

1. Matrices A and B could be divided into k submatrices of size $n \times n$, A_1, \dots, A_k and B_1, \dots, B_k respectively. Then $AB =$

$$\begin{pmatrix} A_1 \\ A_2 \\ \dots \\ A_k \end{pmatrix} \begin{pmatrix} B_1 & B_2 & \dots & B_k \end{pmatrix} = \begin{pmatrix} A_1B_1 & A_1B_2 & \dots & A_1B_k \\ A_2B_1 & A_2B_2 & \dots & A_2B_k \\ \dots & \dots & \dots & \dots \\ A_kB_1 & A_kB_2 & \dots & A_kB_k \end{pmatrix}$$

Thus AB could be found by computing $A_iB_j, 1 \leq i, j \leq k$. Each of A_iB_j can be found by using Strassen's algorithm in $O(n^{\log 7})$ time, thus total time is $O(k^2 n^{\log 7})$.

2. Checking whether the array is distinct or not can be done by sorting the array and scanning through it. We can employ RadixSort to sort the array. The idea is the same as the one presented in class. Each input key has $\log N$ bits. We partition the keys into $\frac{\log N}{\log n}$ parts, each part consisting of $\log n$ bits. In phases we sort the keys. Each phase involves sorting the keys with respect to a single part. Each phase takes $\Theta(n)$ time (as was proven in class). There are $\frac{\log N}{\log n}$ phases.

Note that the space used by RadixSort is $O(n)$.

3. One possible minimum spanning tree has the following edges: $(3, 4), (4, 5), (5, 2), (2, 1), (1, 7)$ and $(7, 6)$. The total weight is 21.
4. At the beginning of the algorithm $dist[s] = 0; dist[1] = 6; dist[4] = 3; dist[5] = 5; dist[2] = dist[3] = \infty$.

In stage 1, node 4 has the minimum $dist$ value and hence is inserted into the set S . Nodes 2, 3 and 5 are the neighbors of 4 and hence we have to check if the $dist$ values of these nodes have to be modified. Since $dist[2] > dist[4] + W(4, 2)$, we change $dist[2]$ to 5. Likewise we set $dist[3] = 7$ and $dist[5] = 4$.

In stage 2, node 5 has the minimum $dist$ value and hence is inserted into S . Node 2 is the only neighbor of 5 but its $dist$ value does not change.

In stage 3, node 2 has the minimum $dist$ value and it becomes a part of S . Node 3 is the only neighbor of 2. The new $dist$ value of 3 becomes 6.

In stage 4, nodes 1 and 3 have the same minimum $dist$ value. One can be chosen arbitrarily. Say we pick node 3. Node 2 is the only neighbor and hence the $dist$ value of node 1 does not change.

In stage 5, the node 1 also enters S . Algorithm terminates then.

Thus the shortest paths from s to the nodes 1, 2, 3, 4, and 5 are 6, 5, 6, 3 and 4, respectively.

5. Let $k = \lfloor \frac{m}{w} \rfloor$. Assume w.l.o.g. that the profits of the objects are distinct. Optimal knapsack contains k objects whose profits are the largest. Find the object that has the k^{th} largest profit. Let its profit be P . This object is found using a linear time selection algorithm. Scan through the input objects and fill the knapsack with only those whose profits are $\geq P$. This algorithm takes $O(n)$ time.

Proof of optimality: let $X = (x_1, x_2, \dots, x_n)$ be an optimal solution and $Y = (y_1, y_2, \dots, y_n)$ be a solution found by the algorithm above, both solutions are sorted in

the descending order of p_i 's. Assume that $X \neq Y$, i.e., there exists an index i s.t. $x_i \neq y_i, x_j = y_j, 1 \leq j \leq i - 1$. Since $|X| = |Y| = k, i \leq k$ and due to the greedy nature of our algorithm $y_i = 1$, implying that $x_i = 0$. Also there exists a $t > k$ s.t. $x_t = 1$. Since $p_i > p_t$, replacing the t^{th} object in the optimal solution by the i^{th} object would improve the optimal one, a contradiction.

6. Let $P(i, k)$ be 1 if there exists a subset whose sum is k from among the first i items and zero otherwise. We are interested in computing $P(n, K)$. A recurrence relation for $P(i, k)$ is given by:

$$P(i, k) = 1 \text{ iff either } P(i - 1, k) = 1 \text{ or } P(i - 1, k - k_i) = 1$$

We can use the above recurrence relation to compute $P(n, K)$ in $O(nK)$ time. For example we can compute the following sequence: $P(1, 1), P(1, 2), \dots, P(1, K), P(2, 1), P(2, 2), \dots, P(2, K), \dots, P(n, 1), \dots, P(n, K)$.