Homework 6

Instructor: Rotem Oshman
Name: Ishan Pranav

References

Collaborated with Crystal Huang.

Question 1: Knapsack with repetitions

In the lecture, you have seen the Knapsack problem: Given n items with non-zero integer weights $w_i \in \mathbb{Z}^+$ and values $v_i \geq 0$, you want to choose a subset that maximizes the total value while the overall weight does not exceed a given bound W. In the following, we want to consider a related problem where we assume that you are given m_i many copies of each item type. In other words, w_i and v_i describe the weight and value of an item type, and you can choose that item up to m_i times.

Given weights $w[1, \ldots, n]$, values $v[1, \ldots, n]$, inventory $m[1, \ldots, n]$, and an overall weight W, we want to maximize the overall value subject to the bound W.

1. Let $K(i, u)$ denote the most valuable solution using items $1, \ldots, i$ and weight bound u. Write a recursive formula for K, do not forget the base cases.

Solution: *Claim.* Given weights $w_1, \ldots, w_i$, values $v_1, \ldots, v_1$, and inventory $m_1, \ldots, m_i$,

$$K(i,u) = \begin{cases} 0, & i = 0, \\ \max\limits_{0 \leq j \leq \min\left(m_i, \frac{u}{w_i}\right)} \left(K(i-1, u - jw_i) + jv_i\right) & i > 0. \end{cases}$$

where $K(i, u)$ denotes the most valuable solution using items $1, \ldots, i$ and weight bound $u \geq 0$.

Proof. We want to show that $K(i, u)$ denotes the most valuable solution using items $1, \ldots, i$ and weight bound $u \geq 0$. We can demonstrate the claim by induction on i.

Basis. Consider $i = 0$. For all $u \geq 0$, the most valuable solution using the first 0 items is 0. Since $K(0, u) = 0$ for all $u \geq 0$, the claim holds in the base case.

Hypothesis. Consider $i = k$ where $k > 0$. Assume that $K(k, u)$ denotes the most valuable solution using items $1, \ldots, k$ for all weight bounds $u \geq 0$.

Inductive step. Consider $i = k + 1$. The most valuable solution maximizes the total value by considering each feasible quantity j of item $k + 1$.

The minimum quantity of item $k + 1$ is 0. The maximum quantity of item $k + 1$ available in the inventory is given by m_{k+1}. However, the maximum quantity of item $k + 1$ that can fit within a weight constraint of u is $\frac{u}{w_{k+1}}$. Thus, the upper bound for j is the smaller of m_{k+1} (maximum units available in the inventory) and $\frac{u}{w_{k+1}}$ (units satisfying the weight bound), or $\min\left(m_{k+1}, \frac{u}{w_{k+1}}\right)$.

For each such quantity j, the most valuable solution is the total weight of the j-many $(k + 1)$-type items used, plus the most valuable solution using all the previous items $1, \ldots, k$ under a smaller weight bound that excludes the total weight of the $(k + 1)$-type items used.

From the inductive hypothesis, $K(k, u - jw_{k+1})$ gives the most valuable solution for items $1, \ldots, k$ under a weight bound, u, less the total weight of the j-many $(k+1)$-type items, jw_{k+1}.

Thus, the expression $K((k+1)-1, u - jw_{k+1}) + jv_{k+1}$ gives the most valuable solution with j-many $(k+1)$-type items for all $u \geq 0$.

Since $k+1 \geq 0$, we know that for all $u \geq 0$, the recurrence $K(k+1, u)$ gives the maximum value considering each feasible quantity of item $k+1$. This corresponds to the most valuable solution, thus completing the inductive step.

Hence, by the principle of mathematical induction, $K(i, u)$ gives the most valuable solution using items $1, \ldots, i$ and weight bound $u \geq 0$. $\square$

2. Write an algorithm $\textsc{MaxValue}(W, n, w, v, m)$ that computes the optimal value using Dynamic Programming. What is the running time of your algorithm?

