Math 448 HW 1

Zachary Kaplan

September 20$^{\text{th}}$, 2018

1.1

Given:

Index:	1	2	3	4	5	6	7	8	9	10	11	12	13
Arrival Times:	12	31	63	95	99	154	198	221	304	346	411	455	537
Service Times:	40	32	55	48	18	50	47	18	28	54	40	72	12

a)

We consider an algorithm that either waits for the next customer to enter the store, or serves the next customer who has not been served that has already arrived.
Since the input is sorted by arrival time, we know that the customer immediately following the last customer will be the next to be served. So in a loop, we handle the two possible cases for this customer:

1. **The customer has not arrived yet:** advance time until they arrive.
2. **The customer has already arrived:** advance time until they are done being served, and mark the resulting time as when they leave.

See a pseudo-code implementation below:

Algorithm 1: Find the departure time for each customer with a single server

Data: A, S, n where A, S are 1-indexed collection of size n, representing Arrival and Service times respectively (ordered by arrival times)
Result: a 1-indexed collection D of size n, representing Departure times

```
i ← 1; index of next customer
t ← 0; current time
while i ≤ n do
    if t < A[i] then
        t ← A[i]; if the next customer hasn't arrived yet, advance time
    end
    t ← t + S[i]; serve the customer and advance time
    D[i] ← t; note that the customer left
    i ← i + 1; advance to the next customer
end
```

Results:

Index:	1	2	3	4	5	6	7	8	9	10	11	12	13
Departure Times:	52	84	139	187	205	255	302	320	348	402	451	527	549

b)

We now need to consider a modification such that there are two servers in the store, and each customer can be served by either of them.
For this, we need to additionally maintain the remaining service time for the customers currently being served. We also need to consider advancing time only enough to finish serving one of the customers we are serving, or in other words advance time only the minimum of the two remaining service times. And edge case in the above, is that whenever either server is not working, we also need to consider when the next customer enters the store in our time advancement step. Our cases become:

1. **No servers are busy**: Advance time to next customer to arrive.
2. **One server is busy**: Advance time $min\{'\text{remaining service time}', '\text{time before next customer arrives}'\}$
3. **Both servers are busy**: Advance time $min\{'\text{remaining service times}'\}$

Note that whenever we advance time, we must also subtract that time from the remaining service times for customers being served. See pseudo-code below for details:

Algorithm 2: Find the departure time for each customer with two servers

Data: A, S, n where A, S are 1-indexed collection of size n, representing Arrival and Service times respectively (ordered by arrival times)

