1. We use the integer sorting (or radix sorting) algorithm here. Let \( X = k_1, k_2, \ldots, k_n \) be the given input sequence. Each \( k_i \) can be thought of as a binary string of length \( \log n \sqrt{\log n} \). We employ LSB first radix sorting. In each stage we sort the keys with respect to \( \log n \) bits just like we discussed in class. Each stage of sorting takes \( O(n) \) time (using bucket sort) and hence the entire algorithm runs in time \( O(n \sqrt{\log n}) \) time.

2. Here we utilize the linear time selection algorithm. Find the median \( A \)

3. Using the repeated squaring trick, we can compute \( A_i \), using only \( O(\log n_i) \) matrix multiplications, for \( 1 \leq i \leq k \). Each matrix multiplication takes \( O(n^{\log_2 7}) \) time, using Strassen’s algorithm. Thus we can compute \( A_i \) in \( O(n^{\log_2 7} \log n_i) \) time, for \( 1 \leq i \leq k \). As a result, we can compute \( A_{\hat{i}} \), \( A_{\hat{2}}, \ldots, A_{\hat{k-1}}, \text{ and } A_{\hat{k}} \) in a total of \( O \left( n^{\log_2 7} (\log n_1 + \log n_2 + \cdots + \log n_k) \right) = O \left( n^{\log_2 7} \log N \right) \) time. These \( k \) matrices can then be multiplied in a total of \( O(n^{\log_2 7}k) \) time. Note that \( \log N = \Omega(k) \) since each \( n_i \) is \( \geq 2 \) \((1 \leq i \leq k)\).

Put together, the total run time is \( O \left( n^{\log_2 7} \log N \right) \).

4. Sort the jobs (in non-decreasing order) based on the time that they need for completion. Select the maximum possible number of jobs in the sorted order such that the sum of their running times does not exceed the global deadline \( D \).

\[ \text{Complexity} = \text{Complexity of sorting} + \text{Complexity of scanning} = O(n \log n) \]

**Proof of optimality:** Let \( g_1, g_2, \ldots, g_m \) be the jobs selected in that order by the greedy strategy. Let \( o_1, o_2, \ldots, o_k \) be the optimal solution. Let \( i \) be the smallest index where the greedy strategy and the optimal strategy differ. That is, \( g_1, g_2, \ldots, g_{i-1} \) are the same as \( o_1, o_2, \ldots, o_{i-1} \) and \( g_i \neq o_i \). Then, do the following: replace \( o_i \) in the optimal solution by \( g_i \). Since the greedy strategy has selected elements in sorted order, \( T_{g_i} \leq T_{o_i} \). Hence, replacing \( o_i \) by \( g_i \) does not increase the total time taken by all the jobs in the optimal solution. By repeating the above procedure at all the indices where the jobs in the greedy solution and the optimal solution differ, we can see that the greedy solution can complete at least as many jobs as the optimal solution. Hence, the greedy solution is optimal.

5. The weight of each object is \( \geq \epsilon m \). Since the knapsack capacity is \( m \), a maximum of \( 1/\epsilon \) objects can be included in the solution. The total number of subsets with \( 1/\epsilon \) objects or less is \( \sum_{i=1}^{1/\epsilon} \binom{n}{i} = O(n^{1/\epsilon}) \). Compute the profit for each such subset and pick the best subset.

\[ \text{Complexity} = O(n^{1/\epsilon}) \]

6. Let \((i,j)\) be any directed edge in the graph with a weight of \( w \). Replace this edge with a directed path of length \( w \) by introducing \( w-1 \) new nodes. Do this for every edge in the graph. Let the resultant graph be \( G'(V', E') \). Note that \( |V'| \leq C|V| \) and \( |E'| \leq C|E| \).

Perform a BFS on \( G'(V', E') \) starting from the source node \( s \). If a node \( u \in V \) is visited for the first time in step \( k \), then the shortest path from \( s \) to \( u \) has a weight of \( k \). By the end of the BFS, we would have determined the shortest paths from \( s \) to every node in \( V \).

Clearly, BFS takes time \( O(|V| + |E|) = O(|V| + |E|) \).