*Design and Analysis of Algorithms* February 21, 2012
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Dana Moshkovitz and Bruce Tidor Handout 7

# Problem Set 1 Solutions

This problem set is due **at 9:00pm** on **Wednesday, February 15, 2012**.

---

**Problem 1-1. Variations of Median Select**

Recall from lecture the worst case linear time algorithm for selecting the ith smallest element from a set S of n distinct elements. In class we organized the elements into groups of 5, in this problem we wil analyze the behavior of the algorithm by picking different sizes for the groups.

**(a)** If we choose the group sizes to be 7, and picking the median element from each group of 7, repeat the analysis done in CLRS and write down the recurrence relation for $T(n)$. Solve the recurrence using the substitution method and prove its correctness using induction, in particular, state the assumptions you make about the base cases (for example, for groups of 5, we assumed $T(n) \leq cn$ for $n \leq 140$).

**Solution:**

We divide the $n$ elements of the input array into $\lfloor n/7 \rfloor$ groups of 7 elements each and at most one other group containing the remaining elements.

At least half of the medians for these groups are greater than or equal to the median-of-medians $x$. With the exception of the one group that contains fewer than 7 elements and the group containing $x$, each of the $\lceil n/7 \rceil$ groups has at least $4$ elements greater than $x$.

So, the number of elements greater than $x$ is at least

$$4\left(\left\lceil \frac{1}{2}\left\lceil\left\lceil \frac{n}{7}\right\rceil\right\rceil\right\rceil - 2\right) \geq \frac{2n}{7} - 8$$

This means that the number of elements less than $x$ is at most:

$$n - \left(\frac{2n}{7} - 8\right) = \frac{5n}{7} + 8$$

By symmetry, the number of elements exceeding $x$ is at most $\frac{5n}{7} + 8$. The cost of dividing the $n$ elements into groups, finding the median of each, and partitioning the input array is $O(n)$ The cost of recursively calling SELECT to compute the median of $\lceil n/7 \rceil$ medians is $T(\lceil \frac{n}{7} \rceil)$. The cost of running SELECT recursively on the elements below or above the pivot is $T(\frac{5n}{7} + 8)$. Hence, we obtain the following recurrence relation:

$$T(n) \leq \begin{cases} O(1) & : n < 126 \\ T(\lceil \frac{n}{7} \rceil) + T(\frac{5n}{7} + 8) + O(n) & : n \geq 126 \end{cases}$$

In the derivation below it will become clear that it is sufficient to have the cutoff be any integer strictly greater than 63, but it is mathematically convenient to make it $126$.

To solve the recurrence using the substitution method first assume that the running time is linear. Then we show that $T(n) \leq cn$ for sufficiently large $c$ and all $n \geq 126$:

$$\begin{aligned} T(n) &\leq c\left\lceil \frac{n}{7} \right\rceil + c\left(\frac{5n}{7} + 8\right) + an \\ &\leq \frac{cn}{7} + c + \frac{5nc}{7} + 8c + an \\ &= \frac{6cn}{7} + 9c + an \\ &= cn + \left(\frac{-cn}{7} + 9c + an\right) \end{aligned}$$

The latter is at most $cn$ if $\frac{-cn}{7} + 9c + an \leq 0$.
Since $n \geq 126$, this is equivalent to:

$$c \geq \frac{7an}{n - 63} \geq 14a. \tag{1}$$

**Base Case**

$$\begin{aligned} T(126) &\leq T\left(\left\lceil \frac{n}{7} \right\rceil\right) + T\left(\frac{5n}{7} + 8\right) + O(n) \\ &\leq O(1) + O(1) + an \\ &\leq 2b + an \\ &= 2b + 126a \leq cn = 126c \end{aligned}$$

So, choosing $c \geq a + \frac{b}{63}$ will satisfy the base case.
Hence, for $c \geq 14a + b$ and $n \geq 126$: $T(n) \leq cn$, satisfying both the base and inductive cases.

**(b)** What happens if we choose the group size to be 4? Since the median of 4 elements is somewhat ambiguous, assume that we always choose the *lower median* from every group, that is, the 2nd smallest element. Again, write down the recurrence relation for $T(n)$, solve it using the substitution method and prove its correctness using induction. Again stating your assumptions.