Solution:

Algorithm. $\textsc{MaxValue}(W, n, w, v, m)$ with an overall weight bound $W \in \mathbb{Z}^+$, a number of items n, an n-element array of weights $w[1, \dots, n]$ such that $w[i] \in \mathbb{Z}^+$ for $1 \leq i \leq n$, an n-element array of values $v[1, \dots, n]$, and an n-element array of quantities $m[1, \dots, n]$; returns the optimal knapsack value:

```
Initialize a two-dimensional array $K[0, \dots, n][0, \dots, W]$ with dimensions $(n+1) \times (W+1)$.
For $u = 0$ to $W$, assign $K[0][u] \leftarrow 0$.
For $i = 1$ to $n$:
  • for $u = 1$ to $W$:
      – let $a \leftarrow 0$;
      – for $j = 0$ to $\min\left(m[i], \frac{u}{w[i]}\right)$:
          ∗ assign $a \leftarrow \max(a, K[i-1][u - j \cdot w[i]] + j \cdot v[i])$.
      – assign $K[i][u] \leftarrow a$.
Return $K[N][W]$.
```

Proposition. *Claim.* $\textsc{MaxValue}(W, n, w, v, m)$ has running time

$$O\left(n \times W \times \max\left(\max_{1 \leq i \leq n}(m_i), W\right)\right).$$

Proof. First, the $\textsc{MaxValue}$ algorithm initializes $W + 1$ elements in K with the value 0, which is an $O(W+1) = O(W)$ process.

Then, for each of the n values of i, for each of the $W + 1$ values of u, the second loop takes the maximum of $\min\left(m[i], \frac{u}{w[i]}\right) + 1$ values. In the worst case, the running time of the innermost loop is

$$\begin{aligned} O\left(\max\left(m[i], \frac{u}{w[i]}\right) + 1\right) &= O\left(\max\left(m[i], \frac{u}{w[i]}\right)\right) && \textit{asymptotically,} \\ &= O(\max(m[i], W)) && \textit{in the worst case } u = W \textit{ and } w[i] = 1. \end{aligned}$$

Then, the running time of the second loop is

$$O\left(n \times W \times \max\left(\max_{1 \leq i \leq n}(m_i), W\right)\right).$$

This is the dominant term in the running time of $\textsc{MaxValue}$, and therefore the asymptotic running time of $\textsc{MaxValue}$. □

Question 2: Fractional knapsack

Now consider a variant of the (original) Knapsack problem where of each item you can take a fractional part $s_i \leq w_i$ and gain value $\frac{s_i}{w_i} \cdot v_i$. In the lecture you have seen a *greedy strategy* for this problem:

- Take as much as you can from the item i that maximizes $\frac{v_i}{w_i}$. In other words, if W is the current weight bound, take $s_i = \min(W, w_i)$ of that item.
- Afterwards, set $W' = W - s_i$, remove the item i from consideration. Repeat until $W = 0$ or there are no more items left.

In the following, we want to prove the correctness of this greedy strategy. That is, we want to prove that it maximizes the value. Assume without loss of generality that the items are sorted by decreasing $\frac{v_i}{w_i}$, i.e., that the greedy chooses the first element.

1. As a warm-up, show that the greedy strategy always outputs a solution that respects the weight bound W.

Solution:

Algorithm I. $\textsc{GreedyStrategy}(W, n, w, v)$ with an overall weight bound $W \in \mathbb{Z}^+$, a number of items n, an n-element list of weights $(w_1, \ldots, w_n)$ such that $w_i \in \mathbb{Z}^+$ for $1 \leq i \leq n$, an n-element list of values $(v_1, \ldots, v_n)$; return an n-element list of the fractional parts used, $(s_1, \ldots, s_n)$, such that $s_i \in \mathbb{Q}$ and $0 \leq s_i \leq w_i$.

Assume without loss of generality that the items are sorted by decreasing value-to-weight ratio, and that the algorithm chooses the first element.

Let $1 \leq j \leq (n+1)$ where $j = 1$ denotes the iteration before the loop begins, $j = i$ denotes the iteration before considering item i for $i \leq n$, and $j = n+1$ denotes the step immediately after the loop terminates.

Let W_j denote the remaining weight bound at the beginning of iteration j.

Invariant I. *Claim.* For every step of the loop $1 \leq j \leq (n+1)$, we claim that

$$W_j = W - \sum_{i=1}^{j-1} s_i.$$

Proof. We can prove this invariant by induction on j.

Basis. Consider $j = 1$. Note that we are considering the case that occurs before the first iteration of the loop.

From Algorithm I, before the loop begins, W has never been reduced and no value within s has been modified; that is, $W_1 = W$ and $s_i = 0$ for all $1 \leq i \leq n$.

