Honor Code**, you will receive a 0.

Name: Pourna Sengupta

ID: 109086577

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Pourna Sengupta

ID: 109086577

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

1. (5 pts) You are given a list of jobs with start and end times. Assume that only one job can be executed at a time on a processor. Your task is to find out the minimum number of processors required so that the jobs are executed as soon as they arrive (no waiting time for a process). The input to your algorithm will be a list of tuples indicating the start and end times of the respective jobs. The output should be a number indicating the minimum number of processors required. Your algorithm should have a runtime of $\mathcal{O}(n\log(n))$, where n is the number of jobs. Assume you do not have access to any auxiliary functions.

   Example
   Input: $(2, 10), (9, 11), (15, 18), (3, 4), (17, 19), (5, 13)$
   Output: 3
   Explanation: Between the interval $(9, 10)$ it can be seen that there will be 3 { (2, 10) , (9, 11), (5, 13)} jobs that need to be executed simultaneously. Thus at the very least 3 processors are required.

   (a) (4 pts) Write down well commented pseudo-code or paste real code to solve the above problem.
   ANSWER:

```
//define function
def maxOverlap(jobs):
    //define variables

    //max
    maxO = 0
    //count
    count = 0
    //temp array vector
    tempA[]

    //fill temp array vector with coordinates
    for i in range(len(jobs)):
        //add the all of the start times
        tempA.append([v[i][0], 'x'])
        //add all of the end times
        tempA.append([v[i][1], 'y])
```

```
//sirt the array
tempA = sorted(tempA)

//traverse the temp array vector
//count the number of overlaps
for j in range(len(tempA)):
    //if x occurs, a new range of s is identified
    //increase count by 1
    if(tempA[i][1] == 'x'):
        count++

    //if y occurs, the range has ended
    //decrease count by 1
    if(tempA[i][1] == 'y'):
        count--

    //update maxO for every loop
    //use max function to find that maximum
    maxO = max(maxO, count)

//output is maxO when loops have complete
print(maxO)
```

(b) (1 pts) Explain your algorithms runtime and space complexity by analyzing your code. For example, stating that a sorting algorithm runs in $\mathcal{O}(n \log(n))$ without any justification is insufficient.

ANSWER: Space Complexity: O(n)
The runtime of the maxOverlap function is ($O(nlog(n))$ because it functions like a merge sort function. The functions merges together the array of jobs into the temp array vector using a for loop with runtime $O(n)$. It then traverses the array using another for loop which has a runtime of $O(log(n)$. Together, this gives us the time complexity of $O(nlog(n))$.
The space complexity is O(n) because the variables used are arrays and array vectors which use 4n bytes each.

Name: Pourna Sengupta

ID: 109086577

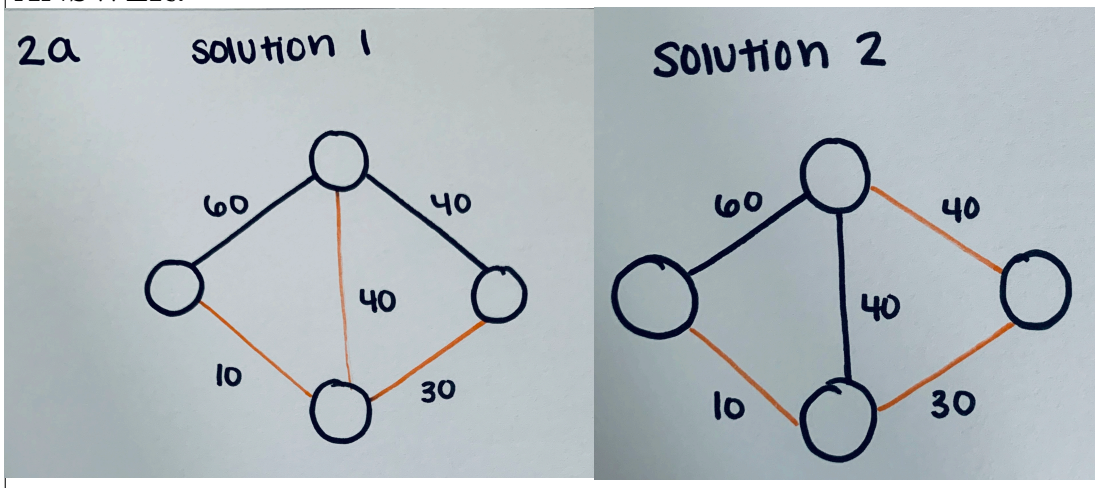
CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

