Solutions to Practice Problems 5

Solution 1: The points are added in the order: $\langle 0, 8, 4, 2, 6, 1, 3, 5, 7\rangle$. It is noteworthy that, ignoring the initial 0, the sequence is decreasing in terms of the largest power of 2 that evenly divides each number.

Solution 2: To compute the values of prior$[1..n]$, we only need access to a sorting algorithm. Let T denote the set of $2n$ numbers of all start and finish times. We sort the elements of T in increasing order. (For simplicity, let us assume that there are duplicate values.) We assume that each entry of T consists not only of the time (s_i or f_i), but also the index i of the request, and an indication of whether this is a start of finish time.

We create a variable prevFinish which we initialize to 0. We traverse this sorted sequence in increasing order of time value. Whenever we encounter finish time f_i, we save this in the variable prevFinish. Whenever we see a start time s_i, we set prior$[i] \leftarrow$ prevFinish. This is given in the following code block.

Computing the prior array

```
get-priors(s[1..n], f[1..n]) {
    let T[1..2*n] be the union of s[1..n] and f[1..n]
    sort T in increasing order
    prevFinish = 0
    for each t in T
        j = index associated with t  // we store this when creating the T array
        if (t is a finish time)
            prevFinish = f[j]        // save the most recent finish time
        else
            prior[j] = prevFinish    // save the prior for request j
}
```

To see this is correct, consider any request $[s_j, f_j]$. If there is no finish time smaller than s_j, then prevFinish will have its initial value of 0, and so, prior$[j] \leftarrow 0$, which is correct. On the other hand, if f_i is the last finish time just prior to s_j, then prevFinish $= i$ (since the elements of T are sorted), which implies that prior$[j] \leftarrow i$, as desired.

The running time is dominated by the $O(n \log n)$ time that it takes to sort the $2n$ elements of T.

Solution 3: See Fig. 1. For example, when computing $W[5]$, we have a choice between $W[4] = 9$ (don't take this request) or $v[5] + W[\text{prior}[5]] = v[5] + W[2] = 7 + 3 = 10$ (take the request). The latter is larger so we take the request, setting $W[5] \leftarrow 10$ and accept$[5] = \mathsf{T}$.

The final solution has total value $W[7] = 12$. To compute the requests, we trace back from the last entry of the accept array.

- accept$[7] = \mathsf{F}$, we skip this request and continue with $7 - 1 = 6$.
- accept$[6] = \mathsf{T}$, we take this request, with value $v[6] = 3$, and continue with prior$[6] = 4$.
- accept$[4] = \mathsf{F}$, we skip this request and continue with $4 - 1 = 3$.

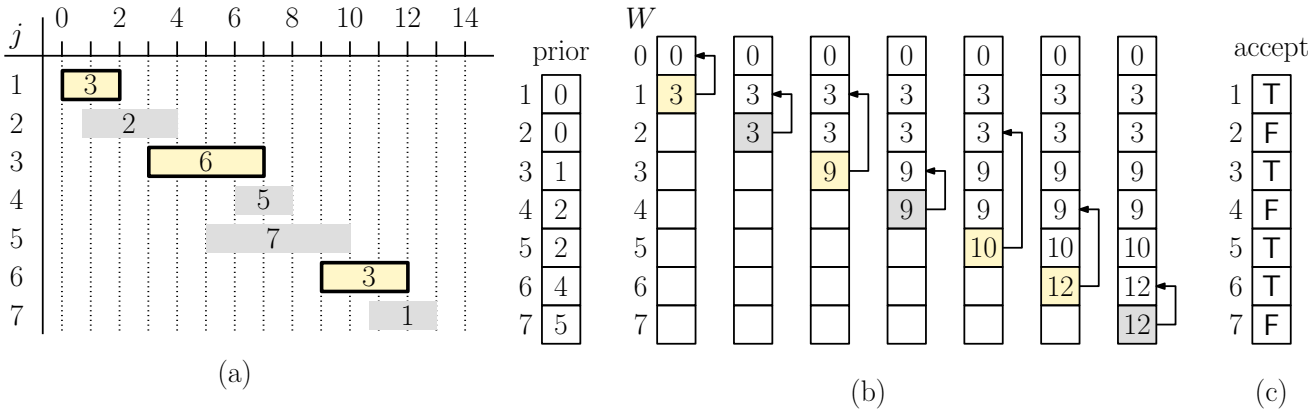

Figure 1: Weighted interval scheduling.

- accept[3] = T, we take this request, with value $v[3] = 6$, and continue with prior[3] = 1.
- accept[1] = T, we take this request, with value $v[1] = 3$, and continue with prior[1] = 0.

At this point the algorithm terminates, returning a total value of $3 + 6 + 3 = 12$ and the accepted requests $\{1, 3, 6\}$.

Solution 4: We will show that Gonzalez's algorithm as an approximation ratio of 4.5. We will show that for a point set P, where G denotes the output of Gonzalez's algorithm and O denotes the optimum k-center solution, $\Delta(G) \leq 4.5\Delta(O)$.

Recall that the proof involved three claims. The only element of the proof given in class where symmetry and triangle inequality are used is in Claim 3. The first two claims resulted in a corollary, which stated that every pair of greedy centers in G_{k+1} is separated by a distance of at least $\Delta(G)$. Using this, we continue with Claim 3.

Claim 3: Let $\Delta_{\min} = \Delta(G)/4.5$. Then for any set C of k cluster centers, $\Delta(C) \geq \Delta_{\min}$.