Since the loop body has not yet executed:

$$\begin{aligned} W - \sum_{i=1}^{1-1} s_i &= W - 0 \\ &= W \\ &= W_1. \end{aligned}$$

Thus the loop invariant holds at initialization.

Hypothesis. Consider $j = k$ where $1 < k \leq n$. Assume that at the beginning of iteration k, we have

$$W_k = W - \sum_{i=1}^{k-1} s_i.$$

Inductive step. Consider $j = k + 1$. Note that this is the iteration immediately after considering item k and before considering item $k + 1$.

Observe

$$\begin{aligned} W_{k+1} &= W_k - s_k && \textit{from Algorithm I,} \\ &= \left(W - \sum_{i=1}^{k-1}(s_i)\right) - s_k && \textit{from the inductive hypothesis,} \\ &= W - \left(\sum_{i=1}^{k-1}(s_i) + s_k\right) \\ &= W - \sum_{i=1}^{k}(s_i) \\ &= W - \sum_{i=1}^{(k+1)-1}(s_i) && \textit{thus completing the inductive step.} \end{aligned}$$

Hence, by the principle of mathematical induction, we have for all iterations $1 \leq j \leq (n + 1)$,

$$W_j = W - \sum_{i=1}^{j-1} s_i.$$

From this result, we know that the loop invariant holds at initialization, maintenance, and termination.

Proposition I. *Claim.* GREEDYSTRATEGY(W, n, w, v) always respects the weight bound W.

Proof. Let $s =$ GREEDYSTRATEGY(W, n, w, v). We want to show that

$$\sum_{i=1}^{n} s_i \leq W.$$

Algorithm I terminates when its loop terminates; that is, it terminates when $W_j = 0$, or when $W_j > 0$ and $j = n + 1$:

- Suppose $W_j = 0$. At the beginning of iteration j, the weight bound was exhausted, so the algorithm terminates on iteration j and therefore $s_i = 0$ for all $j \leq i \leq n$. Observe

$$\begin{aligned}
W_j &= W - \sum_{i=1}^{j-1} s_i && \textit{from Invariant I,}\\
0 &= W - \sum_{i=1}^{j-1} s_i && \textit{in this case,}\\
W &= \sum_{i=1}^{j-1} s_i\\
W &= \sum_{i=1}^{n} s_i && \textit{since } s_i = 0 \textit{ for all } j \leq i \leq n,\\
\sum_{i=1}^{n} s_i &= W \leq W && \textit{so the claim holds in this case.}
\end{aligned}$$

- Suppose instead $W_j > 0$ and $j = n + 1$. At the beginning of iteration j, all items were considered, so the Algorithm terminates on iteration j.

$$\begin{aligned}
W_j &= W_{n+1} && \textit{in this case,}\\
W_j &= W - \sum_{i=1}^{(n+1)-1} s_i && \textit{from Invariant I,}\\
W_j &> 0 && \textit{in this case,}\\
W - \sum_{i=1}^{n} s_i &> 0\\
\sum_{i=1}^{n} s_i &< W \leq W && \textit{so the claim holds in this case.}
\end{aligned}$$

In all cases, the claim holds.

Therefore, regardless of when GreedyStrategy terminates, the algorithm respects the weight bound W. $\square$

2. Argue that if $\sum_{i=1}^{n} w_i \leq W$, the greedy strategy selects the optimal outcome.

Solution: Proposition II. *Claim.* If

$$\sum_{i=1}^{n} w_i \leq W,$$

then GreedyStrategy selects the optimal outcome.

Let $s = \textsc{GreedyStrategy}(W, n, w, v)$. Then there does not exist a solution s^* where

$$\sum_{i=1}^{n} \frac{s_i^*}{w_i} \cdot v_i > \sum_{i=1}^{n} \frac{s_i}{w_i} \cdot v_i.$$

Proof. Suppose that

$$\sum_{i=1}^{n} w_i \leq W.$$

Assume, for the sake of contradiction, that there exists a solution s^* where

$$\sum_{i=1}^{n} \frac{s_i^*}{w_i} \cdot v_i > \sum_{i=1}^{n} \frac{s_i}{w_i} \cdot v_i.$$

Note that since

$$\sum_{i=1}^{n} w_i \leq W,$$

the overall weight bound W is not a binding constraint for any s_i for $1 \leq i \leq n$. From Algorithm I, GreedyStrategy chooses $s_j = \min(W_j, w_j)$ on each iteration j. Since W is not a binding constraint, $s_j = w_j$. We can take the maximal amount of each item, knowing that the sum of these weights will not exceed the overall weight bound.