2. (5 pts) You are designing a network, to connect several offices in different locations. You can contact your network cable provider about the cost of connecting any pair of offices. Your task is to design an algorithm, so that all offices are connected and the cost associated with building the network is minimized.
   The input to your algorithm is a Graph in the form of an adjacency list or matrix. The nodes in this graph represent the offices and edge weights represent the cost associated with connecting a pair of offices. The output of the algorithm will be a list of office pairs such that the total connection cost is minimized.

   (a) (2 pts) Provide an example with at least 3 offices and 3 connections. Your example must include at least 2 unique solutions.

ANSWER:



(b) (3 pts) Provide well commented pseudo code or actual code to solve the above problem.
ANSWER:

```
//helper functions
//find set helper function
int findSet(int node){
    //if node is its own parent
    if (node = parent[node]){
        //return the node
        return node;
    }
    //else, node isn't its own parent
    else{
        //the node is not representative of the set
        //recursively call find set
        //parent[node] as input
        //return value
        return findSet(parent[node]);
    }
}

//union helper function
void union(a, b){
    int x = findSet(a);
    int y = findSet(b);
    parent[x] = y;
}

//algorithm
//Using Kruskal's Algorithm
int kruskal(S[][V]){
    //S is the adjacency matrix
    //declare and initialize empty array
    //array will hold the updated MST
    int arr = [];
    //set each vertex as its own parent
    for(int i = 0; i < V; i++){
```

```
            parent[i] = i;
        }
        //set min weight edges added one by one
        //declare and initialize counter
        int edges = 0;

        //check for edges (a,b)
        //check if endpoint a and v belong in the same tree
        while(edges < V - 1){
            int min = INT_MAX;
            a = -1;
            b = -1;
            for(int m = 0; m < V; m++){
                for(int n = 0; n < V; n++){
                    //use find set helper function
                    //returns a representative element from set
                    //if findSet(a) = findSet(b)
                    //belong in the same tree
                    if(findSet(m) != findSet(n) && S[m][n] < min){
                        min = S[m][n];
                        a = m;
                        b = n;
                    }
                }
            }
            //update array
            //add edges to array
            arr[edges] = (a,b);
            //call union helper function
            //merge nodes from each tree
            union(a,b);
            //add to counter
            edges++
        }
        //return updated array
        return arr;
    }
```

Name: Pourna Sengupta

ID: 109086577

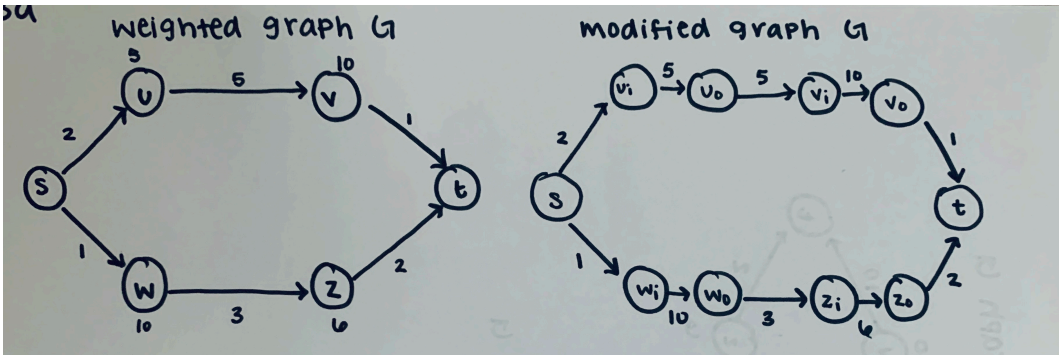
CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

3. (10 pts) Consider the graph $G = (V, E)$ with edge capacities $c(e)$, $\forall e \in E$, the edge capacity represents the maximum amount of flow that can pass through the edge. In addition to edge capacities, the above graph has vertex capacities $c(v)$, $\forall v \in V - \{s, t\}$ (The source and the sink vertex do not have vertex capacities). Each vertex capacity represents the maximum amount of flow that can pass through the vertex.

