- The instructions are the same as in Homework-0, 1.

There are 6 questions for a total of 100 points.

---

1. (12 points) Given a strongly connected directed graph, $G = (V, E)$ with positive edge weights along with a particular node $v \in V$. You wish to pre-process the graph so that queries of the form "what is the length of the shortest path from s to t that goes through v" can be answered in constant time for any pair of distinct vertices s and t. The pre-processing should take the same asymptotic run-time as Dijkstra's algorithm. Analyse the runtime and provide a proof of correctness.
Answer:

Main Idea: For the pre-processing part, we are given the vertex v which should be present in the path from the starting vertex, s, to the target vertex, t, which would be given later. So, what we plan is run Dijkstra's algorithm on the given graph G, with v as the starting vertex, and store all this shortest distances from v to any other vertex in the graph. Again, we will reverse the graph and run Dijkstra's algorithm on reversed graph starting from vertex v, and store all these distances as well.
The required algorithm for pre-processing is as follows:

preprocess(G,v):
-initialize arrays from and to for every vertex
-to ¡- Dijkstra(G.v)
-G' ¡- reverse(G)
-from ¡- Dijkstra(G',v)
-return to,from

shortestPathThroughV(G,s,e): (not including v as I assume it was already given)
-to,from ¡- preprocess(G,v)
-distance ¡- to[t] + from[s]
-return distance

Running Time: We are running Dikjstra twice in our preprocessing, as well as reversing the graph. The Dijkstra takes O((V+E)logV), while reversing the graph is O(V+E). So, our algorithm is O((V+E)logV), i.e. in order of Dijkstra.

Proof of correctness: First of all, we need to proof that the shortest path from v to s in reverse graph, is another shortest path from s to v in the original graph.
Let us suppose that there is some shorter path in original graph, p', rather than the reverse of path from v to s in the original graph, p.
This leads to a contradiction because the reverse graph Dijkstra from v to s would have taken reverse of p' instead of p as the shortest path.
Therefore, our original claim is true.
Now, the main idea our algorithm is based on the fact that the length of the shortest path from s to t through v is equal to the sum of length of shortest path from v to t in given graph and the length of shortest path from v to s in the reverse graph.
Proof:
Let us suppose that there exists a smaller path from s to t through v.
So, this implies that either the path from s to v is shorter than ours or the path from v to t is shorter than ours or both.
If the path from v to t is smaller in the given path, than it is a contradiction as the Dijkstra with v as starting node would've taken that path to t instead of our one.

We have proved for the path from s to v before this proof.
Hence, our pre-processing gives the length of required path in unit time.

2. Counterexamples are effective in ruling out certain algorithmic ideas. In this problem, we will see a few such cases.

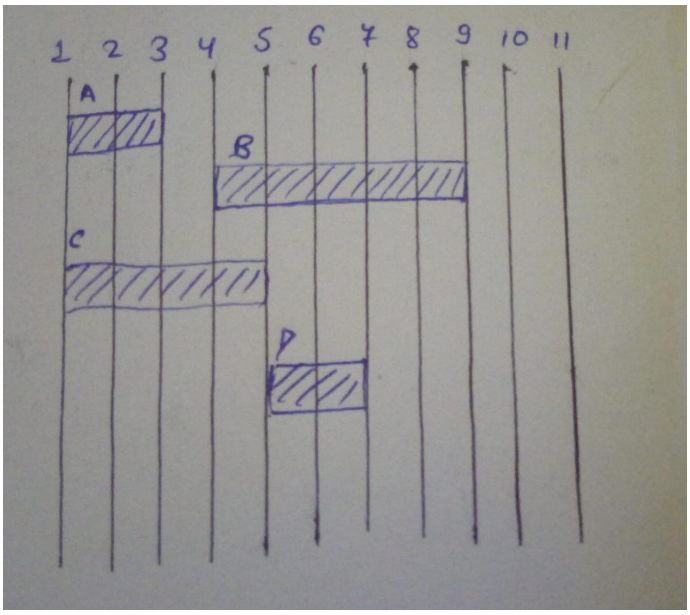
   (a) (5 points) Recall the following event scheduling problem discussed in class (lecture 15):

   > You have a conference to plan with n events and you have an unlimited supply of rooms. Design an algorithm to assign events to rooms in such a way as to minimize the number of rooms.

