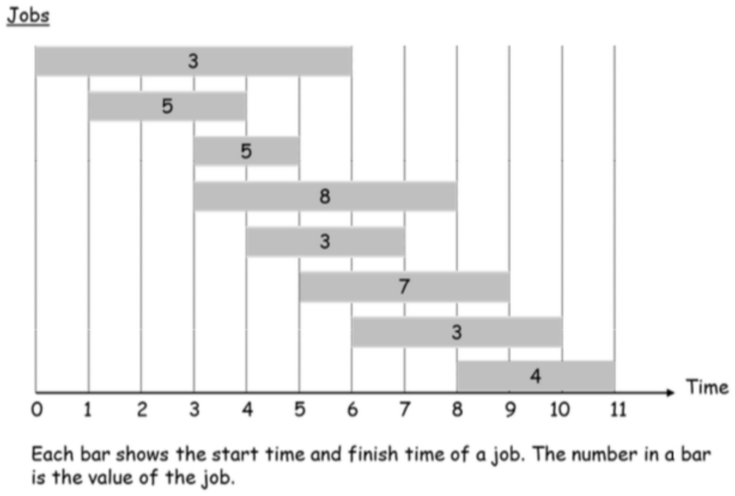
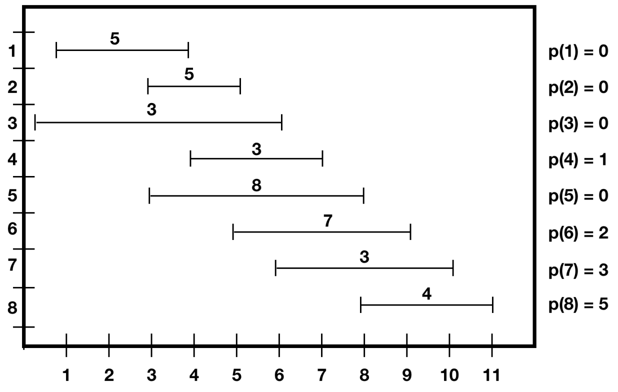
Homework 7

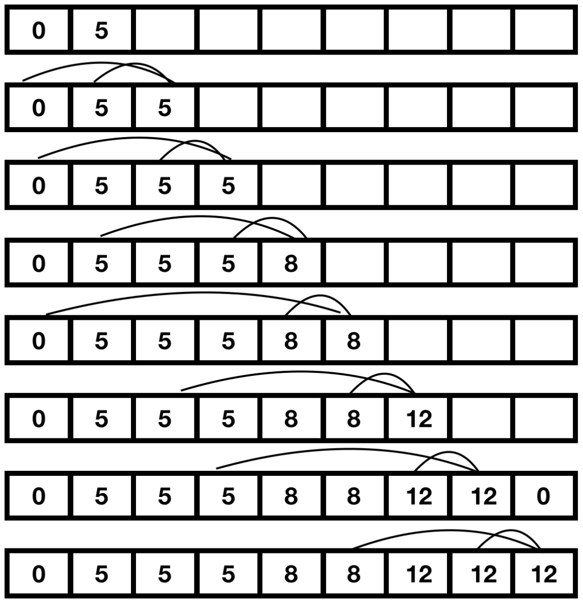
[(25) Weighted Interval Scheduling: algorithm tracing]

Consider the dynamic programming question algorithm we discussed for the weighted interval scheduling problem. Show the trace of running a bottom-up (iterative) implementation of the algorithm on the problem instance shown below. Show the trace in the same manner as Figure 6.5 (page 260) of the textbook. Remember to number the jobs in the increasing order of their finish times.





(a) Sorted jobs by finish time



(b) Algorithm tracing

[(25) Rising Trend]

Textbook Exercise 17 in Chapter 6. The problem to solve is, in other words, to select from a sequence of n numbers a subset, including the first number such that the selected numbers make the longest monotonously increasing sequence. In the exercise b, build the optimal substructure (with an explanation), write the iterative dynamic programming algorithm and show the running time analysis. Hints: the algorithm in the exercise (a) is a greedy algorithm; the answer returned in the exercise (b) is the length of the longest rising trend starting from $P[1]$, so think "backward" beginning from $P[1]$.

a) The greedy algorithm fails when $P = 10, 60, 11, 12, 13, 14$

b) Iterative dynamic programming algorithm: The optimal substructure of the problem has 2 cases:

Algorithm 1 Iteratively compute the length of the longest rising trend

```
Require: $P$ is a list of prices of length $n$

  $L_1 = 1$, $L_2 = 1$
  $S_1 = P[1]$, $S_2 = P[2]$
  for $i = 2, \dots, n$ do
      if $P[i] > S_1[L_1]$ then
          Append $P[i]$ to $S_1$
          $L_1 = L_1 + 1$
      else
          if $P[i] > S_2[L_2]$ then
              Append $P[i]$ to $S_2$
              $L_2 = L_2 + 1$
              if $L_2 \geq L_1$ then
                  $S_1 = S_2$        ▷ Erase all of the elements of $S_1$ and copy all elements of $S_2$ into $S_1$
                  $L_1 = L_2$
  return $S_1$, $L_1$
```

when i has reached the position n (the base case) and when $P[i]$ maybe be considered as a candidate member of the optimal solution. This relationship can be expressed as

$$OPT(i) = \begin{cases} 1 & i = n \\ \max\{1 + OPT(i+1), OPT(i+1)\} & \text{otherwise} \end{cases}$$

where the two cases described in the "otherwise" statement are the "if" and "else if" statements in the algorithm respectively.

The runtime complexity of the algorithm is $O(n^2)$. This is due to the fact that the algorithm iterates over the n elements in P and has the possibility of erasing all elements of S_1 and copying all elements of S_2 into S_1 in sequence, an operation that takes linear time. Thus, this gives us a worst-case run time order of growth of $O(n^2)$.