   (a) (4 pts) Given a source vertex $s \in V$ and a sink $t \in V$. How will you modify the given graph G, so that you can find the max-flow from s to t with a known algorithm? Also, show an example input and modified graph with at least 5 vertices. This must be a drawing!

The graph G can be modified by splitting every vertex v with a capacity into two vertices, v_i and v_o. We then add one edge (v_i, v_o) of the capacity c(v) and then convert each edge with the inflow and outflow. The inflow to v is (u, v), so the edges are converted to (u, v_i). The outflow to v is (v, w), so the edges are converted to (v_o, w). These modifications account for the vertex capacity and allow the problem to be treated as a maximum flow problem that can be solved using the Ford-Fulkerson algorithm.

ANSWER:

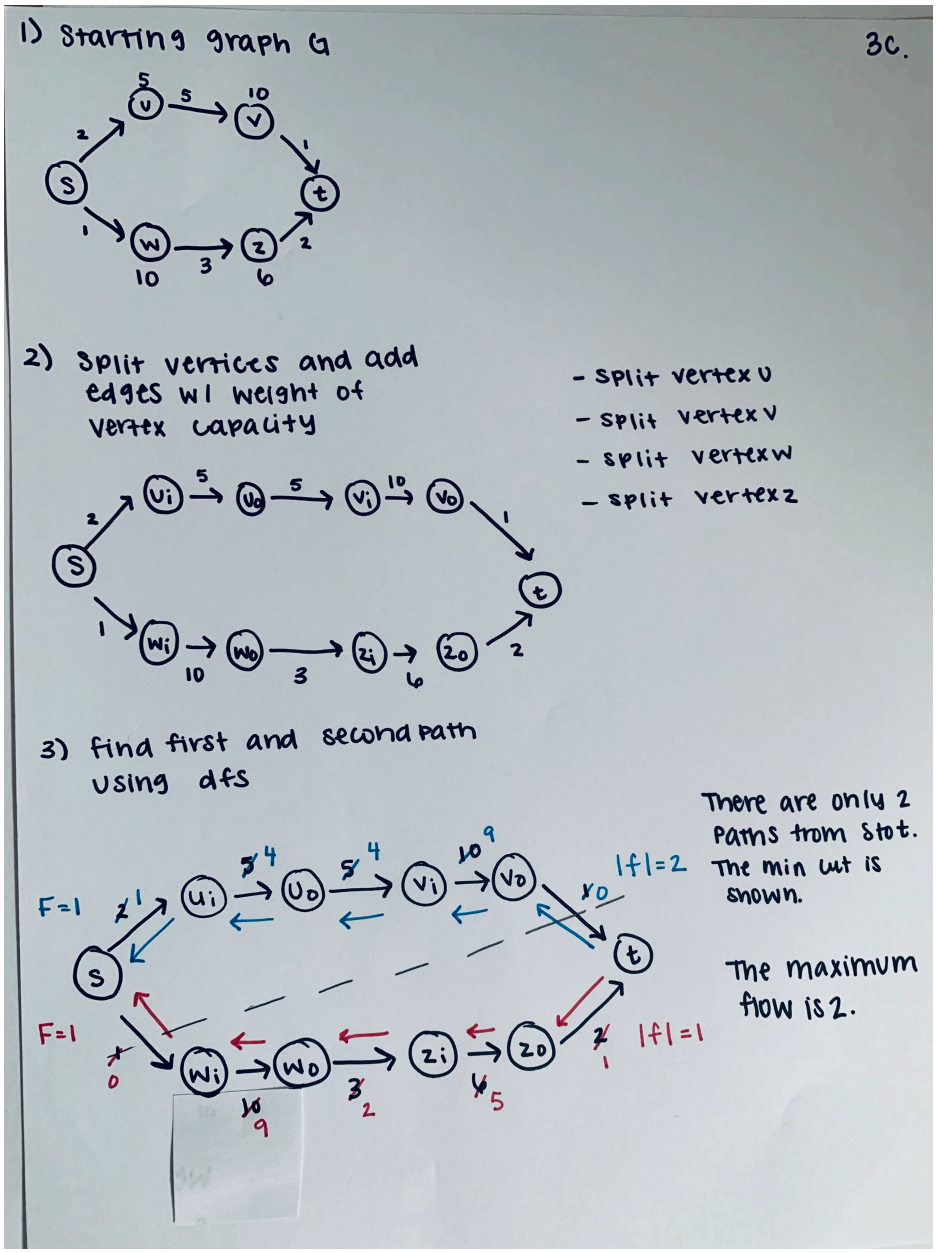


(b) (4 pts) Provide an algorithm to find the max-flow in the above graph from s to t.
ANSWER:

```
#Ford-Fulkerson algorithm
#finds max flow from s to t in graph G

def fordFulk(G, s, t):
    #split vertices with a capacity
    for every vertex v in G:
        define v_i
        define v_o
        #set capacity of edge (v_i, v_o)
        c(edge) = c(v)
        #convert inflow
        create edge (u, v_i)
        delete edge (u,v)
        #convert outflow
        create edge (v_o, w)
        delete edge (v, w)

        delete vertex v

    #set inflow and outflow to zero
    for each edge (u,v) in G:
        (u,v).f = 0

    #set maxflow
    maxFlow = 0
    #traverse using DFS to find path from s to t
    while there exists path P from s to t in G_f:
        c_f(P) = min{c_f(u,v) : (u,v) in P}
        #update maxflow with capacity
        maxflow += c_f(p)

    #continue traversal of paths
    for each edge (u,v) in P:
        #change edge weights in G and G_f
        #forward edge
```

```
        if (u,v) in E:
            (u,v).f = (u,v).f + c_f(P)
        #backward edge
        else:
            (v,u).f = (v,u).f - c_f(P)

    return maxflow
```

(c) (2 pts) Find the max-flow in your example graph by stepping though your algorithm. Provide a new graph image each time the flow though an edge is altered.
ANSWER:



4. (10 pts) There are n rooms numbered from $\{1, 2, ..n\}$ in Williams Village. Each of the rooms has a teleportation device with a value $v_i \ \ \forall i \in \{1, 2, ...n\}$. The teleportation device placed in the room i with value v_i teleports to a room numbered $(v_i \ \% \ n) + 1$.

   A room in Williams Village is said to be beautiful if you can get back to the room you started using the teleportation devices. The task is to count the number of beautiful rooms in Williams Village.
   The input to your algorithm is the list of values associated with the teleportation device placed in the rooms. The output should be a number indicating the number of beautiful rooms.

   Example 1
   Input: [2, 3, 4, 5]
   Output: 4
   Explanation: All the 4 rooms that are beautiful and the corresponding paths to get back to them are as follows.

   - **Room 1** The teleportation path for Room1 is Room1, Room3, and back to Room1
   - **Room 2** The teleportation path for Room2 is Room2, Room4, and back to Room2
   - **Room 3** The teleportation path for Room3 is Room3, Room1, and back to Room3
   - **Room 4** The teleportation path for Room4 is Room4, Room2, and back to Room4

   Example 2
   Input: [1, 2, 3, 6]
   Output: 2
   Explanation: It can be seen that only Room3 and Room4 are beautiful.

(a) (2 pts) Give a high-level explanation of how you plan to solve this problem. (How would you explain your potential solution to a younger sibling or someone who has never taken a computer science class?) Minimum 1 paragraph.

ANSWER: We start by making a graph. The graph shows all the rooms and also shows us which rooms we can teleport to. The path we take to teleport to a room is what we call a directed edge, meaning we can only travel forward there and cannot come back to the room we were in before teleporting. If we travel to every room, we can count the number of beautiful rooms. If we start in a room and end in the same room, then we can count that as a beautiful room. On our graph, we look for cycles where we start and end in the same room. When we visit a room, we mark that we have visited it and then teleport to the number of rooms we have. If we have 10 rooms, we will mark the room we start in as visited and visit 10 rooms. If we end up back in the starting room, which will be marked as visited already, we have found a beautiful room. If the last room we teleport to is not marked as visited, we have not found a beautiful room.