We conclude that $s = w$. However,

$$\begin{aligned}
\sum_{i=1}^{n} \frac{s_i^*}{w_i} \cdot v_i &> \sum_{i=1}^{n} \frac{s_i}{w_i} \cdot v_i && \textit{our hypothesis,} \\
&> \sum_{i=1}^{n} \frac{w_i}{w_i} \cdot v_i && \textit{since } s = w. \\
s_i^* &\geq w_i && \textit{for some } 1 \leq i \leq n.
\end{aligned}$$

This contradicts our hypothesis that s^* is a solution.

Hence, we reject the hypothesis and conclude that there exists no such solution s^* with greater value than s. Ergo the GreedyStrategy algorithm selects the optimal outcome. □

3. We call the selection $s = (s_1, \ldots, s_n)$ a strategy and let $\mathsf{value}(s) = \sum_{i=1}^{n} \frac{s_i}{w_i} \cdot v_i$ denote the obtained value. Let s^* denote an optimal strategy (an arbitrary one if multiple exist) and s the greedy strategy. Show the following: if $s(1) \neq s^*(1)$, then there exists a valid strategy s' with $\mathsf{value}(s') = \mathsf{value}(s^*)$ and $s'(1) = s(1)$. In other words, there exists an optimal strategy s' that agrees with the greedy one on the first choice.

Solution: Proposition III. *Claim.* Let $s = \textsc{GreedyStrategy}(W, n, w, v)$ and s^* denote an optimal solution. If $s_1 \neq s_1^*$, then there exists a solution s' where $s_1' = s_1$ and

$$\sum_{i=1}^{n} \frac{s_i'}{w_i} \cdot v_i = \sum_{i=1}^{n} \frac{s_i^*}{w_i} \cdot v_i.$$

Proof. Let $s = \textsc{GreedyStrategy}(W, n, w, v)$. Suppose s^* is an optimal solution where $s_1 \neq s_1^*$.

From Algorithm I, we assume without loss of generality that the items are sorted by decreasing value-to-weight ratio, and that $\textsc{GreedyStrategy}$ chooses the first element. Thus item 1 is the item with the maximum value-to-weight ratio: $\frac{v_1}{w_1} \geq \frac{v_i}{v_1}$ for all $1 \leq i \leq n$.

From Algorithm I, we know that $\textsc{GreedyStrategy}$ chooses $s_1 = \min(w_1, W_1)$. Likewise from Algorithm I, for iteration 1, we know that $W = W_1$.

There are two cases: either $w_1 \leq W$ or $w_1 > W$.

- Suppose $w_1 \leq W$. Then $s_1 = w_1$. Since $s_1^* \leq w_1$, we have $s_1^* \leq s_1$.
- Suppose $w_1 > W$. Then $s_1 = W$. Of course, s^* is a solution, so

$$\sum_{i=1}^{n} s_i^* \leq W.$$

  Thus $s_1^* \leq W$, and therefore, $s_1^* \leq s_1$.

In all cases, $s_1^* \leq s_1$. However, we supposed that $s_1^* \neq s_1$, so $s_1^* < s_1$.

We want to construct a solution s' with $s_1' = s_1$.

Note $(s_1' - s_1^*) = (s_1 - s_1^*)$ denotes the additional fractional amount allocated to s_1' compared to s_1^*. Of course $s_1^* < s_1$ implies $(s_1 - s_1^*) > 0$.

We can construct $s' = (s_1, s_2' \ldots, s_n')$ with total value

$$\frac{s_1}{w_1} \cdot v_1 + \sum_{i=2}^{n} \frac{s_i'}{w_i} \cdot v_i$$

where for all $2 \leq i \leq n$, we assign $s_i' = s_i^* - d_i$, choosing $0 \leq d_i \leq s_i^*$ such that

$$\sum_{i=1}^{n} \frac{s_i'}{w_i} \cdot v_i = \left[\frac{s_1}{w_1} \cdot v_1 + \sum_{i=2}^{n} \frac{(s_i^* - d_i)}{w_i} \cdot v_i \right] = \sum_{i=1}^{n} \frac{s_i^*}{w_i} \cdot v_i.$$

We know that

$$\sum_{i=2}^{n} d_i \geq (s_1 - s_1^*).$$

This is because item 1 has the maximum value-to-weight ratio: $\frac{v_1}{w_1} \geq \frac{v_i}{w_i}$ for all $1 \leq i \leq n$. We need to subtract at least as much *weight* from the other items collectively as we added to item 1 in order to offset the *value* contributed by the amount of item 1. Otherwise, the total value for s^* is not equal to the total value for s', contradicting the construction of d_i for some $2 \leq i \leq n$.