   The following algorithm was suggested during class discussion. `ReduceToSingleRoom`$(E_1, ..., E_n)$
   - $U \leftarrow \{E_1, ..., E_n\}$; $i \leftarrow 1$
   - While U is not empty:
     - Use Earliest Finish Time greedy algorithm on events in set U to schedule a subset $T \subseteq U$ of events in room i
     - $i \leftarrow i + 1$; $U \leftarrow U \setminus T$

   Show that the above algorithm does not always return an optimal solution.

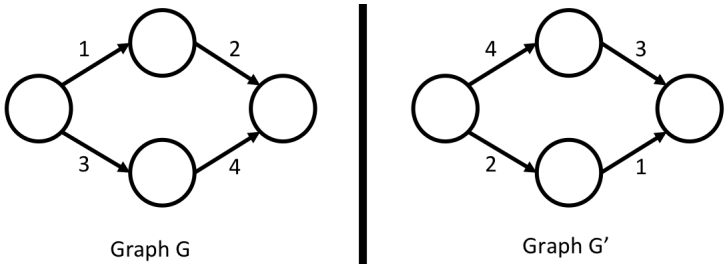
   Answer:
   Consider the following set of events:



   If we go according to the suggested algorithm, it will allot a room to (A,D)(because D has earlier finish time as B), another to C, and another to B. So, in total, it will take 3 rooms, while a better assignment of (A,B) to one room, and (C,D) to another is possible, which results in just occupying 2 rooms.
   Hence, our algorithm doesn't always give the optimal solution.

   (b) (5 points) A longest simple path from a node s to t in a weighted, directed graph is a simple path from s to t such that the sum of weights of edges in the path is maximised. Here is an idea for finding a longest path from a given node s to t in any weighted, directed graph $G = (V, E)$:

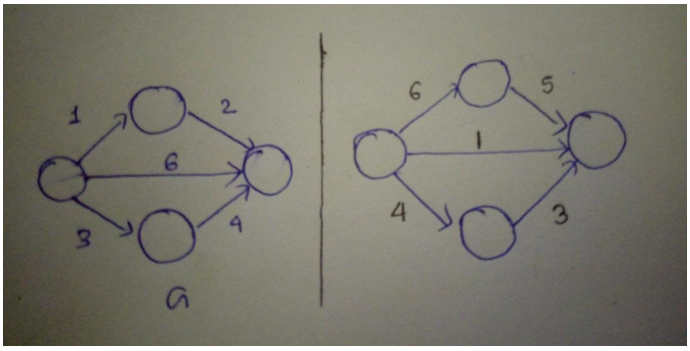
Let the weight of the edge $e \in E$ be denoted by $w(e)$ and let w_{max} be the weight of the maximum weight edge in G. Let G' be a graph that has the same vertices and edges as G but for every edge $e \in E$, the weight of the edge is $(w_{max} + 1 - w(e))$. (*For example, consider the graph G below and its corresponding graph G'.*)



Run Dijkstra's algorithm on G' with starting vertex s and return the shortest path from s to t.

Show that the above algorithm does not necessarily output the longest simple path.

Answer:
Consider the following case:



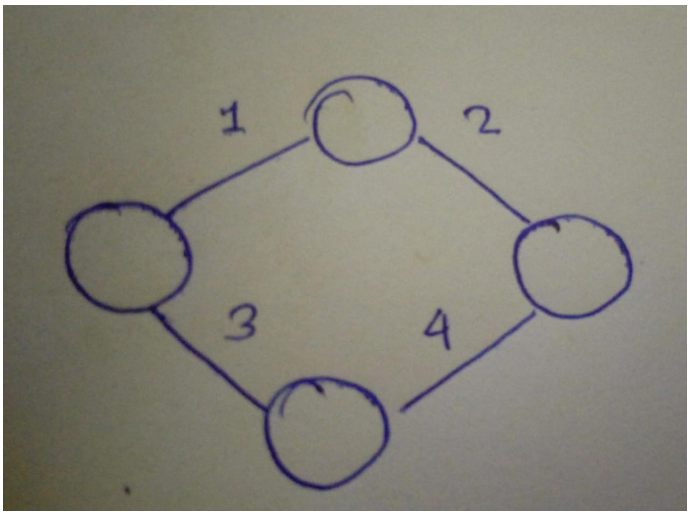
The graph on the left hand side is the graph we are considering, and the right hand side figure is the transformed graph as suggested in the question.
Now, when we run Dijkstra on the transformed graph, we get the shortest path as the single edge from source to sink node, whose transform, i.e. 6 is not the longest path in original graph. The longest path is the one which consists of the edge weighted 4, followed by the one weighted 3.
Hence, our algorithm doesn't output the longest path all the time. Therefore, incorrect.

(c) (5 points) Recall that a *Spanning Tree* of a given connected, weighted, undirected graph $G = (V, E)$ is a graph $G' = (V, E')$ with $E' \subseteq E$ such that G' is a tree. The cost of a spanning tree is defined to be the sum of weight of its edges. A *Minimum Spanning Tree (MST)* of a given connected, weighted, undirected graph is a spanning tree with minimum cost. The following idea was suggested for finding an MST for a given graph in the class.

Dijkstra's algorithm gives a shortest path tree rooted at a starting node s. Note that a shortest path tree is also a spanning tree. So, simply use Dijkstra's algorithm and return the shortest path tree.

Show that the above algorithm does not necessarily output a MST. In other words, a shortest path tree may not necessarily be a MST. (*For this question, you may consider only graphs with positive edge weights.*)

Answer:
Consider the following undirected graph:



So, what we want to state here is that the shortest path tree being the minimum spanning tree depends on the node from where we start running our Dijkstra.
Let us suppose we ran Dijkstra on above graph with the rightmost node as the starting vertex, the tree we will get will have a sum of edges as 7, i.e. the weight of this spanning tree is 7.
If we ran Dijkstra on the leftmost node, we will get a spanning tree weighted 6, which is the minimum spanning tree.
Hence, the algorithm depends on the starting vertex and does not necessarily output the minimum spanning tree.

3. (*Example for "greedy stays ahead"*) Suppose you are placing sensors on a one-dimensional road. You have identified n possible locations for sensors, at distances $d_1 \leq d_2 \leq ... \leq d_n$ from the start of the road, with $0 \leq d_1 \leq M$ and $d_{i+1} - d_i \leq M$. You must place a sensor within M of the start of the road, and place each sensor after that within M of the previous one. The last sensor must be within M of d_n. Given that, you want to minimize the number of sensors used. The following greedy algorithm, which places each sensor as far as possible from the previous one, will return a list $d_{i_1} \leq d_{i_2} \leq ... \leq d_{i_k}$ of locations where sensors can be placed.

`GreedySensorMin(`$d_1 ... d_n, M$`)`
- Initialize an empty list
- Initialize $I = 1$, $PreviousSensor = 0$.
- While $(I < n)$:
  - While $(I < n$ and $d_{I+1} \leq PreviousSensor + M)$ $I++$
  - If $(I < n)$ Append d_I to list; $PreviousSensor = d_I$;$I++$.
- if list is empty, append d_n to list
- return(list)

In using the "greedy stays ahead" proof technique to show that this is optimal, we would compare the greedy solution $d_{g_1}, ..d_{g_k}$ to another solution, $d_{j_1}, ..., d_{j_{k'}}$. We will show that the greedy solution "stays ahead" of the other solution at each step in the following sense:
Claim: For all $t \geq 1, g_t \geq j_t$.

(a) (5 points) Prove the above claim using induction on the step t. Show base case and induction step.
Answer

Base Case :-
t = 1
$d_{g1} \geq d_{j1}$ since the greedy approach places the sensor as far as possible from the previous one.