(b) (5 pts) Provide well commented pseudo code or actual code to solve the above problem.
ANSWER:

```
def beautifulRooms(R):
    #R is the teleportation device placed in the rooms
    #number of rooms
    num = R.len
    #counter
    count = 0
    #visited array to keep track of visited rooms
    visited[]

    #DFS
    for(i = 1; i <= num; i++):
        #set all the rooms as not visited
        for(j = 1; j <= num; j ++):
            visited[j] = false

        #set starting room to visited
        visited[i] = true

        #set current room as i
        current = i
        #create temp node to be next room
        temp = 0

        #visit n rooms
        #check if end room is the same as start room
        for(int m = 1; m <= n; m++):
            //find the next room to teleport to
            temp = (R[current]%n) + 1

            #if the rooms has already been visited
            if(visited[temp]):
                #it is a beautiful room
                #add to counter
                count++
```

```
            break
        //set the current room as the next room to teleport to
        current = temp
#return counter once all rooms have been visited
return count
```

Name: Pourna Sengupta

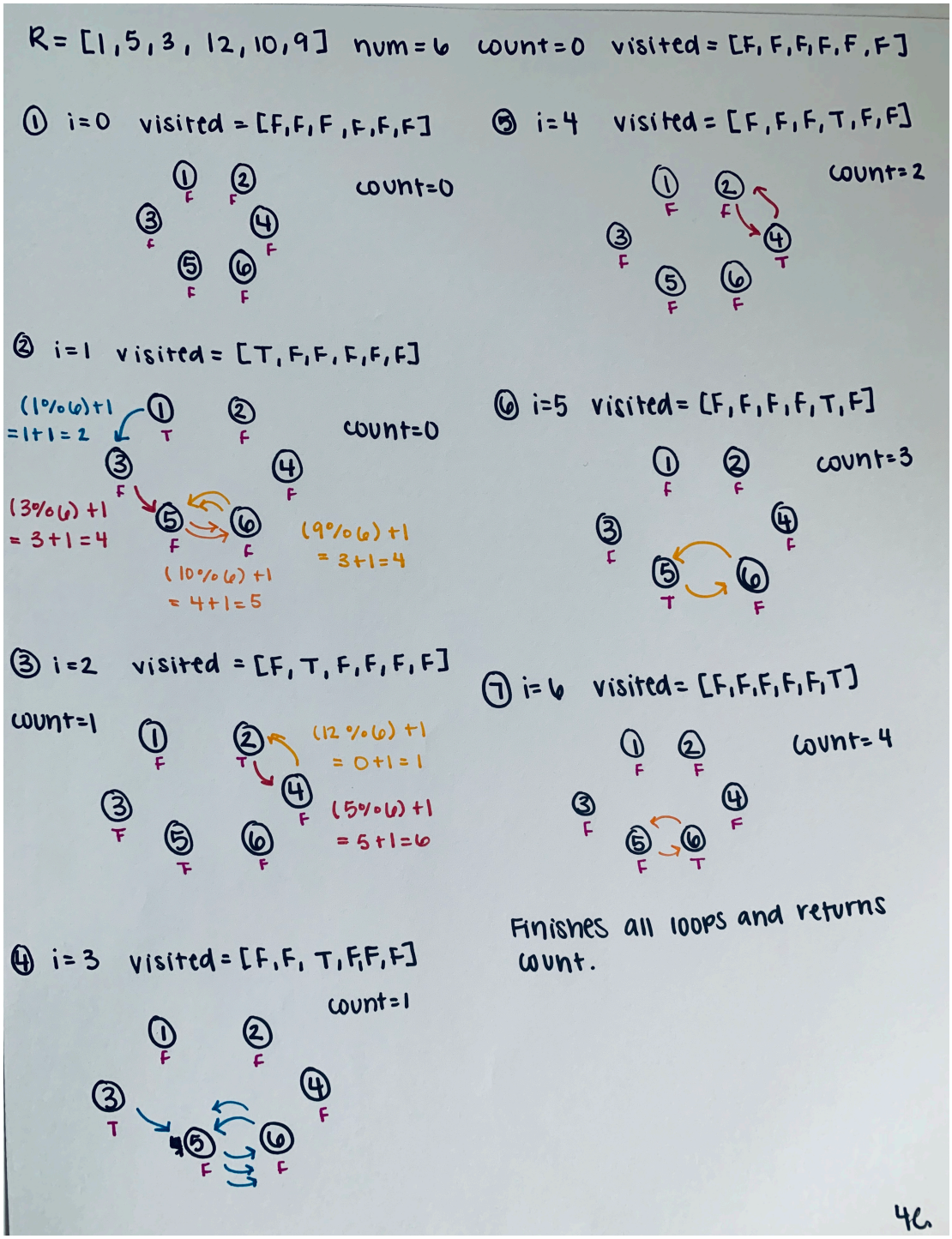
ID: 109086577

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Final Exam Summer 2020 (60 points) **Summer 2020, CU-Boulder**

(c) (3 pts) Give an example input with at least 5 rooms and structure of how your algorithm explores the rooms. This must be an image! Your input can **not** be an example already given.

ANSWER:



5. (10 pts) You are at your home in Boulder enjoying the summer and your friend challenges you to ride your bike for at least s miles starting from your home and ending at any of the scenic outlooks. You have a map of Boulder that has information about roads connecting various view points and your house. The task is to determine if there is a sequence of scenic outlooks that you can travel to starting from your house such that you have travelled at least s miles. You can not travel on the same edge more than 1 time and once you visit an outlook you cannot travel to it again.
   The input to your problem is a graph $G = (V, E)$ in the form of an adjacency list or adjacency matrix and the source vertex s corresponding to your house. Your algorithm should output a boolean value indicating if it's possible to complete the challenge of riding at least s miles starting from your house.

   (a) (2 pts) Give a high-level explanation of how you plan to solve this problem. (How would you explain your potential solution to a younger sibling or someone who has never taken a computer science class?) Minimum 1 paragraph.