Result: a 1-indexed collection D of size n, representing Departure times

$i \leftarrow 1$; index of next customer
$t \leftarrow 0$; current time
$r_1^c \leftarrow -1$; index of customer server 1 is helping
$r_2^c \leftarrow -1$; index of customer server 2 is helping
$r_1^s \leftarrow 0$; remaining service time for server 1
$r_2^s \leftarrow 0$; remaining service time for server 2
function `AcceptNextCustomer`(j):
    if $t < A[i]$ **then**
        $t \leftarrow A[i]$; if the next customer hasn't arrived yet, advance time
    end
    $r_j^s \leftarrow S[i]$; assign service time of customer i to server j
    $r_j^c \leftarrow i$; assign index of customer i to server j
    $i \leftarrow i + 1$; move index to next customer
end
while $i \leq n$ **or** $r_1^c \neq -1$ **or** $r_2^c \neq -1$ **do**
    if $r_1^c = -1$ **and** $r_2^c = -1$ **then**
        `AcceptNextCustomer(1)`; give customer to server 1
    else if $r_1^c = -1$ **then**
        if $i \leq n$ **and** $A[i] \leq t$ **then**
            `AcceptNextCustomer(1)`; give customer to server 1
        else if $i \leq n$ **and** $A[i] - t < r_2^s$ **then**
            $r_2^s \leftarrow r_2^s + (A[i] - t)$; subtract wait time from the service time of server 2
            `AcceptNextCustomer(1)`; give customer to server 1
        else
            $t \leftarrow t + r_2^s$; advance time the remaining service time for server 2
            $D[r_2^c] \leftarrow t$; Mark customer being served by server 2 as done
            $r_2^c \leftarrow -1$; mark server 2 as open
        end
    else if $r_2^c = -1$ **then**
        if $i \leq n$ **and** $A[i] \leq t$ **then**
            `AcceptNextCustomer(2)`; give customer to server 2
        else if $i \leq n$ **and** $A[i] - t < r_1^s$ **then**
            $r_1^s \leftarrow r_1^s + (A[i] - t)$; subtract wait time from the service time of server 1
            `AcceptNextCustomer(2)`; give customer to server 2
        else
            $t \leftarrow t + r_1^s$; advance time the remaining service time for server 1
            $D[r_1^c] \leftarrow t$; Mark customer being served by server 1 as done
            $r_1^c \leftarrow -1$; mark server 1 as open
        end
    else
        if $r_1^s < r_2^s$ **then**
            $t \leftarrow t + r_1^s$; advance time the remaining service time for server 1
            $r_2^s \leftarrow r_2^s - r_1^s$; deduct this time from server 2's time
            $D[r_1^c] \leftarrow t$; Mark customer being served by server 1 as done
            $r_1^c \leftarrow -1$; mark server 1 as open
        else
            $t \leftarrow t + r_2^s$; advance time the remaining service time for server 2
            $r_1^s \leftarrow r_1^s - r_2^s$; deduct this time from server 1's time
            $D[r_2^c] \leftarrow t$; Mark customer being served by server 2 as done
            $r_2^c \leftarrow -1$; mark server 2 as open
        end
    end
end

Results:

Index:	1	2	3	4	5	6	7	8	9	10	11	12	13
Departure Times:	52	63	118	143	136	204	245	239	332	400	451	527	549

c)

We again use the algorithm presented in part a) as a base, but prioritize the customers as if they were in a stack as opposed to a queue.
In order to implement this, we will need to know which customers have been served at any given point. To do this, we will initialize D to a known invalid value to determine whether a customer has departed or not. Using this info, whenever we finish serving a customer, we can do a search for the last customer to arrive before the current time point using A and our book-keeping in D. Other than this, everything works like it did in a). See the pseudo-code below:

Algorithm 3: Find the departure time for each customer with a single server, using stack priority

Data: A, S, n where A, S are 1-indexed collection of size n, representing Arrival and Service times respectively (ordered by arrival times)
Result: a 1-indexed collection D of size n, representing Departure times
$t \leftarrow 0$; current time
for $i \in \mathbb{N} \mid i \leq n$ **do**
    $D[i] \leftarrow -1$; an initial invalid value for elt's of D
end
do
    $c^n \leftarrow -1$; will store the index of the next customer to arrive
    $i \leftarrow n$; start at the end of A when searching for the next customer
    while $i > 0$ **do**
        if $A[i] > t$ **then**
            $c^n \leftarrow i$; customer hasn't arrived yet, mark as possible next customer to arrive
        else if $D[i] = -1$ **then**
            break; found the most recent customer who hasn't been served and has arrived
        end
        $i \leftarrow i - 1$; decrement i
    end
    if $i = 0$ **then**
        if $c^n = -1$ **then**
            break; the next customer does not exist, we are finished.
        end
        $t \leftarrow A[c^n]$; if the next customer hasn't arrived yet, advance time
        $i \leftarrow c^n$; Set the current customer i to the next customer c^n
    end
    $t \leftarrow t + S[i]$; serve the customer and advance time
    $D[i] \leftarrow t$; note that the customer left
end

Results:

Index:	1	2	3	4	5	6	7	8	9	10	11	12	13
Departure Times:	52	84	139	320	157	207	254	272	348	402	451	527	549

2.7

Given RVs X, Y with the joint-PDF $f(x, y) = 2e^{-(x+2y)}\ \forall x, y \in \mathbb{R} \mid x, y > 0$, find $P\{X < Y\}$.