Induction Hypothesis :-
For all t in range 1,2,...,(k-1), let $d_{gt} \geq d_{jt}$

Induction Step :-
Looking at d_{gk}, since the successor pole has to be farthest possible from the previous pole, we can say that $d_{gk} = d_{g(k-1)}$ + M.
Let $d_{jk} = d_{j(k-1)}$ + M' where $(0 \leq M' \leq M)$.
From induction hypothesis we know that $d_{j(k-1)} \leq d_{g(k-1)}$
Using the above 3 equations, we can write,
$d_{jk} = d_{j(k-1)} + M' \leq d_{j(k-1)} + M \leq d_{g(k-1)} + M \leq d_{gk}$
So, $d_{jk} \leq d_{jk}$ for all $k \geq 1$.
Hence the claim is proven true.

(b) (3 points) Use the claim to argue that $k' \geq k$. (Note that this completes the proof of optimality of the greedy algorithm since it shows that greedy algorithm places at most as many sensors as any other solution.)
Answer
Proof by Contradiction :-
Let us say k' (number of poles by some other solution) ¡ k (number of poles from greedy solution)
Now from the claim proven above we know that $d_{jk'} \leq d_{gk'}$. And we also know that $d_{jk'} = d_{gk}$.
Using these 2 equations, $d_{gk} \leq d_{gk'}$. But this leads to contradiction since we assumed k' ¡ k.
So, we always have $k' \geq k$.
Hence Proved

(c) (2 points) In big-O notation, how much time does the algorithm as written take? Write a brief explanation.
Answer
Running Time = Time to traverse position of every sensor
= Number of locations for n sensors = O(n)

4. (*Example for "modify the solution"*) You have n cell phone customers who live along a straight highway, at distances $D[1] < D[2] < ... < D[n]$ from the starting point. You need to have a cell tower within Δ distance of each customer, and want to minimize the number of cell towers.

(*For example, consider* $\Delta = 3$ *and there are* 3 *customers (i.e.,* $n = 3$*) with* $D[1] = 3, D[2] = 7, D[3] = 10$. In this case, you can set up two cell towers, one at 6 and one at 10.)

Here is a greedy strategy for this problem.

> **Greedy strategy**: Set up a tower at distance d which is at the farthest edge of the connectivity range for the customer who is closest to the starting point. That is, $d = \Delta + D[1]$. Note that all customers who are within Δ distance of this tower at d, are covered by this tower. Then recursively set up towers for the remaining customers (who are not covered by the first tower).

We will show that the above greedy strategy gives an optimal solution using modify-the-solution. For this, we will first need to prove the following exchange lemma.

Exchange Lemma: *Let G denote the greedy solution and let g_1 be the location of the first cell phone tower set up by the greedy algorithm. Let OS denote any solution that does not have cell phone tower at g_1.*

Then there exists a solution OS' that has a cell phone tower set up at g_1 and OS' has the same number of towers as OS.

Proof. Let $OS = \{o_1, ..., o_k\}$. That is, the locations of the cell phone towers as per solution OS is $o_1 < o_2 < ... < o_k$. We ask you to complete the proof of the exchange lemma below.

(a) (1 point) Define OS'.
Answer
Let j_1 be the first location in OS
Let g_1 be the first location chosen by the greedy method Now we know that $j_1 \leq g_1$, otherwise the first customer will not be covered.
OS' = OS - j_1 + g_1 (we replace the first location in OS by g_1)

(b) (2 points) OS' is a valid solution because ... (*justify why OS' provides coverage to all customers.*)
Answer
OS is a valid solution, so every customer is covered by the towers set up at the locations given by OS. We replace the first tower location and set up the tower at g_1 to obtain OS'. Now like OS, OS' also covers all the customers since all the customers covered by j_1 are also covered by g_1 as g_1 is within Δ of every customer covered by j_1. The remaining customers are covered in OS' exactly the same way as in OS.
So, all customers are covered with the locations of cell phone towers given by OS'.
Hence OS' is a valid solution.