This implies that in s', relative to s^*, there is no net addition in terms of weight, so the overall weight bound W is respected.

Similarly, for each $2 \leq i \leq n$, we take $s'_i = (s^*_i - d_i)$. We know $s^*_i \leq w_i$, therefore $s'_i \leq w_i$ because $d_i \geq 0$. Of course, $s_1 \leq w_1$ as well, so the per-item weight bounds are met for all items in s'.

Thus, if there is an optimal solution s^*, it is possible to construct a valid optimal solution s' whose solution agrees with GreedyStrategy for the fractional portion of the first item. $\square$

4. Perform an inductive argument to argue that the greedy strategy is optimal overall.

Solution: Proposition IV. *Claim.* GreedyStrategy is optimal overall.

Let $s = \textsc{GreedyStrategy}(W, n, w, v)$ and s^* be an overall optimal solution. Then s is optimal overall.

Proof. Assume, for the sake of contradiction, that $s = \textsc{GreedyStrategy}(W, n, w, v)$ is not optimal. That is, assume s^* has total value greater than that of s:

$$\sum_{i=1}^{n} \frac{s_i^*}{w_i} \cdot v_i > \sum_{i=1}^{n} \frac{s_i}{w_i} \cdot v_i.$$

We want to show that we can construct an optimal solution s' where for all $1 \leq i \leq n$, we have $s_i = s'_i$. We can demonstrate the claim by induction on i.

Basis. Consider $i = 1$. From Proposition III, we know that we can construct an optimal solution s' for which $s_1 = s'_1$. Therefore, the claim holds in the base case.

Hypothesis. Consider $1 < i < k \leq n$. Assume that we can construct an optimal solution s' for which $s_j = s'_j$ for $1 \leq j \leq k$.

Inductive step. Consider $i = k$. Then $s = (s_1, \ldots, s_n)$ and $s^* = (s_1^*, \ldots, s_n^*)$.

Consider the subproblems $(s_k, \ldots, s_n)$ and $(s_k^*, \ldots, s_n^*)$. In these cases, item k is the item with the greatest value-to-weight ratio because s and s^* are sorted by value-to-weight, descending.

Now $s_k = s_k^*$ or $s_k \neq s_k^*$:

- Suppose $s_k = s_k^*$. Then we can take $s'_k = s_k^*$. It is possible to construct an optimal solution s' with this constraint because we know that the optimal solution s^* exists such that $s'_k = s_k^*$.
- Suppose instead $s_1 \neq s_1^*$. From Proposition III, the existence and optimality of solution s^* implies that we can construct an optimal solution s' where $s'_1 = s_1$ and

$$\sum_{i=1}^{k+1} \frac{s'_i}{w_i} \cdot v_i = \sum_{i=1}^{k+1} \frac{s_i^*}{w_i} \cdot v_i.$$

In all cases, assigning $s'_k = s_k$ does not compromise our ability to construct a solution for the subproblem $(s_k, \ldots, s_n)$ using the remaining items.

From the inductive hypothesis, we know that we have not compromised our ability to construct an optimal solution using the first $(k - 1)$ elements $(s'_1, \ldots, s'_{k-1})$ such that $s'_j = s_j$ for all $j \leq k$. That is, we can use the first $(k - 1)$ elements of s to construct an optimal solution.

Thus, we conclude that we have not compromised our ability to construct an optimal solution using the first k elements $(s_1, \ldots, s_k)$.

Hence, by the principle of mathematical induction, we can construct an optimal solution s' whose value is equal to that of s^* such that $s' = s$. We showed that we can use all n elements of s to construct an optimal solution whose value is that of s'.

However, since s' is optimal, the value of s' is that of s^*, which contradicts the hypothesis that s^* has a greater total value than s.

Ergo, s is optimal. $\square$

Question 3: Let's Paint a Fence!

