1. Partition the range \([0, R]\) into \(n\) equal intervals: \([0, \frac{R}{n}), [\frac{R}{n}, \frac{2R}{n}), \ldots, [R - \frac{R}{n}, R]\). Since the keys are uniformly distributed in the range \([0, R]\), we expect that the number of keys falling into any of these \(n\) intervals is \(O(1)\). We create an array \(A[1 : n]\) of lists which are empty to begin with. We then scan through the input keys one at a time. Each key is inserted into the list corresponding to its interval. For example, if the key \(k\) has a value in the interval \([i\frac{R}{n}, (i + 1)\frac{R}{n})\), for some value of \(i\), then \(k\) is inserted into the list \(A[i + 1]\). After inserting the keys into their corresponding lists, we sort each list. Since each list has an expected \(O(1)\) number of keys, all the \(n\) lists can be sorted in a total expected time of \(O(n)\). Finally, we output the sorted lists \(A[1], A[2], \ldots, A[n]\). The total time spent is \(O(n)\) in expectation.

2. Find the median \(M\) of \(X\) in \(O(n)\) time. Partition \(X\) into \(X_1 = \{q \in X : q \leq M\}\) and \(X_2 = \{q \in X : q > M\}\). At the end of this partitioning we will get to know which of \(X_1\) and \(X_2\) has the element that is repeated thrice. We then proceed recursively with the sequence that has the element of interest. Note that \(X_1\) and \(X_2\) have very nearly \(\frac{n}{2}\) elements each.

If \(T(n)\) is the run time of this algorithm, then we have: \(T(n) = T(n/2) + O(n)\) that solves to: \(T(n) = O(n)\).

3. To begin with the edge \((3, 4)\) enters the tree. From out of all the edges going out of the nodes 3 and 4, the edge \((4, 5)\) has the least weight and hence it enters the tree next. From out of the edges going out of the nodes 3, 4, and 5, the edge \((2, 4)\) has the least weight. Thus the edge \((2, 4)\) enters the tree next. Proceeding in a similar manner we end up with a minimum spanning tree that has the following edges: \((3, 4), (4, 5), (2, 4), (4, 7), (1, 3)\) and \((1, 6)\). The total weight is 21.

4. One possible greedy algorithm will sort the objects in nondecreasing order of their weights and pick the \(r\) smallest objects whose total weight is \(\leq m\) and the total weight of the \((r + 1)\) smallest objects is \(> m\). The run time of this algorithm is \(O(n \log n)\).

Let \(R = \{R_1, \ldots, R_n\}\) be the output of greedy algorithm, and \(S = \{S_1, \ldots, S_s\}\) be that of an optimal strategy in nondecreasing order of object weight. Let \(i\) be the least index in which these two differ. Clearly, \(R_i < S_i\). Also note that \(R_i \notin S\). Thus we can insert \(R_i\) into \(S\) before \(S_i\) and delete \(S_i\) if there is a need. The new solution has at least the same number of objects as before. Proceeding in this fashion, we can make the first \(r\) objects of \(S\) the same as the corresponding \(r\) objects of \(R\). There cannot be space left in the knapsack for any more objects. If there were, the greedy algorithm would have added at least one more object. Thus \(s \leq r\).

If each \(w_i > m\), the utilization ratio is 0.

5. (a) \(f_i(y) = \max\{f_{i-1}(y), \max_{k, y \geq k w_i} \{f_{i-1}(y - k w_i) + kp_i\}\}\).

(b) If we fix \(i\) and \(y\), \(f_i(y)\) can be computed in \(O(m)\) time. But \(i \in [1..n], y \in [1..m]\). Thus the running time is \(O(m^2 n)\).

(c) For \(i \in [1..n]\), calculate \(p_i/w_i\). As a result, find the object \(k\) with the maximum profit density. Fill the knapsack with this object. Then \(x_k = m/w_k\), and the total profit is \(\frac{m}{w_k} p_k\).

6. Note that if the input graph is a tree then it will have exactly \(|V| - 1\) edges. To begin with check if this is the case. This can be done by checking if there are \(2(|V| - 1)\) entries in the adjacency lists. This checking can be done in \(O(|V|)\) time. If this condition is not satisfied, output: "The input graph is not a tree" and quit. Otherwise continue to perform either a BFS
or DFS in $O(|V|)$ time. At the end of this search if there is only one connected component, output: "The input graph is a tree" else output: "The input graph is not a tree". The total run time is $O(|V|)$. 