   ANSWER: We begin at the house, which is the starting point. From there, we explore all the different paths we can take to see each of the outlooks. As we explore each path, we keep track of how many miles we have traveled. Every time we reach a new outlook, we look at how far we have traveled and see if it is greater or less than s miles. If it's greater, than we have found a path that works. If not, we keep trying different paths till we have traveled s or more miles. When we reach an outlook, we mark that we've been there. If we cross it again, that means we have to start over and try a different path. We want to avoid going in loops so that we can see as many outlooks as possible.

(b) (6 pts) Provide well commented pseudo code or actual code to solve the above problem.
ANSWER:

```
//helper function to compare route traveled to s
bool bestRouteHelper(int start, int s){
    //create array for the route travelled
    vector<bool> route(V, false);

    //add the starting vertex to the route
    route[start] = 1;

    //call main function and return value
    //use route value
    return bestRoute(start, s, route);
}

//function to find best route
bool bestRoute(int start, int s, vector<bool> &route){
    //check that the min miles is not 0 or negative
    //return true if it is
    if(s <= 0){
        return true;
    }

    //declare variables
    int vertex = 0;
    int weight = 0;

    //recursively traverse all the routes
    //start at the source vertex (house)
    //use a for loop that goes through adjacent vertices
    for(i = adj[start].begin(); i != adj[start].end(); i++){
        //find the adjacent vertex
        int vertex = (*i).first;
        //find the weight of the edge
        int weight = (*i).second;
```

```
            //check if vertex has been visited
            //if it has, ignore the edge
            if(route[vertex] == true){
                //ignores the edge
                //continues traversal
                continue;
            }

            //if the weight is greater than s
            //return true
            if(weight >= s){
                return true;
            }

            //else, add the vertex to the route
            else{
                route[vertex] = true;
            }

            //if the adjacent vertex creates a route greater than s
            //return true
            if(bestPath(vertex, s - weigth, route)){
                return true;
            }

            //backtrack route
            route[vertex] = false;
        }

        //if the adjacent vertex can't create a route greater than s
        //return false
        return false;
    }
```

(c) (2 pts) Explain your algorithms runtime and space complexity by analyzing your code. For example, stating that a sorting algorithm runs in $\mathcal{O}(n\log(n))$ without any justification is insufficient.