**Solution:**

With groups of size $4$, at least half of the medians are greater than or equal to the median-of-medians $x$. With the exception of the one group that contains fewer than $4$

elements and the group containing $x$, each of the $\lceil n/4 \rceil$ groups has at least 3 elements greater than $x$.
The number of elements greater than $x$ is at least

$$3\left(\left\lceil \frac{1}{2}\left\lceil\left\lceil \frac{n}{4}\right\rceil\right\rceil\right\rceil - 2\right) \geq \frac{3n}{8} - 6$$

This means that the number of elements less than $x$ is at most:

$$n - \left(\frac{3n}{8} - 6\right) = \frac{5n}{8} + 8$$

.
Each of the $\lceil n/4 \rceil$ groups except for the group containing $x$ and the residual group, has at least 2 elements less than $x$. So, the number of elements less than $x$ is at least

$$2\left(\left\lceil \frac{1}{2}\left\lceil\left\lceil \frac{n}{4}\right\rceil\right\rceil\right\rceil - 2\right) \geq \frac{n}{4} - 4$$

This means that the number of elements greater than $x$ is at most:

$$n - \left(\frac{n}{4} - 4\right) = \frac{3n}{4} + 4$$

.
In the worst case scenario, the algorithm will keep recursing on the bigger partition. So, the recurrence is as follows:

$$T(n) \leq \begin{cases} O(1) & : n < 512 \\ T(\lceil \frac{n}{4} \rceil) + T(\frac{3n}{4} + 4) + O(n) & : n \geq 512 \end{cases}$$

The solution to this recurrence is no longer linear. We will prove that it is $O(n \lg(n))$ by the substitution method. We show that for $n \geq 512$ and sufficiently large $c$, $T(n) \leq nc \lg(n)$:

$$\begin{aligned}
T(n) &\leq T(\left\lceil \frac{n}{4} \right\rceil) + T\left(\frac{3n}{4} + 4\right) + O(n) \\
&\leq c(\frac{n}{4} + 1)(\lg(\frac{n}{4} + 1)) + c(\frac{3n}{4} + 4)\lg(\frac{3n}{4} + 4) + an \\
&\leq c(\frac{n}{4} + 1)(\lg(\frac{n}{2})) + c(\frac{3n}{4} + 4)\lg(n) + an \\
&\leq c(\frac{n}{4} + 1)(\lg(n) - 1) + c(\frac{3n}{4} + 4)\lg(n) + an \\
&\leq c(\frac{n}{4}\lg(n) - \frac{n}{4} + \lg(n) - 1) + \frac{3cn}{4}\lg(n) + 4c\lg(n) + an \\
&\leq cn\lg(n) + (5c\lg(n) - \frac{cn}{4} - c + an) \\
&\leq cn\lg(n) + (c(5\lg(n) - \frac{n}{4} - 1) + an)
\end{aligned}$$

The latter term is at most $cn\lg(n)$ if $c(5\lg n - \frac{n}{4} - 1) + an \leq 0$. Making use of the fact that for $n \geq 512$, $n + 4 - 20\lg n > 0$, we can rewrite the inequality as follows:

$$\begin{aligned} 0 &\geq c(5\lg n - \frac{n}{4} - 1) + an \\ -an &\geq c(5\lg n - \frac{n}{4} - 1) \\ c &\geq \frac{4an}{n + 4 - 20\lg n} \end{aligned} \tag{2}$$

Also, note that for $n \geq 512$, $n + 8 - 40\lg n > 0$. Hence, now if we choose $c \geq 8a$, then

$$\begin{aligned} c &\geq 8a \\ c &\geq \frac{8an}{n} \\ &\geq \frac{8an}{n + (n + 8 - 40\lg n)} \\ &= \frac{8an}{2n + 8 - 40\lg n} \\ &= \frac{4an}{n + 4 - 20\lg n} \end{aligned}$$

**Base Case**

$$\begin{aligned} T(512) &\leq T\left(\left\lceil \frac{n}{4} \right\rceil\right) + T\left(\frac{3n}{4} + 4\right) + O(n) \\ &\leq O(1) + O(1) + an \\ &\leq 2b + an \\ &= 2b + 512a \leq cn = 512c \end{aligned}$$

So, choosing $c \geq a + \frac{b}{256}$ will satisfy the base case.
This proves that for $n \geq 512$ and $c \geq 8a + b$, $T(n) \leq cn\lg(n)$ satisfying both the base and inductive cases.

