Solutions to the problems of Exam I

1 *Apply the Master Method to solve the recurrence:*

$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n \log n$$

$\boxed{\text{Solution}}$

The Master Method is applicable to the recurrences of the kind $T(n) = a \cdot T(n/b) + f(n)$, where a and b are constants not depending on n. In our case $a = b = \sqrt{n}$ are not constants. Hence, the Master method cannot be <u>directly</u> applied.

To work around the difficulty with b, let us introduce a new variable $m = \log_2 n$, so $n = 2^m$. In this case $\sqrt{n} = \sqrt{2^m} = 2^{m/2}$. Hence, if we denote $Q(m) = T(2^m)$, one has in terms of m:

$$\begin{aligned} P(2^m) &= 2^{m/2} \cdot P(2^{m/2}) + m2^m, \quad \text{or} \\ Q(m) &= 2^{m/2} \cdot Q(m/2) + m2^m. \end{aligned}$$

This way we got $b = 2$, but $a = 2^{m/2}$ is still not a constant. To work around this problem, divide both parts of the last equation by 2^m:

$$\frac{Q(m)}{2^m} = \frac{Q^{m/2}}{2^{m/2}} + m,$$

and denote $R(m) = \frac{Q(m)}{2^m}$. One has

$$R(m) = R(m/2) + m.$$

The recursion for $R(m)$ can be solved with the help of the Master Method. In this case $a = 1$, $b = 2$, so case (3) applies. One has $R(m) = \Theta(m)$. This implies $Q(m) = R(m) \cdot 2^m = \Theta(m \cdot 2^m)$. Finally, $T(n) = Q(\log_2 n) = \Theta(n \log n)$. □

2 *Suppose that in a 0-1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give a greedy-type algorithm to find an optimal solution to this variant of the knapsack problem, and argue that your algorithm is correct.*

$\boxed{\text{Solution}}$

Let $i_1, \dots, i_n$ be the items with values $v_1, \dots, v_n$ and $w_1, \dots, w_n$ be their weights. Let W be the maximum knapsack weight. We have:

$$w_1 \le w_2 \le \cdots \le w_n \qquad (1)$$
$$v_1 \ge v_2 \ge \cdots \ge v_n. \qquad (2)$$

The following linear-time algorithm does the job.

Algorithm 1 Knapsack 1

```
$w = 0$              // knapsack weight
$S = \emptyset$              // knapsack content
 for ($i = 1$; $i \le n$; $i$++)
     if ($w + w_i \le W$)
         $w$ += $w_i$
         $S = S \cup \{i\}$
```

Proof of correctness:
A proof of correctness of a general greedy algorithm usually consists of two steps.

The greedy choice property. Let S be an optimal knapsack load. We show that without loss of generality one can assume $i_1 \in S$. Indeed, if $i_1 \notin S$, let k be the smallest index of an item of S. Consider the packing $S' = (S \setminus i_k) \cup i_1$. Since $w_1 \le w_k$, we have $w(S') \le w(S) \le W$, so S' is a legal packing. On the other hand, $v_1 \ge v_k$ implies $v(S') \ge v(S)$, so S' is also optimal.

The optimal substructure property. For an optimal packing S with $i_1 \in S$, the packing $S'' = S \setminus i_1$ is optimal for the items $i_2, \dots, i_n$ and $W'' = W - w_1$. Indeed, if S'' is not optimal, that one can improve the original packing S by improving S''. □

3 *Give a dynamic programming algorithm for solving the 0-1 Knapsack problem in the case if you have unlimited number of copies of each objects and you can pack multiple copies of an object into the knapsack. The running time should be $O(nW)$. Prove that your algorithms works correctly.*

$\boxed{\text{Solution}}$

Let $K(w)$ denote the maximum value of a knapsack of capacity w to be filled with items of weights $w_1, \dots, w_n$ and values $v_1, \dots, v_n$ so that the total weight on the items in the knapsack does not exceed w. If an optimal packing involves an item i, then removing this item leads to an optimal packing of a knapsack of capacity $K(w - w_i)$. Hence,

$$K(w) = K(w - w_i) + v_i$$

for some i. To find the i we try all possibilities, which leads to a recursion

$$K(w) = \max_{i\,:\,w_i \le w} \{K(w - w_i) + v_i\}$$

with the initial condition $K(0) = 0$. The algorithm then follows immediately.

Algorithm 2 KNAPSACK 2

```
$K(0) = 0$                         // knapsack value
$S = \emptyset$                         // knapsack content (a multiset)
 for ($w = 1$; $w \le W$; $w$++)  do
     $K(w) = K(w - 1)$
     ind = 0                      // index for which the max is attained
```

```
        for ($i = 1$; $i \le n$; $i$++) do        // computing the max in the recursion
            if ($w_i \le w$ && $K(w) > K(w - w_i) + v_i$) then
                ind = $i$
        if (ind > 0) then                        // if an improvement of $K(w)$ is possible, do it
            $K(w) = K(w - w_{\text{ind}}) + v_{\text{ind}}$
            $S = S \cup \{\text{ind}\}$
    return $K(W)$ and $S$
```

The algorithm involves two nested loops and all other operations take a constant time. So, its running time is the product of the number of times each loop is running, which is $O(nW)$.

4 *Let $G = (V, E)$ be a weighted, directed graph with source vertex s, and let G be initialized by* INITIALIZE-SINGLE-SOURCE(G, s). *Prove that if a sequence of relaxation steps sets $\pi[s]$ to a non-*NIL *value, then G contains a negative-weight cycle.*

Solution

After applying the INITIALIZE-SINGLE-SOURCE(G, s), for the vertex s one has $d[s] = 0$ and $d[v] = \infty$ for all other vertices, and $\pi[v] =$NIL for any vertex. If $\pi(s)$ is set to a non-NIL value, this can be only a result of relaxation of an edge (u, s). Let u be the first such vertex. One has

$$d[s] > d[u] + w(u, s).$$

Since $d[s] = 0$ prior to this relaxation, the value $d[u]$ is finite. This, in turn, implies that there is an oriented path $s \leadsto u$ consisting of vertices with final labels. Let $p_0 = s$, $p_k = u$ and $p_1, \dots, p_{k-1}$ be the remaining vertices of this path. Then every edge (p_{i-1}, p_i) $(i = 1, \dots, k)$ was relaxed prior to the relaxation of the edge (u, s), so the values $d[p_i]$ are all finite. One has

$$\begin{array}{rcl} d[p_1] & = & d[p_0] + w(p_0, p_1) \\ d[p_2] & = & d[p_1] + w(p_1, p_2) \\ \dots & & \dots \\ d[p_k] & = & d[p_{k-1}] + w(p_{k-1}, p_k) \\ d[p_0] & > & d[p_k] + w(p_k, p_0). \end{array}$$

Summing up these equalities and the inequality results in

$$\sum_{j=0}^{k} d[p_j] > \sum_{j=0}^{k} d[p_j] + \sum_{j=1}^{k} w(p_{j-1}, p_j) + w(p_k, p_0).$$

This, in turn, implies

$$0 > \sum_{j=1}^{k} w(p_{j-1}, p_j) + w(p_k, p_0).$$

That is, the sum of the edge weights of the cycle $p_0, p_1 \dots, p_k, p_0$ is negative. $\square$