ANSWER:
The time complexity of this algorithm is $O(n!)$. Each route is visited and the weights are compared to s for each. We start at the source vertex and we have n-1 adjacent nodes. The time needed to connect two vertices is 2 since we join the source and the adjacent vertex. We then continue to the next node from n-1 adjacent nodes to n-2 adjacent nodes. So the recurrence relation for the times complexity is $T(n) = n*(2+T(n-2))$ which can be simplified into $T(n) = O(n!)$. The algorithm has a space complexity of $O(n^2)$ because it uses arrays, vectors, and integers. Arrays and vectors use 4n bytes while arrays use 4 bytes, so we end up with 4n + 4n + 4 bytes in total.

6. (20 pts) There are an even number of items arranged along a line. Each item has a value v_i associated with it. You are competing against an opponent to collect items such that total value of your items collected is maximized. At each round of the game, one player is allowed to pick an item from either end of the line.

   Determine the maximum total value that can be obtained, if you are allowed to play first assuming that the opponent is equally clever. The input to your algorithm will be a list of values. The output should be a number indicating the maximum amount of value that you can obtain.

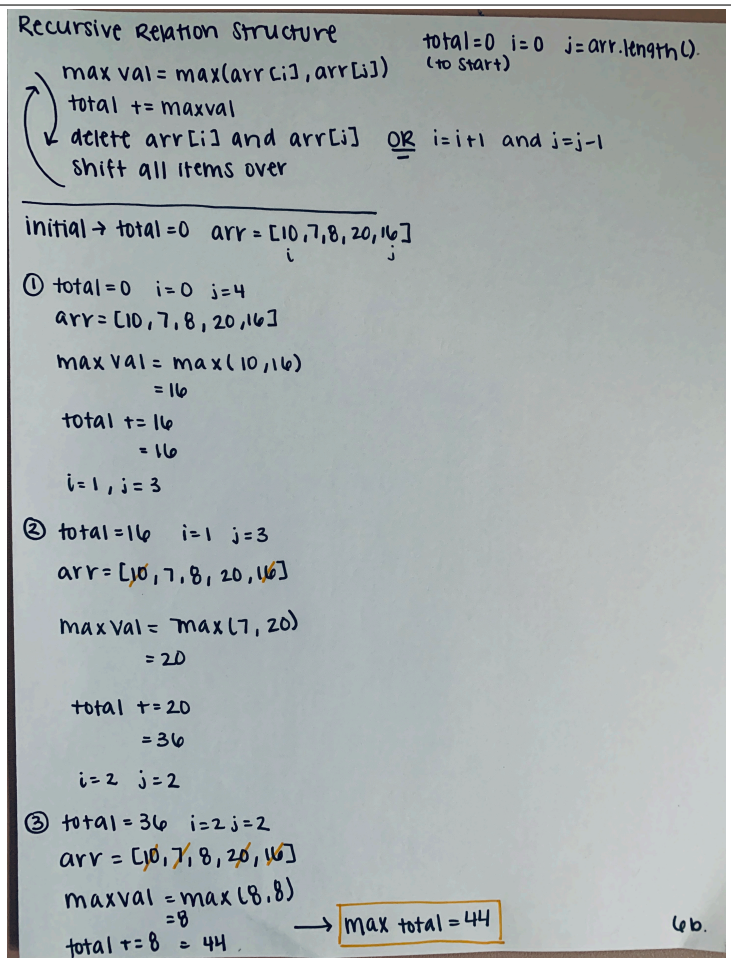
   Example
   Input: $22, 43, 10, 20$
   Output: 63
   Explanation: As the first player I would first pick 20 from the right end. The opponent would then pick 22. Out of the two elements 43, 10 remaining, I can pick 43. Thus 20+43=63 is the maximum total value that can be obtained.
   Note: Observe in the above example that the greedy choice of picking the max valued item does not work. If I had picked 22 instead of 20 in the first round, the maximum value that I could have obtained is 32.

   (a) (5 pts) State the base case and recursive relation that can be used to solve the above problem using dynamic programming.