Proof: By definition of $\Delta(C)$, every point of P lies within distance $\Delta(C)$ of some point of C. Since $G_{k+1} \subseteq P$, this is true for G_{k+1} as well. Since $|G_{k+1}| = k + 1$, and C has only k clusters, by the pigeonhole principle, there exists at least two centers $g, g' \in G_{k+1}$ that are in the same neighborhood of some center $c \in C$.

This implies that both $\delta(g, c)$ and $\delta(g', c)$ are less than or equal to $\Delta(C)$. Since $g, g' \in G_{k+1}$, by the corollary, $\delta(g, g') \geq \Delta(G)$. Combining these observations with the properties of our (relaxed) metric space, we have

$$\begin{aligned}
\Delta(G) &\leq \delta(g, g') \leq 1.5(\delta(g, c) + \delta(c, g')) \quad \text{(by the relaxed triangle inequality)} \\
&\leq 1.5(\delta(g, c) + 2\delta(g', c)) \quad \text{(by relaxed distance symmetry)} \\
&\leq 1.5\Delta(C) + 3\Delta(C) = 4.5\Delta(C).
\end{aligned}$$

Rewriting this, we have $\Delta(C) \geq \Delta(G)/4.5 = \Delta_{\min}$, as desired.

Since Claim 3 applies to any set of k clusters, it applies to the optimum, O. We conclude that $\Delta(O) \geq \Delta_{\min}$. By definition, $\Delta_{\min} = \Delta(G)/4.5$, and so $\Delta(G) \leq 4.5\Delta(O)$. Therefore, Gonzalez's algorithm is a factor 4.5 approximation algorithm for the k-center problem.

Solution 5: We will prove the following theorem.

Theorem: Given any set system $\Sigma = (X, S)$, let G be the output of the greedy heuristic, and let O be any 2-cover of X. Then $|G| \leq 1 + (|O| \cdot \ln(2n))$, where $n = |X|$.

We will use the technical lemma from the lecture, which states that for all $c > 0$, $\left(1 - \frac{1}{c}\right) \leq e^{-1/o}$.

Let $o = |O|$ denote the size of the optimum set cover, and let $g = |G| - 1$ denote the size of the greedy set cover minus 1. Our approach will be to treat the set X to be covered as a *multi-set*, meaning that we allow multiple copies of each element. In our case, if $\{x_1, \ldots, x_n\}$ was the original set to be covered using a 2-cover, we will employ the greedy algorithm on a multi-set, where each element of X is replicated twice, that is $\widehat{X} = \{x_1, x_1, x_2, x_2, \ldots, x_n, x_n\}$. When we refer to the *size* $|\widehat{X}|$ of a multi-set, the multiplicities of the elements are counted, thus $|\widehat{X}| = 2n$.

Whenever a set is added to the 2-cover, it covers one of the two duplicated elements. The set is only 2-covered, when every element has been covered at least twice. Otherwise, the proof proceeds essentially as before, but with this minor adjustment.

For $i \geq 0$, let $\widehat{X}_i$ denote the multi-subset of $\widehat{X}$ that remains to be covered after the ith iteration of the algorithm, and let $n_i = |\widehat{X}_i|$. (For example, if an element has not been covered at all, it contributes twice to n_i, if it has been covered once, it contributes once to n_i, and if it has been covered twice or more, it does not contribute to n_i.) Initially, $\widehat{X}_0 = \widehat{X}$ and $n_0 = 2n$.

At the start of the ith iteration, there are n_{i-1} elements that remain to be covered. We know that there is a 2-cover of size o for the original set X, and therefore there is a 2-cover of size o for the multi-subset $\widehat{X}_{i-1}$. Since $n_{i-1} = |\widehat{X}_{i-1}|$, by the pigeonhole principal there exists some set that covers at least n_{i-1}/o elements. Since the greedy algorithm selects the set covering the largest number of remaining elements, it selects a set that covers at least this many elements. Therefore, the number of elements that remain to be covered is at most

$$n_i \;\leq\; n_{i-1} - \frac{n_{i-1}}{o} \;=\; n_{i-1}\left(1 - \frac{1}{o}\right).$$

Since this applies to every iteration, we see that with each iteration the number of remaining elements decreases by a factor of at least $(1 - 1/o)$. If we repeat this i times, we have

$$n_i \;\leq\; n_0\left(1 - \frac{1}{o}\right)^i \;=\; n\left(1 - \frac{1}{o}\right)^i.$$

Since the greedy heuristic ran for $g + 1$ iterations, we know that just prior to the last iteration we must have had at least one remaining uncovered element, and so we have

$$1 \;\leq\; n_g \;\leq\; 2n\left(1 - \frac{1}{o}\right)^g$$

By the technical lemma, with $c = o$, we have

$$1 \;\leq\; 2n\left(e^{-1/o}\right)^g \;=\; 2ne^{-g/o}$$

Now, if we multiply by $e^{g/o}$ on both sides and take natural logs we find that g satisfies:

$$e^{g/o} \;\leq\; 2n \qquad \Rightarrow \qquad \frac{g}{o} \;\leq\; \ln(2n) \qquad \Rightarrow \qquad g \;\leq\; o \cdot \ln(2n).$$

Since $g = |G| - 1$, $o = |O|$ and n was the size of our original set (before duplicating elements), we conclude that

$$|G| \ \leq \ 1 + (|O| \cdot \ln(2n)), \quad \text{where } n = |X|,$$

as desired.