1 Exercise 4.9

We are given 3 machines that are identical i.e. same processing power, with n jobs that need to be processed. Each job j takes p_j processing time. Each machine can process at most one job at a time. And the processing time is independent on the job itself.

The goal is to assign the jobs to the three machines in such a way that the maximum load of the three machines is minimized. That is , let S_i denote the set of jobs assigned to machine i ($i = 1, 2, 3$), where the objective is: minimize $\max\{\sum_{j\in S_1} p_j, \sum_{j\in S_2} p_j, \sum_{j\in S_3} p_j\}$.

We can solve this problem by a Dynamic Program. Consider the following sub-problem. Given:

- Let job i be, $i \in \{1, ..., n\}$
- Let v_1 be $v_1 \in \{0, ..., \frac{1}{3}\sum_j^n p_j\}$
- Let v_2 be $v_2 \in \{0, ..., \frac{1}{3}\sum_j^n p_j\}$

Then there exists, or we need to make, a set $S_m \subseteq \{1, ..., i\}$ for machine every machine m with $|S_m| = k$ and $v = \sum_{j\in S_m} p_j$. In other words, for every machine check if we can process exactly $\frac{1}{3}$ items, as we want to minimize the maximum, we check if the sum of the processed jobs are also of a third of the weight. Next, we have the following function:

$$F(i, v_1, v_2) = \begin{cases} 1 & \text{if, } \exists \mathrm{S}_1 \subseteq \{1, ..., k\},\ S_2 \subseteq \{1, ..., i-k\} \\ \text{such that sum of each subset equals } v_1 and v_2 & \\ 0 & \text{otherwise.} \end{cases}$$

Now that we have identified the sub-problem and the function, we follow with the initialization of the algorithm, the recursion and the final output.

Algorithm 1 Dynamic program

Initialization:

for l in $\{1, ..., m-1\}$ **do** $\mathcal{O}(m-1)$

for v in $\{0, ..., B = \frac{1}{3}\sum_j p_j\}$ **do** $\mathcal{O}(B)$ $v = 0$ or $v = p_1$: $\mathcal{O}(1)$

$F(l, 1, v, k) = 1$ $\mathcal{O}(1)$

$F(l, 1, v, k) = 0$ $\mathcal{O}(1)$

Algorithm 2 Recursion

for m in $\{1, ..., m-1\}$ **do** $\mathcal{O}(m-1)$

  for i in $\{2, ..., n\}$ **do** $\mathcal{O}(n-1)$

    for v in $\{0, ..., B\}$ **do** $\mathcal{O}(B)$

      if $p_i > v$: **then** $\mathcal{O}(1)$

        $F(m, i, v, k) = F(m, i-1, v, k)$ $\mathcal{O}(1)$

      end if

      if $p_i \leq v$: **then** $\mathcal{O}(1)$

        $F(m, i, v, k) = F(m, i-1, v, k)$ or $F(m, i-1, v-p_i, k-1)$ $\mathcal{O}(1)$

      end if

    end for

  end for

  update*

end for

Output:

You want to return $F(l, n, v, k) = 1$ ($\mathcal{O}(1)$) where v is the largest. Once you know the set, we can update* the new list of possible for jobs for the next iteration.

For example, whatever we get for S_1 we remove it from the set of all possible jobs, $S_n = S_n \setminus S_l$, and change $n = |S_n| - |S_l|$. Once we did this, we move on to the next machine. Eventually, once the $(m-1)^{th}$ loop is done, you are finished. You will have a set left that can be used for the last machine m.

We see that the Running Time of the initialization, the recursion and the output totals to :$TR(m-1 \times B) + TR(m-1 \times n-1 \times B) + TR(1)$. Hence, we can say that our running time $TR(n) \in \mathcal{O}(mnB)$, with $B = \frac{1}{3}\sum_{j \in S_n} p_j max$ i.e., the highest value that of the possible processing times of job j.

Then the input size is $\mathcal{O}(n + n\log(max_{\forall j} p_j))$. Which is the size of the array and the time it takes to 'program' each value of the array. The algorithm run in pseudo-polynomial time, as the running time of the algorithm is also dependent the values of the array, the $max_{\forall j} p_j$.