(c) (2 points) The number of cell phone towers in OS' is at most the number of cell phone towers in OS because... (*justify*)
Answer
Since the coverage of j_1 is $\leq$ the coverage of g_1, upto Δ + D[1] distance, there might be j1, j2, ..., jk towers in OS where k $\geq$ 1.
Hence, $|OS'| \leq |OS|$.
Therefore, number of towers in OS' is atmost the number of towers in OS.

We will now use the above exchange lemma to argue that the greedy algorithm outputs an optimal solution for any input instance. We will show this using mathematical induction on the input size (i.e., number of customers). The base case for the argument is trivial since for $n = 1$, the greedy algorithm opens a single tower which is indeed optimal.

(a) (3 points) Show the inductive step of the argument.
Answer
Induction on Number of customers.
Induction Hypothesis :-
Greedy solution is optimal for n = 1,2,... (k-1) customers ie it requires minimum number of towers to cover n customers using the greedy approach.

Induction Step :-
Let GS(k) be the greedy solution for k customers. The first tower location in GS(k) is g (say).
Let OS be any other optimal solution for k customers.
Using the Exchange Lemma, we can say that there exists an OS' such that it has the first location of tower at g and remaining locations same as that of OS.
Then $|OS'| \leq |OS|$.

$|GS(k)| = 1 + |GS(k-1)| \leq 1 +$ |some other solution for k-1 customers| $= |OS'| \leq |OS|$
So, the greedy solution for

Having proved the correctness, we now need to give an efficient implementation of the greedy strategy and give time analysis.

(a) (5 points) Give an efficient algorithm implementing the above strategy, and give a time analysis for your algorithm.

Answer

```
procedure locations(Δ,D,n):
    d = initialise()  initialises d to an empty list
    d.append(Δ + D[1])
    dist = Δ + D[1]
    for i in range [2,n]:
        if(D[i] - dist ¿ Δ):
            dist = Δ + D[i]
            d.append(dist)

    return(d)
```

Running Time = Time to traverse the array of customer locations (D) once + k*(Time to append to list d)

= O(n) + k*O(1)
= O(n) + O(k)
= O(n) here n is the number of customers

5. (25 points) You are a conference organiser and you are asked to organise a conference for a company. The capacity of the conference room is limited and hence you would want to minimise the number of people invited to the conference. To make the conference useful for the entire company, you need to make sure that if an employee is not invited, then every employee who is an immediate subordinate of this employee gets the invitation (*if an employee is invited, then you may or may not invite a subordinate*). The company has a typical hierarchical tree structure, where every employee except the CEO has exactly one immediate boss.

   Design an algorithm for this problem. You are given as input an integer array $B[1...n]$, where $B[i]$ is the immediate boss of the i^{th} employee of the company. The CEO is employee number 1 and $B[1] = 1$. The output of your algorithm is a subset $S \subseteq \{1, ..., n\}$ of invited employees. Give running time analysis and proof of correctness.

   Answer:
   (Greedy - Modify the solution)
   Greedy Strategy: Construct a graph where a directed edge from i to j implies B[j]=i. This will be a tree with the CEO as the root. Pick an employee all of whose subordinates are leaves. This means we don't need to pick any of the subordinates. Remove the subtree from the graph and repeat till the graph is empty.
   Algorithm:
   First, we will construct the adjacency list of the graph G, where edge exists from i to j if j is the immediate subordinate of i.Then we will run BFS on G from i=1(CEO) to get a list of all levels of employees. Then starting from the last layer,we will go through all the employees and if an employee is not invited, we will invite the employee's boss.