**Problem 1-2. Land for sale in one dimension**

**(a)** In 1-D land, everybody lives in a one dimensional region. You own a stretch of land in the shape of a circle, and would like to sell it to n customers. Each of these customers

is interested in a contiguous segment on the perimeter (ie. an arc), which can be represented in polar coordinates. For example, customer i wants to have $\theta_{i1}$ to $\theta_{i2}$ on the circle. Unfortunately, many of these arcs overlap, and you cannot sell a portion of a customer's request. Devise an algorithm to maximize the number of customers' requests you can fulfill.

**Solution:**

The challenge of the circular-arc scheduling problem lies in being able to "linearize" the input and exploit the unweighted greedy activity scheduling problem covered in lecture to obtain an optimal solution.

To "linearize" the input we need to determine an appropriate breakpoint at which to cut the circle without affecting the optimality of the solution to the modified problem. Choosing an arbitrary cutpoint on the circle may invalidate the scheduling of optimal segments crossing that boundary.

Now, suppose we are provided with extra information – we are given $\alpha_{k2}$, the endpoint of an activity in an optimal schedule $A^\star = (\alpha_{11}, \alpha_{12}) \dots (\alpha_{k1}, \alpha_{k2})$. Then we can conclude that any segments crossing $\alpha_{k2}$ are not part of an optimal solution, and we can safely choose $\alpha_{k2}$ as the initial condition in our linear scheduling algorithm, discarding any overlapping segments from the input. We don't actually need $\alpha_{k2}$ if we run the linear scheduling algorithm for every possible choice of segment endpoint. One of the instantiations of the linear scheduling problem is guaranteed to have $\alpha_{k2}$ as an intial condition, yielding an optimal solution.

**1. Algorithm Description** We provide an $O(n^2)$ algorithm for this problem, that uses the greedy unweighted activity scheduling algorithm presented in class as a subroutine.

Assume that the input consists of $n$ ordered pairs representing the start and end points of the arcs in polar coordinates: $\{(\theta_{i1}, \theta_{i2})\}$, for $i = 1 \dots n$. Our algorithm first sorts the list of ordered pairs by the second coordinate. Then going through each of the sorted list of segments, we run the greedy unweighted scheduling algorithm as a subroutine on segments from the current segment end time onwards.

The input to the linearized greedy scheduling algorithm is already sorted, and as part of its execution the subroutine discards segments that overlap the starting point from which the algorithm is executed.

In order to be able to recover an optimal scheduling, we keep track of the set of segments used for each of the executions of the linearized scheduling subroutine.

At termination we return the set with the maximum number of segments.

**Remarks** Note that an $O(n \lg n)$ solution has been identified in recent literature, but an $O(n^2)$ algorithm is sufficient to receive full credit.

**2. Worked Out Example** See Figure 1 below.[1]

**3. Correctness Proof** The circular scheduling problem can be reduced to the linearized scheduling problem given that the cut point on the circle corresponds to the end time for one of the segments in an optimal solution. Since we run the linearized algorithm for all possible cutpoint choices, one of these will have the appropriate initial condition. So, the correctness proof of the original algorithm is then reduced to the correctness proof for the greedy linear scheduling algorithm described in class.

**4. Running time analysis** The algorithm runs in $O(n^2)$ time. We only need to sort the set of ordered pairs by the endpoint of each segment once, taking $O(n \lg n)$ time. Then for each for the $n$ segment endpoints we discard segments overlapping with the start point and execute the greedy selection subroutine in linear time. This contributes $O(n^2)$ time. Finding the maximum among the $n$ solutions runs in linear time. So, the overall time complexity is $O(n^2)$.

**(b)** You are a firm believer in the free market and competition. So, in addition to each customer giving you an arc that they would like, he also offers a price. However, you still cannot sell a portion of any request. Devise an algorithm to maximise your profit.

**Solution:**

**1. Algorithm Description** Here the algorithm uses the weighted activity scheduling dynamic program presented in class as a subroutine in a manner analogous to the previous problem. The overall complexity for the algorithm is $O(n^2)$.

The algorithm first sorts all of the angles corresponding to the start and end points of the segments in ascending order. Then it iterates through each possible segment end point in ascending order, removes segments overlapping with that boundary, and executes the linear weighted scheduling dynamic program on the previously sorted input from the boundary onwards.

In addition to the optimal cumulative profit for each of the subroutine executions, segment choice backpointers are kept allowing for the recovery of an optimal solution

**2. Worked Out Example** See Figure 2 below.[2]

[1] Example courtesy Lauren Procz.
[2] Example courtesy Lauren Procz.