$$f(x,y) = 2e^{-(x+2y)} = e^{-x} \cdot 2e^{-2y} = f_X(x) \cdot f_Y(y), \text{where } f_X(x) = e^{-x}, f_Y(y) = 2e^{-2y}.$$
$$\implies X \text{and } Y \text{are } \underline{\text{independent}}.$$

Let $F_X(x), F_Y(y) s.t. dF_X(x) = f_X(x), dF_Y(y) = f_Y(y)$.

$$\begin{aligned}
\therefore P\{X < Y\} &= E_Y[P\{X < y | Y = y\}] = E_Y[P\{X < Y\}] = E_Y[\int_0^Y dF_X(x)] \\
&= E_Y[\int_0^Y f_X(x)dx] = E_Y[\int_0^Y e^{-x}dx] = E_Y[-e^{-x}|_0^Y] = E_Y[-e^{-Y}+1] = 1 - E_Y[e^{-Y}] \\
&= 1 - \int_Y e^{-y} dF_Y(y) = 1 - \int_0^\infty e^{-y} \cdot f_Y(y)dy = 1 - \int_0^\infty e^{-y} \cdot 2e^{-2y}dy \\
&= 1 - \int_0^\infty 2e^{-3y}dy = 1 - \frac{2}{3} e^{-3y}|_0^\infty = 1 - \frac{2}{3}(0-1) = 1 - \frac{2}{3} = \frac{1}{3}.
\end{aligned}$$

2.13

Show $Var\left[aX + b\right] = a^2 Var\left[X\right]$.

$$\begin{aligned}
Var\left[aX+b\right] &= E\left[((aX+b) - E\left[aX+b\right])^2\right] \\
&= E\left[(aX + b - aE\left[X\right] - b)^2\right] \\
&= E\left[(a(X - E\left[X\right]))^2\right] \\
&= a^2 E\left[(X - E\left[X\right])^2\right] \\
&= a^2 Var\left[X\right].
\end{aligned}$$

2.22

Find $P\{X > n\}$ where X is a geometric random variable with parameter p.

$$\begin{aligned}
&P\{X = n\} = p(1-p)^{n-1} \; \forall n \geq 1. \\
\implies &P\{X \leq n\} = \sum_{i=1}^{n} P\{X = i\} = \sum_{i=1}^{n} p(1-p)^{i-1} \\
&\qquad = p \cdot \frac{1-(1-p)^n}{1-(1-p)} = p \cdot \frac{1-(1-p)^n}{p} \\
&\qquad = 1 - (1-p)^n. \\
\therefore &P\{X > n\} = 1 - P\{X \leq n\} = 1 - (1 - (1-p)^n) = (1-p)^n.
\end{aligned}$$

2.28

Given X is a exponential random variable with parameter λ,

a)

Show that $E[X] = 1/\lambda$.

$$
\begin{aligned}
E[X] &= \int_0^\infty x \cdot \lambda e^{-\lambda x} dx \\
&= x \cdot -e^{-\lambda x}\Big|_0^\infty - \int_0^\infty \left(-e^{-\lambda x}\right) dx \\
&= (0-0) - \frac{e^{-\lambda x}}{\lambda}\bigg|_0^\infty \\
&= -\left(0 - \frac{1}{\lambda}\right) \\
&= \frac{1}{\lambda}.
\end{aligned}
$$

b)

Show that $Var[X] = 1/\lambda^2$.

$$
\begin{aligned}
Var[X] &= E\left[\left(X - \frac{1}{\lambda}\right)^2\right] \\
&= E\left[X^2 - \frac{2X}{\lambda} + \frac{1}{\lambda^2}\right] \\
&= E\left[X^2\right] - \frac{2}{\lambda} \cdot \frac{1}{\lambda} + \frac{1}{\lambda^2} \\
&= \int_0^\infty x^2 \cdot \lambda e^{-\lambda x} dx - \frac{1}{\lambda^2} \\
&= x^2 \cdot -e^{-\lambda x}\Big|_0^\infty - \int_0^\infty 2x \cdot \left(-e^{-\lambda x}\right) dx - \frac{1}{\lambda^2} \\
&= (0-0) + \frac{2}{\lambda}\int_0^\infty x \cdot \lambda e^{-\lambda x} dx - \frac{1}{\lambda^2} \\
&= \frac{2}{\lambda} E[X] - \frac{1}{\lambda^2} \\
&= \frac{2}{\lambda^2} - \frac{1}{\lambda^2} \\
&= \frac{1}{\lambda^2}.
\end{aligned}
$$