   ANSWER:
   BASE CASE:
   if $n = 2$, $DP(i, j) = max(v_i, v_j)$
   if $i = j$, $DP[i, j] = v_i$
   RECURSIVE RELATION:
   $DP(i, j) = max[v_i + min(DP(i+2, j), DP(i+1, j-2)), v_j + min(DP(i+1, j-1), DP(i, j-2))]$

(b) (5 pts) Provide an example input consisting of at least 5 items. Show how the dynamic programming data structure used to store previous computations is built based on the recursive relation.

ANSWER:



(c) (8 pts) Write down well commented pseudo-code or paste real code to solve the above problem using Dynamic Programming or Memoization based on the above recurrence relation.
ANSWER:

```
int maxTotal(int arr[], int i, int j, int total){
    //declare variables
    //max value
    int maxVal = 0;
    //size of array
    int n = sizeof(arr);

    //Base Case 1
    //if there are only 2 items
    if(n == 2){
        //return the max item as maxTotal
        maxVal = max(arr[i], arr[j]);
        return maxVal;
    }

    //Base Case 2
    //if the items are the same value
    if(i == j){
        //set maxVal to the items value
        maxVal = arr[i];
    }
    else{
        //compare i and j values
        //update maxVal to maximum
        maxVal = max(arr[i], arr[j]);
    }
    //recursively call function
    total = maxTotal(arr[], i+1, j-1, total+maxVal);
    return total;
}
```

(d) (2 pts) Discuss the space and runtime complexity of the code, providing necessary justification.

ANSWER Time Complexity: $O(n)$ Space Complexity: $O(n)$
Since there are no nested loops in the function or helper function, the time complexity is simply $O(n)$. The space complexity is $O(n)$ since there is an array being used in the algorithm which takes up 4n bytes of space.

7. **Extra Credit (3 pts) will only be considered if your final exam score is less than 100%**

   Write two double spaced pages about the job or position you hope to have when you graduate from college (If you already have a job lined up, write about your next career goal). Your essay must include i) a tractable career goal, ii) a reason why you want to be hired/accepted to this position, iii) a step-by-step plan of how you intend to achieve your goal.

   I would like to work for CU Boulder's OIT Department with the Design and Networking team. The team works on the network for the devices installed into classrooms as well as the schools network as a whole. I have been very interested in working in the cyber security industry and I believe this would be a great next step for me to gain hands-on industry knowledge while also having flexibility in what I learn and work on. I currently work for OIT's Classroom Support department and I have been working alongside the integrators doing installs with them. I am very keen on learning the network of the devices we install and I currently have access to mentors who can teach me the work I would like to do. By working with them as a student, I am able to show them my knowledge, interest in learning, and work ethic while working with them for almost two years. Since I am already knowledgeable of all of the installment processes, the jump to back-end programming of the devices and network is not difficult. I plan on achieving this goal by seeking out mentors that work for our department in various positions. I have the opportunity to learn from programmers, engineers, and other experts who work on our AV and other classrooms systems daily. By doing this, I not only have the chance to learn skills at the industry level, but I can also benefit our

department and the work we do. Since I graduate within the next year, the department has the opportunity to mold me into an employee with the knowledge they need on our systems. With this, I can apply for positions within the department after graduating and can guarantee that I have the knowledge and skills they are looking for. I am also focusing my last year of classes on cyber security and networking so that I can be more of an asset to my department and have the base knowledge I need to work in the industry. By gaining hands-on experience with these mentors as a student, I can apply the concepts I learn in my courses to the work I do with the university. This gives me an edge over other graduates fresh out of college since finding an experience like mine is very difficult. Compared to an internship, having almost two years of experience working with such a large system of devices and school wide network is almost impossible to find anywhere else. I am lucky to have such a great support system within my workplace where my own manager and others are ready to vouch for me and my ability to learn and work with these mentors in an effective and beneficial manner. Aside from my knowledge through my courses, I have also been working on installs and learning every step of the process of setting up classroom systems as a whole. Because of my interest in learning and working outside of just the classroom support role, I have been able to surpass the knowledge I would get from traditional CS courses and enter knowledge that you only receive from on the job training and practice. Knowing how to install and commission systems from start to finish is a very appealing trait I will have right out of college and I believe that it gives me the opportunity to continue working for OIT and continue learning from those around me.