**3. Correctness** Similarly to 2b, the problem of finding the set of requests to fulfill to obtain maximum profit for circular arc segment inputs can be reduced to the problem of finding the 'linearized "schedule" with the maximum cumulative weight, using the dynamic programming algorithm presented in lecture. Again, this is due to the fact that one of the executions of the subroutine will have for an initial starting point, the endpoint that corresponds to the endpoint for one of the segments in an optimal solution. So, we simply refer to the correctness proof for the linearized weighted dynamic programming algorithm.

**4. Running time analysis** Sorting all of the endpoints takes $O(n \lg n)$ time. We only need to do this once. For each of the $n$ boundaries, we remove the overlapping segments in $O(n)$ time and run the dynamic program on the $2n$ angles in linear time. This requires $O(n^2)$ time. Finding the solution with the maximum profit among the various runs of the subroutine takes linear time. the overall time is $O(n^2)$.

**(c)** You have aquired a new area in the shape of a cross. Once again, there are n customers wanting to pay a price for a contiguous 1-D region on the cross (which could be a line segment, a "T" segment, or a "cross" segment). Like before, these requested regions overlap and can only be sold in its entirety. Devise an algorithm to maximize your profit.

**Solution:**

We will present a $\Theta(N \lg N)$ algorithm.[3]

**Description of Algorithm**

First, we look through the requests, and separate them into five categories:

- requests that contain the center of the cross
- requests that contain only the northern arm of the cross
- requests that contain only the eastern arm of the cross
- requests that contain only the southern arm of the cross
- requests that contain only the western arm of the cross

Then, we do dynamic programming on each of the arms to calculate the maximum profit we can make by selling tips of the arms. To do this, we sort all of the starting indices $s_i$ and ending indices $e_i$ of the requests on an arm, and iterate through them starting at the end of the arm. If we let $p[j]$ represent the maximum profit possible by selling the end of the arm up to index $j$, and if we let $c_i$ represent the price that customer $i$ is willing to pay, then we have the recurrence relation:

$$p[j] = \begin{cases} p[j-1] & \text{if } j = s_i \\ max\left(p[j-1], p[s_i - 1] + c_i\right) & \text{if } j = e_i \end{cases}$$

[3]Sample solution reproduced with permission from Kerry Xing.

Using this recurrence relation, we can compute the values of $p[j]$ for each arm, starting with a base case of $p[0] = 0$. We will store the values of $p[j]$ for each arm so that we can use them later.

Then, we iterate through the requests that contain the center of the cross. For each request, we consider accomodating it and selling the remaining arms for the best profit possible. Out of all these requests, we find the maximum profit possible, and report it as our answer. Note however, that we should also consider an empty request so that we will be able to consider selling only the arms of the cross.

**Illustrative Example**

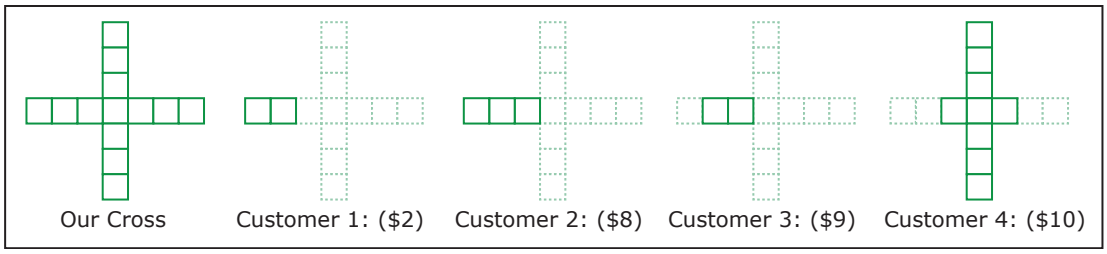



Image by MIT OpenCourseWare.

Consider the situation above, in which we have a cross that has arms of length 3. We have four customers, who are willing to pay different prices for different sections of land.

When we separate the requests into categories, we see that request 4 contains the center of the cross, and all the other requests contain only the western arm.

When we run dynamic programming on each of the arms of the cross, and we get the following results:

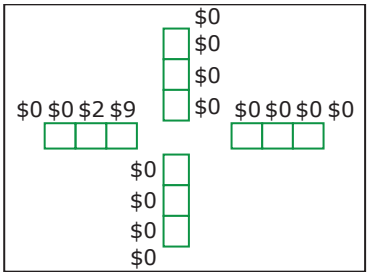



Image by MIT OpenCourseWare.

