1. (40 marks) **Descending Kruskal’s algorithm**

Another approach to the minimum spanning tree algorithms is to remove “heavy” edges from the graph until only the tree is left, rather than adding “light” edges to an originally empty graph.

(a) (20 marks) Design a version of Kruskal’s algorithm that would exploit this idea. Your algorithm should work by removing heavy edges from the original graph until only an MST is left. Provide pseudo-code for your algorithm.

**Answer:**

1. Sort the edges $e_1, \ldots, e_m$ in *non-increasing* order of weights
2. Set $S \leftarrow G$
3. for $i = 1$ to $m$ do
   1. To check connectivity: let $e_i = (u, v)$,
   2. run BFS starting from $u$ and see if $v$ is reachable from $v$
   3. if $S - \{e_i\}$ is connected then
      1. $S \leftarrow S - \{e_i\}$
   end if
4. end for
5. return $S$

(b) (20 marks) Prove that your algorithm always returns an MST of $G$. The steps of your proof will be similar to that of Kruskal’s algorithm. Clearly identify your loop invariant and prove that it is established and maintained.

**Answer:**

Let $S_i$ be the value of $S$ after $i$ iteration of the “for loop”; that is after examining edges $e_1, \ldots, e_i$. So $S_0$ is the initial value of $S$, which is $G$. We prove that the following predicate $LI(i)$ holds for all values of $i$, $0 \leq i \leq m$:

**Loop invariant** $LI(i)$: There exists a minimum spanning tree $S_{opt}$ of $G$ such that $(S_i \cap \{e_1, \ldots, e_i\} \subseteq S_{opt} \subseteq S_i)$. That is, there exists an optimal solution $S_{opt}$ that can be obtained by removing from $S_i$ some subset of edges from among $\{e_{i+1}, \ldots, e_m\}$.

It is easy to see that once we prove this predicate for all values of $i$ we are done, because: $LI(m)$ implies that for some minimum spanning tree $T_{opt}$: $S_m \cap \{e_1, \ldots, e_m\} \subseteq T_{opt} \subseteq S_m$. Since $\{e_1, \ldots, e_m\}$ is the set of all the edges, therefore, $S_m \cap \{e_1, \ldots, e_m\} = S_m$. This, shows that $S_m \subseteq T_{opt} \subseteq S_m$, i.e. $S_m = T_{opt}$.

Now we prove the loop invariant by induction on $i$. 

2. (30 marks) SubsetSum (greedy algorithms)

A SubsetSum is defined as follows: given positive integers \(a_1 \ldots a_n\) (not necessarily distinct), and a positive integer \(t\), find a subset \(S\) of \((1 \ldots n)\) such that \(\Sigma_{i \in S} a_i = t\), if it exists.

(a) (10 marks) Suppose each \(a_i\) is at least twice as large as the sum of all smaller numbers \(a_j\). Give a greedy algorithm to solve SubsetSum under this assumption.

- Answer:
  
  Sort the \(a_i\) in non-increasing order
  
  \(w = 0; \ S = \emptyset\)
  
  for \(i = 1\) to \(n\)
    
    if \(w \leq t - a_i\)
      
      \(S \leftarrow S \cup i\)
      
      \(w \leftarrow w + a_i\)
    
  if \(w = t\) return true
  else return false
(b) (20 marks) Prove correctness of your greedy algorithm by stating and proving the loop invariant.

• Answer:
  LI: \( S_i = S \cap 1, \ldots, i - 1 \), where \( \sum_{i \in S} a_i = t \), if such an \( S \) exists.

  Initialization: \( S_1 = \emptyset = S \cap 1, \ldots, 0 \rightarrow LI \) satisfied trivially.

  Maintenance:
  If no \( S \) satisfying \( \sum_{i \in S} a_i = t \) exists, then the LI is satisfied trivially. Now assume that such an \( S \) does exist.
  By inductive assumption we know that \( S_{i-1} = S \cap 1, \ldots, i - 2 \), where \( \sum_{i \in S} a_i = t \).
  If \( w < t - a_{i-1} \), then \( \sum_{j=i}^{n} a_i + w < t \). Thus \( a_{i-1} \in S \), and so by adding \( a_{i-1} \) to \( S \) the LI is maintained.
  If \( w = t - a_{i-1} \), then \( S = S_{i-1} \cup a_{i-1} \) satisfies \( \sum_{i \in S} a_i = t \), and so by adding \( a_{i-1} \) to \( S \) the LI is maintained.
  If \( w > t - a_{i-1} \), \( a_{i-1} \) cannot be added without exceeding \( t \), hence \( a_{i-1} \notin S \). Thus by not adding \( a_{i-1} \) to \( S \) the LI is maintained.

  Termination: By the LI, \( S_{n+1} = S \cap 1, \ldots, n = S \), where \( \sum_{i \in S} a_i = t \), if such an \( S \) exists. Thus if \( \sum_{i \in S_{n+1}} a_i = t \), a solution has been found and we rightly return true. Otherwise no solution exists and we rightly return false.

3. (30 marks) **SubsetSum (dynamic programming)** Now suppose that the \( a_i \) values are arbitrary. Design a dynamic programming algorithm to solve the **SubsetSum** problem. The running time of your algorithm should be polynomial in both \( n \) and \( t \).

(a) (10 marks) Give the definition of the array \( A \) you will use to solve this problem and state how you find out if there is such a set \( S \) from that array.

• Answer:
  \( A[i, \tau] \) is a 2D array over \( i \in [0, \ldots, n] \), \( \tau \in [0, \ldots, t] \), specifying the maximum \( \sum_{j \in S} a_j \) such that \( \sum_{j \in S} a_j \leq \tau \) and \( j \leq i \forall j \in S \). An \( S \) such that \( \sum_{i \in S} a_i = t \) exists iff \( A[n, t] = t \).

(b) (10 marks) Give the recurrence to compute the elements of the array \( A \), including initialization.

• Answer:
  \[ A[i, \tau] = A[i-1, \tau] \text{ if } a_i > \tau \]
  \[ = \max\{A[i-1, \tau], a_i + A[i-1, \tau - a_i]\} \text{ if } a_i \leq \tau \]

(c) (5 marks) State how you would recover the actual set \( S \) given \( A \).  

• Answer:
  PrintOpt(i, t)
  if \( i = 0 \) return
  if \( A[i, t] = A[i-1, t] \)
  PrintOpt (i - 1, t)
  Else
  PrintOpt(i - 1, t - a_i)
Print $a_i$

(d) (5 marks) Analyze the running time of your algorithm (including the step reconstructing $S$), in terms of $n$ and $t$.

- Answer: Initialization: $\Theta(n) + \Theta(t)$.
  Construction of $A: \Theta(nt)$.
  Recovery of $S: \Theta(n)$.
  Total Running Time: $\Theta(nt)$. 