We want to paint our new fence, which is made up N boards. The lengths of the N boards are given in an array $L[1, \ldots, N]$. We have hired K painters, and each painter takes 1 hour to paint a 1 unit of the board. For example, if one of the painters paints boards 3, 4 and 5, then they complete at time $t_i = L[3] + L[4] + L[5]$. Our goal is to assign each painter to some subset of boards, to minimize the time when the fence has been completely painted. Since the K painters can work in parallel, this corresponds to minimizing $\max(t_1, \ldots, t_K)$, where t_i is the time taken by painter i to complete their job. The painting task must be accomplished under the following constraints:

- Each board must be completely painted by exactly one painter; i.e., no board can be painted partially by one painter and partially by another.
- Each painter paints a contiguous collection of boards. For example, a configuration where painter 1 paints boards 1 and 3 but not 2 is not a valid solution.

You are given as input the following: the number of painters K, and an array L with the board lengths. In the following problems, denote by $T(i, j)$ the minimum time to paint the first i boards using j painters. Using Dynamic Programming, we want to come up with a procedure to minimize the painting time.

1. Write a recurrence for $T(i, j)$ in terms of $T(*, j-1)$, i.e., of entries $T(k, j-1)$ for arbitrary k.

Solution: *Claim.* Let $L[i]$ denote the length of board i for $1 \leq i \leq N$. Then for all $0 \leq i \leq N$,

$$T(i,j) = \begin{cases} 0, & i = 0, \\ +\infty, & i > 0 \text{ and } j = 0, \\ \displaystyle\sum_{k=1}^{i} L(k), & i > 0 \text{ and } j = 1, \\ \displaystyle\min_{0 \leq k \leq i} \left(\max \left(T(i-k, j-1), \sum_{m=i-k}^{i} L(m) \right) \right), & i > 0 \text{ and } j > 1. \end{cases}$$

denotes the minimum time required to paint the first i boards using $1 \leq j \leq K$ painters.

Proof. We can demonstrate the claim by induction on j.

Basis. Consider $j = 0$. Then $i = 0$ or $i > 0$:

- Suppose $i = 0$. There are no boards, so the minimum time required is 0. Of course, $T(0, 0) = 0$, so the claim holds in this case.
- Suppose instead $i > 0$. There are boards but no painters, so the minimum time required to paint the first i boards is unbounded ($+\infty$). Of course, $T(i, 0) = +\infty$ for all such i, so the claim holds in this case.

Consider $j = 1$. Then $i = 0$ or $i > 0$:

- Suppose $i = 0$. Then there are no boards, so the minimum time required is 0. Of course, $T(0, 1) = 0$, so the claim holds in this case.

- Suppose instead $i > 0$. Then there is only one painter, so the time required to paint the first i boards is the sum of the times required to paint each board. There is no parallelization in this case. Note $T(i, 1)$ gives this result for all $0 < i \leq N$, so the claim holds.

The claim holds in all base cases.

Hypothesis. Consider $j = \ell$ where $\ell > 1$. Assume for all $0 \leq i \leq N$ that $T(i, \ell)$ gives the minimum time required to paint the first i boards using ℓ painters.

Inductive step. Consider $j = \ell + 1$. For all $0 \leq i \leq N$, the optimal solution for i-many boards minimizes the required time by considering each feasible contiguous number of boards k that painter $(\ell + 1)$ may paint.

The minimum value of k is 0, since if $j > i$, a single painter may be assigned to paint no boards. The maximum value of k is i. Since $\ell > 1$, this case occurs when $i = 1$ so a single painter is assigned to paint all i boards.

For each such quantity of boards k, the minimum time required to paint the first k boards is the larger of the minimum time required for the single painter $(\ell + 1)$ to paint the last k-many boards, and of the minimum time required for the remaining ℓ painters to paint the first $(i - k)$ boards.

From the inductive hypothesis, $T(i-k, (\ell+1)-1)$ gives the minimum time required for ℓ-many painters to paint $(i - k)$-many boards and

$$\sum_{m=i-k}^{i} L(m)$$

gives the minimum time required for 1 painter to paint the last k-many boards.

Thus, the expression

$$\max\left(T(i-k, \ell), \sum_{m=i-k}^{i} L(m)\right)$$

gives the optimal solution where painter $(\ell + 1)$ paints the last k-many boards.