Now, when we look at customer 4's request, we see that the remaining land can be sold for $\$2$, so we can make $\$11$ if we satisfy customer 4's request. When we look at the empty request (in which we don't sell the center of the cross), we see that the 4 arms of the cross can be sold for $\$9$. Thus, we see that the optimal solution is to sell to customers 1 and 4 for $\$11$.

**Proof of Correctness**

**Lemma 1** *For each arm, the dynamic programming algorithm will correctly compute the values of $p[j]$ described above.*

PROOF. To do this, we will prove a loop invariant: $p[j]$ is the maximum profit possible by selling the tip of the arm up to index $j$.

During initialization, $p[0] = 0$, so the loop invariant holds.

For each iteration of the dynamic programming algorithm, we use the recurrence relation

$$p[j] = \begin{cases} p[j-1] & \text{if } j = s_i \\ max\left(p[j-1], p[s_i - 1] + c_i\right) & \text{if } j = e_i \end{cases}$$

If $j$ is a starting index, then we can't have enough land to accomodate the request yet (since we are only selling the land from the end of the arm to $j$ at this time). Thus, we must have $p[j] = p[j-1]$.

If $j$ is an ending index, then we have enough land to accomodate the request. If we fulfill the request, we will make $c_i$ profit, and we only have the land from the end of the arm to $s_i - 1$ remaining, since the land from $s_i$ to $e_i$ was sold. Thus, our maximum profit if we sell to customer $i$ is $p[s_i - 1] + c_i$. On the other hand, if we don't fulfill the request, our maximum profit would be $p[j-1]$, as in the previous case. Thus, the maximum profit $p[j]$ we can make is $max\left(p[j-1], p[s_i - 1] + c_i\right)$.

In either case, we see that the loop invariant is maintained, so the computed values of $p[j]$ are correct. □

Now, we will show that the algorithm will compute the optimal profit. We will consider two cases:

**Case 1:** The optimal solution satisfies a request that contains the center of the cross.

In this case, our algorithm will have considered the same request (that contained the center of the cross). Since the algorithm knows how to sell the remaining pieces of land in the optimal way, the algorithm will return an optimal solution.

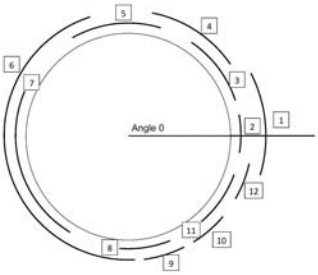
**Case 2:** The optimal solution does not satisfy a request that contains the center of the cross.

In this case, our algorithm will have considered the empty request and considered selling each arm of the cross in the optimal way. Thus, our algorithm will return an optimal solution.

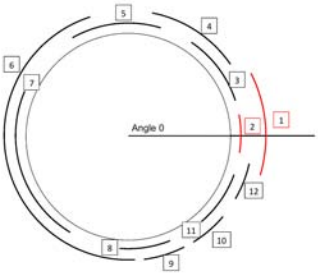
In all cases, we see that our algorithm will return an optimal solution, so we are done. □

**Running Time Analysis**

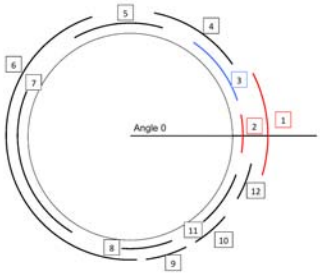
First, we see that separating the requests into five categories takes $\Theta(N)$ time. Next, we see that running dynamic programming on each arm takes $\Theta(N \lg N)$ time, because we have to sort the starting and ending indices for each request (which takes $\Theta(N \lg N)$ time) and applying the recurrence relation for each index (which takes $\Theta(N)$ time). Finally, we see that trying out all the requests that contain the center of the cross (and the empty request) takes $\Theta(N)$ time. Thus, we see that the total running time is $\Theta(N \lg N)$.