   To create the graph G

   1) Initialize L to be arraylist of size n with empty arraylists at each index //for the adjacency list of graph G

```
for i = 1 to n:
    b = B[i]
    add i to L[b]
```

   2) Perform BFS(G) and store each level in the list levels. Let the number of levels be k.
   Initialize arraylist of size n invited as false for all i = 1..n

```
3) call procedure inviteelist on graph G and list of levels
procedure inviteelist(G,B,levels):
    invitees = empty list
    for i = k to 1:
        l = levels[i] //list of nodes at level i
        for all e in l:
            if invited[e] == false:
                if(not invited[B[e]]):
                    invited[B[e]] = true
                    add B[e] to list invitees
    return invitees
```

Exchange Lemma: *Let G denote the greedy solution and let i be the first employee picked according to the greedy strategy. Let OS denote any solution that does not include i. Then there exists a solution OS' that includes i and OS' has the same or lesser number of employees invited than OS.*

Proof. Let $OS = \{i_1, ..., i_k\}$. The greedy pick is i for some $1 \leq i \leq n$.
According to the constraints given, if $i \notin OS$ that implies that all of i's immediate subordinates(which are leaves of the graph according to the greedy strategy) will be invited.
We can remove all the subordinates of i and add i to OS to get OS'.The

$$\begin{aligned} &number\ of\ subordinates\ of\ any\ employee \geq 1. \\ &\implies |OS'| = |OS| + 1 - (number\ of\ subordinates\ of\ i) \\ &\implies |OS'| \leq |OS| \end{aligned}$$

Therefore the exchange lemma generates OS' which includes i and has at most as many invited employees as OS.

Proof of Optimality. Proof by induction on the number of employees n.
Let $GS(J)$ be the greedy solution and $OS(J)$ be the any valid solution for n employees.
Base Case: for n=2, there will be the CEO and one employee subordinate to him/her. Then only the CEO should be invited which is the output of our greedy strategy. Hence true.
Induction Hypothesis: Consider that the greedy strategy gives optimal solution for all k = 1...n-1.
Induction Step: Consider the problem J for n employees. According to the greedy strategy, after the greedy pick the problem is reduced to n-(m+1) employees where the first employee invited had m subordinates since they are all leaves.

$$\begin{aligned} |GS(J)| &= 1 + |GS(J')|\ (J'\ represents\ the\ problem\ with\ n - (m+1)\ employees) \\ &< 1 + |OS(J')|\ (Induction\ hypothesis) \\ &\leq |OS'(J)|\ (Exchange\ Lemma) \\ &< |OS(J)| \end{aligned}$$

Hence, the greedy solution gives a solution which is as good as any solution for the given problem.
Hence proved.

Running Time: The number of edges in the graph is n-1(every node has exactly one incoming edge), where n is the number of employees. The time to create the adjacency list is O(V+E) which is O(n). The time to run BFS is also O(n). In procedure inviteelist, each node is evaluated once, hence time taken is O(n). Therefore, the total time complexity is O(n).

6. (25 points) A town has n residents labelled $1, ..., n$. In the midst of a virus outbreak, the town authorities realise that hand sanitiser has become an essential commodity. They know that every resident in this

town requires at least T integer units of hand sanitiser. However, at least $\lceil \frac{n}{2} \rceil$ residents do not have enough sanitiser. On the other hand, there may be residents who may have at least T units. With very few new supplies coming in for the next few weeks they want to implement some kind of sharing strategy. At the same time, they do not want too many people to get in close proximity to implement sharing. So, they come up with the following idea:

Try to *pair up* residents (based on the amount of sanitiser they possess) such that:

1. A resident is either unpaired or paired with exactly one other resident.
2. Residents in a pair together should possess at least $2T$ units of sanitiser.
3. The number of unpaired residents with less than T units of sanitzer is minimised.

Once such a pairing is obtained, the unpaired residents with less than T units of sanitiser can be managed separately. The town authorities have conducted a survey and they know the amount of sanitiser every resident possesses. You are asked to design an algorithm for this problem. You are given as input integer n, integer T, and integer array $P[1...n]$ where $P[i]$ is the number of units of sanitiser that resident i possesses. You may assume that $0 \leq P[1] \leq P[2] \leq ... \leq P[n]$. Your algorithm should output a pairing as a list of tuples $(i_1, j_1), (i_2, j_2), ..., (i_k, j_k)$ of maximum size such that (i) For all $t = 1, ..., k$, $P[i_t] + P[j_t] \geq 2T$ and (ii) $i_1, ..., i_k, j_1, ..., j_k$ are distinct. Give proof of correctness of your algorithm and discuss running time.