Since $\ell + 1 \geq 1$, we know that for all $0 \leq i \leq N$, the recurrence $T(i, \ell+1)$ gives the minimum value considering each feasible number of boards k. This corresponds to the optimal solution, thus completing the inductive step.

Hence, by the principle of mathematical induction, $T(i, j)$ gives the minimum time required to paint the first $0 \leq i \leq N$ boards using $1 \leq j \leq K$ painters. $\square$

2. Write an algorithm MinTime that computes the minimal completion time. To this end, complete the following skeleton. Make sure your algorithm uses only the minimal amount of extra storage.

```
1 MinTime($N, K$)
2     $T \leftarrow$ new array of length .........
3     $T[\dots\dots] \leftarrow$ ......
4     for ...... = 1 to ...... do
5         $T[\dots\dots] \leftarrow$ .........
6     for ...... = 1 to ......... do
7         for ...... = ...... downto 1 do
8             $T[\dots\dots] \leftarrow$ Calc($T, L, N, K, i, j$)
9     return $T[\dots\dots]$
```

Solution:

Algorithm 1. MinTime(L, N, K) with an N-element array $L[1, \dots, N]$, where $L[i]$ denotes the length of board $1 \leq i \leq N$, and a number of painters $K \in \mathbb{N}$; return the minimum time required to paint N boards with K painters:

If $i = 0$, then return 0.

Otherwise, if $j = 0$, then return $+\infty$.

Initialize two N-element arrays $T_0[1, \dots, N]$ and $T_1[1, \dots, N]$.

Assign $T_0[0] \leftarrow 0$.

For $i = 1$ to N, assign $T_0[i] \leftarrow T_0[i-1] + L[i]$.

For $j = 1$ to K:

- for $i = 0$ to N, assign $T_1[i] \leftarrow$ Calc(T_0, L, N, K, i, j);
- swap the references T_0 and T_1.

Return $T_0[N]$.

Proposition 1. *Claim.* MinTime uses the minimal amount of extra storage.

Proof. MinTime uses a constant amount of extra space for its non-array local variables as well as N elements for T_0 and N elements for T_1.

Asymptotically, the space complexity of MinTime is

$$O(1) + 2 \cdot O(N) = O(N).$$

For each of the K painters where $K > 1$, the optimal solution depends upon a prior solution for $K - 1$ painters. Therefore, in a bottom-up approach, for each unique number of boards N, we only need to allocate space for one subproblem.

However, for each of the N possible numbers of boards, the optimal solution depends on a prior solution for some number of boards $0 \leq i \leq N$. Therefore, in a bottom-up approach, for each unique number of painters, it is not possible to allocate space for fewer than N subproblems.

Homework 6

Using this approach, the minimal amount of extra storage for this problem is $O(N \times 1) = O(N)$.

Since the space complexity of MinTime is $O(N)$, we conclude that MinTime uses the minimal amount of extra storage. $\square$

3. Next, write an algorithm $\textsc{Calc}(T, L, N, K, i, j)$ that calculates $T[i, j]$ according to your recursive formula from part (1). Your algorithm should run in time $O(N)$.

Solution:

Algorithm 2. $\textsc{Calc}(T, L, N, K, i, j)$ given an N-element array $T[1, \ldots, N]$, an N-element array $L[1, \ldots, N]$, where $L[\ell]$ denotes the length of board $1 \leq \ell \leq N$, the subproblem number of boards $1 \leq i \leq N$, and the subproblem number of painters $1 \leq j \leq K$; returns the minimum time required for j painters to paint i boards:

```
Let $s \leftarrow 0$.
Let $a \leftarrow +\infty$.
For $k = 0$ to $i - 1$:
    • assign $s \leftarrow s + L[i - k]$;
    • assign $a \leftarrow \min(a, \max(T[i - k], s))$.
Return $a$.
```

Proposition 2. *Claim.* $\textsc{Calc}(T, L, N, K, i, j)$ has running time $O(N)$.

Proof. First, the algorithm assigns $+\infty$ to a and 0 to s, which are $O(1)$ processes.

Then, for each of the i indices from $k = 0$ to $k = i - 1$, the algorithm assigns computed values to a and s. The loop body computation is an $O(1)$ process. The maximum value for i is N. Thus, the worst-case running time of the loop is $O(N)$.

Therefore, the running time of $\textsc{Calc}(T, L, N, K, i, j)$ is $O(1) + O(N) = O(N)$. □