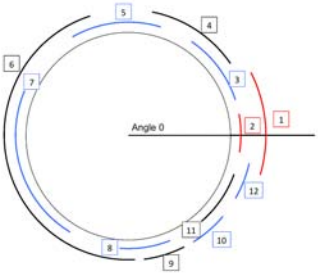

(a) Initial State



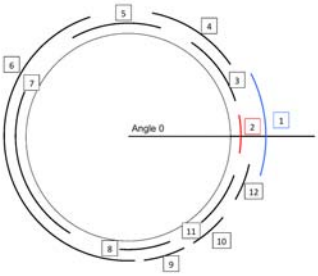

(b) Excluded Arc Set $R = \{1, 2\}$:



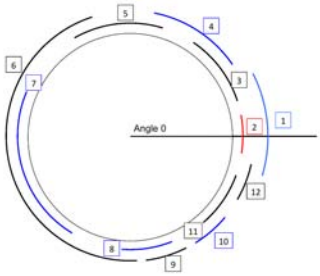

(c) Find solution $s$. Sort arcs by $\theta_{i2}$. Choose arc with minimum $\theta_{i2}$ to be in $s$. $s = \{3\}$.



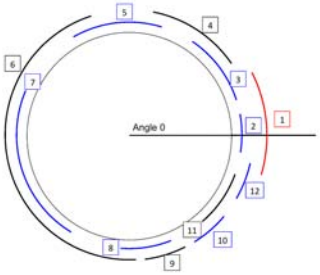

(d) Finish running the scheduling algorithm described in class. $s = \{3, 5, 7, 8, 10, 11\}, |s| = 6$.




(e) Now find a solution with alternative start points. Start with arc 1. $s_1 = \{1\}$:

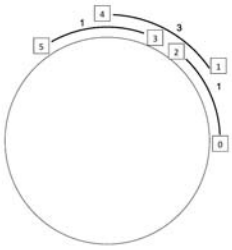



(f) $s_1 = \{1, 4, 7, 8, 10\}$, $|s_1| = 5$. $|s| > |s_1|$, so we need to keep iterating through arcs in $R$.

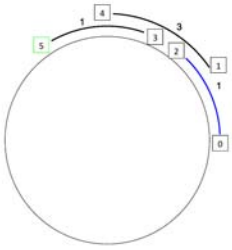



(g) Continue running the linear scheduling algorithm for different time starts. For arc 2. $s_2 = \{2, 3, 5, 7, 8, 10, 12\}$, $|s_2| = 7$. $|s| < |s_2|$, so return solution $s_2$.

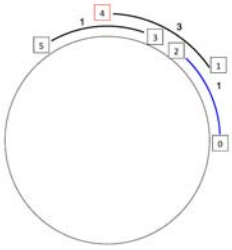
**Figure 1**: Worked out Example for Problem 2a



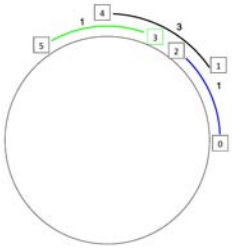
(a) Initial State. Start and end points are numbered (in squares) in sorted order based on an arbitrary "0" angle reference point. Numbers not in squares are arc costs.



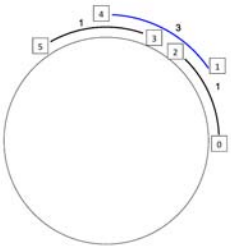
(b) Iterate through $n = 3$ arcs. Start with the highlighted arc. That arc is now fixed. We want to use dynamic programming to find the maximum weight solution on the circle segment from point 2 to point 0 (counterclockwise). $w(s_1) =$ maximum weight solution on segment $+w(1)$, where $w$ is a weight function.



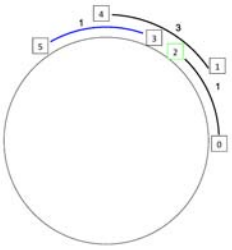
(c) Ignore this point because it is part of an arc that intersects the fixed arc $i$.



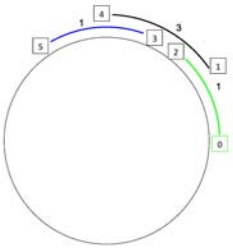
(d) Add the green edge to solution $s_1$. The algorithm has finished considering all points on the circle segment. $w(s_1) = 2$.



(e) Fix the next arc in solution $s_2$. All points on the circle segment are part of arcs that intersect the blue arc, so we ignore all of them. We're done with this solution: $w(s_2) = 3$.



(f) We end up with $w(s_3) = 2$.



(g) Return $s_2$ because it has the largest weight of all 3 solutions.

**Figure 2**: Worked out Example for Problem 2b

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2012