Answer all questions. Give informal (at least) proofs for all answers. Grading will be on completeness and logical correctness, as well as correctness. For the data structure question, the efficiency of your solution will also be taken into account.

Analyzing loops - 10 pts Consider the following iterative algorithm, that uses an \( \Theta(I) \) time procedure \( \text{proc}(I) \), which does not change \( I \).

```
Algorithm(n: positive integer);
1. begin;
2. \( T \leftarrow 1 \);
3. While \( T^2 \leq n \) do:
4. begin; \{while\}
5. \( \text{proc}(T^2) \);
6. \( T++ \);
7. end; \{while\}
8. end;
```

Give a time analysis, up to \( \Theta \), for this algorithm.

Proof of correctness - 10 pts Below is an algorithm, that, given a sorted array \( A[1..n] \) of integers, and an integer \( T \), decides whether there is a pair \( 1 \leq K < L \leq n \) with \( T = A[K] + A[L] \). After the algorithm, there is a proof of correctness, with some parts missing. Fill in the missing sections in the proof to get a complete proof of correctness.

```
IsSum[A[1..n]: sorted array of integers, T: integer];
1. \text{Found} \leftarrow \text{False};
2. \( I \leftarrow 1; \)
3. \( J \leftarrow n; \)
4. While \( I < J \) and NOT \text{Found} do:
5. begin; \{while\}
8. end; \{while\}
9. Return \text{Found}.
```

Proof of correctness: Let \( A[1..n] \) be a sorted array, i.e., \( I \). Let \( T \) be any integer. We need to show that if \( II \) for \( III \ 1 \leq K < L \leq n \) then \( \text{IsSum} \) returns \text{True}; and conversely, that if \( \text{IsSum} \) returns \text{True}, then for \( IV \ 1 \leq K < L \neq n \), \( V \).

To see the first direction, we use the loop invariant method. Assume there are \( 1 \leq K \leq L \leq n \) with \( VI \). Let \( I_t \) and \( J_t \) be the values of \( I, J \) after \( t \) iterations of the while loop. The intuition is that \( K, L \) never leave the range \( I_t, J_t \) to be searched by the algorithm, i.e, that \( VII \ 1 \leq K < L \leq VIII \).

We prove this by \( IX \). First we show that it is true at \( t = 0 \). At initialization, \( I_0 = X \) and \( J_0 = XI \). Thus, \( XII \leq K < L \leq XIII \). So the \( XIV \) is true.

For the induction step, assume the statement is true after some number of loops \( t \), i.e. that \( XV \).

We want to show that after the next iteration, \( XVI \).

There are three cases: if \( A[I_t] + A[J_t] = T \), the algorithm returns true and halts, and there is nothing to prove.
If $A[I_t] + B[J_t] > T$, then in the next iteration $I_{t+1} = XVII$ and $J_{t+1} = XVIII$. Thus, the only way we could not have what we want to show, $I_{t+1} \leq K < L \leq J_{t+1}$, is if $XVIV = XX$. If that happens, then $T < A[I_t] + A[J_t] = A[I_t] + XXI \leq A[K] + XXII = Y$. This is a $XXIII$, which proves our goal, $XXIV$.

The final case $XXV < XXVI$, is very similar. In the next iteration $I_{t+1} = XXVII$ and $J_{t+1} = XXVIII$. Thus, the only way the induction claim could fail is if $XXVIV$. In this case, then $XXX > A[I_t] + A[J_t] = A[K] + A[J_t] \geq A[K] + A[L] = XXXI$, which is again a $XXXII$.

Thus, the loop invariant still holds in all three cases. So by induction it holds after any number of iterations $t$.

The first direction then follows from the loop invariant, since at the end of the loop, either $Found = True$ or $I_t = J_t$, a contradiction since then $J_t = I_t \leq XXXIII < L \leq J_t$. Thus, if $A[K] + A[L] = T$, the algorithm returns True.

To see the converse direction, if $IsSum$ returns True, then $Found$ must be set to True, which can only happen in the THEN clause of line $XXIV$. Because of the IF clause of that line, we must have $XXXV$, which proves the goal, since we can pick $K = I$ and $L = J$.

**Using data structures and pre-processing:** Say that a graph is $d$-dense if every node has at least $d$ edges adjacent to it. The following strategy determines whether graph $G$ has a non-empty $d$-dense sub-graph $G'$.

**DenseSubgraph(G: undirected graph, d: positive integer)**

1. While there is a node $x$ of degree $< d$ do:
   2. $G \leftarrow G - \{x\}$.
   3. If $G$ is empty return True, else return False.

Give an efficient implementation of the above strategy when $G$ is given in adjacency list format. Specify pre-processing and data structures used.


1. Program: TargetSum(A[1..n], B[1..n]: Arrays of Integers, T: integer):
   2. IF $n = 0$ return False;
   3. IF $n = 1$
          THEN return True;
          ELSE return False;
   4. $k \leftarrow \lceil n/2 \rceil$
   5. $k' \leftarrow \lfloor (n + 1)/2 \rfloor$
   7. IF $A[k] + B[k] > T$ THEN return $TargetSum(A[1..k], B[1..k], T)$
      OR $TargetSum(A[1..k], B[k'..n], T)$
      OR $TargetSum(A[k'..n], B[1..k], T)$;
   8. ELSE return $TargetSum(A[k'..n], B[1..k], T)$ OR $TargetSum(A[1..k], B[k'..n], T)$ OR $TargetSum(A[k'..n], B[k'..n], T)$;
Here, \( OR \) is the Boolean OR operation on the values of the returned calls.

Example: say \( n = 4 \), \( A[1..4] = 3, 8, 13, 21 \); \( B[1..4] = 1, 2, 3, 5 \); \( T = 13 \). Then the algorithm would first compare \( A[2] + B[2] = 10 \) to \( T \). Since it is smaller, it would recursively check \( \text{TargetSum}[A[1..2],B[3..4],13] \), \( \text{TargetSum}[A[3..4],B[1..2],13] \) and \( \text{TargetSum}[A[3..4],B[3..4],13] \). (It could ignore the case \( I, J \leq 2 \), since all such sums are less than \( T \)). After computing recursively, which I won’t show here, the sub-procedures would return \( True, False \) and \( False \). Thus, the main procedure would return \( True \).