Answer:
(Greedy - Modify the solution)
The greedy strategy used is
Greedy Strategy: Pair up residents from the front and back of the list P[1...n]. If $P[i] + P[j] < 2T$ then put resident i in the unpaired list and check P[i+1] with P[j]. Keep doing until all the residents are either paired or put in the unpaired list.

Algorithm :

```
procedure findpairs(P,n,T):
    L = empty list
    i = 0
    j = n
    while(i<j):
        if(P[i]+P[j]≥2T):
            add pair (i,j) to L
            i = i+1
            j = j-1
        else:
            i = i+1
    return L
```

Exchange Lemma: *Let G denote the greedy solution and let $g_1 = (i, n)$ be the first pair of residents picked. According to the greedy strategy the first pair will have resident n. Let OS denote any solution that does not have the pair g_1. Then there exists a solution OS' that has the pair g_1 and OS' has the same or more number of pairs as OS.*

Proof. Let $OS = \{o_1, ..., o_k\}$. The greedy pick $g_1 = (i, n)$ for some $1 \leq i \leq n$

- Case1: resident i is paired but n is not. Let $(i, j) \in OS$ for some $1 \leq j < n$
  . Replace the pair (i,j) with (i,n) and put j in the unpaired list. The number of pairs remains same.
- Case2: resident n is paired but i is not. Let $(i', n) \in OS$ for some $1 \leq i' \leq n$. Replace the pair (i',n) with (i,n) and put i' in the unpaired list. The number of pairs remains same.
- Case3: Neither i nor n are paired. Add the pair (i,n) to the OS. (i,n) is valid according to the greedy strategy. The number of pairs increases by 1.

- Case4: Both residents i and n are paired.Let $(i', n), (i, j') \in OS$ for some $1 \leq i' \leq n$.

$$P[i'] + P[n] \geq 2T \ -(1)$$
$$P[i] + P[j'] \geq 2T \ -(2)$$

According to the greedy strategy, i is the first resident such that $P[i] + P[n] \geq 2T$.

$$\begin{aligned} &\implies P[l] + P[n] < 2T \ (\forall\, l < i) \\ &\qquad \implies i' > i \ (since\ all\ l < i\ are\ unpaired) \\ &\qquad \implies P[i'] \geq P[i] \ -(3) \\ &\implies P[i'] + P[j'] \geq 2T \ (from\ (2)\ and\ (3)) \\ &\qquad P[i] + P[n] \geq 2T \ (greedy\ first\ pick,\ so\ valid) \end{aligned}$$

Replace (i',n) and (i,j') with (i,n) and (i',j'). The number of pairs remain same.

Therefore the exchange lemma generates OS' which includes (i,n) and has atleast as many pairs as OS.

Proof of Optimality. Proof by induction on the number of residents n.
Let $GS(J)$ be the greedy solution and $OS(J)$ be the any valid solution for n residents.
Base Case: for n=1, the resident will not be paired. For n = 2, The number of pairs will be 1 if $P[1] + P[2] \geq 2T$ which is the output of our greedy strategy. Hence true.
Induction Hypothesis: Consider that the greedy strategy gives optimal solution for all k = 1...n-1.
Induction Step: Consider the problem J for n residents.

$$\begin{aligned} |GS(J)| &= 1 + |GS(J')|\ (J'\ represents\ the\ problem\ with\ n-2\ residents) \\ &> 1 + |OS(J')|\ (Induction\ hypothesis) \\ &\geq |OS'(J)|\ (Exchange\ Lemma) \\ &\geq |OS(J)| \end{aligned}$$

Hence, the greedy solution gives a solution which is as good as any solution for the given problem. Hence proved.

Running Time: Worst case we traverse the list exactly once. Therefore time complexity is O(n).