2.36

An urn contains four white and six black balls. A random sample of size 4 is chosen. Let X denote the number of white balls in the sample. An additional ball is now selected from the remaining six balls in the urn. Let Y equal 1 if this ball is white and 0 if it is black. Find

a)

$$
E[Y|X=2] = 1 \cdot P\{Y=1|X=2\} + 0 \cdot P\{Y=0|X=2\} = \frac{2}{6} = \frac{1}{3}.
$$

b)

$$\begin{aligned}
E[X|Y=1] &= \sum_{i=0}^{4} i \cdot P\{X=i\,|Y=1\} \\
&= \sum_{i=0}^{4} i \frac{P\{X=i \cap Y=1\}}{P\{Y=1\}} \\
&= \sum_{i=0}^{4} i \frac{P\{Y=1|X=i\}P\{X=i\}}{P\{Y=1\}} \\
&= \frac{1}{P\{Y=1\}} \sum_{i=0}^{4} i \left[\frac{4-i}{6}\right] \left[\frac{\binom{4}{i}\binom{6}{4-i}}{\binom{10}{4}}\right]. \\
P\{Y=1\} &= \sum_{i=0}^{4} P\{Y=1 \cap X=i\} \\
&= \sum_{i=0}^{4} P\{Y=1|X=i\}P\{X=i\} \\
&= \sum_{i=0}^{4} \frac{4-i}{6} \left[\frac{\binom{4}{i}\binom{6}{4-i}}{\binom{10}{4}}\right]. \\
\therefore E[X|Y=1] &= \frac{\sum_{i=0}^{4} i \left[\frac{4-i}{6}\right] \left[\frac{\binom{4}{i}\binom{6}{4-i}}{\binom{10}{4}}\right]}{\sum_{i=0}^{4} \frac{4-i}{6} \left[\frac{\binom{4}{i}\binom{6}{4-i}}{\binom{10}{4}}\right]} \\
&= \frac{\sum_{i=0}^{4} i(4-i) \left[\binom{4}{i}\binom{6}{4-i}\right]}{\sum_{i=0}^{4} (4-i) \left[\binom{4}{i}\binom{6}{4-i}\right]} = \frac{4}{3}.
\end{aligned}$$

c)

$$\begin{aligned}
Var[Y|X=0] &= E[(Y-E[Y|X=0])^2|X=0] \\
&= E[Y^2 - 2YE[Y|X=0] + E[Y|X=0]^2|X=0] \\
&= E[Y^2|X=0] - 2E[Y|X=0]E[Y|X=0] + E[Y|X=0]^2 \\
&= E[Y^2|X=0] - E[Y|X=0]^2 \\
&= E[Y|X=0] - E[Y|X=0]^2. \ \left[\textit{Note that } Y^2 = Y\right] \\
E[Y|X=0] &= 1 \cdot P\{Y=1|X=0\} + 0 \cdot P\{Y=0|X=0\} \\
&= \frac{4}{6} = \frac{2}{3}. \\
\therefore Var[Y|X=0] &= \frac{2}{3} - \left(\frac{2}{3}\right)^2 \\
&= \frac{2}{3} - \frac{4}{9} = \frac{2}{9}.
\end{aligned}$$

d)

$$
\begin{aligned}
Var[X|Y=1] &= E[X^2|Y=1] - E[X|Y=1]^2 \\
&= \frac{\sum_{i=0}^{4} i^2(4-i)\left[\binom{4}{i}\binom{6}{4-i}\right]}{\sum_{i=0}^{4}(4-i)\left[\binom{4}{i}\binom{6}{4-i}\right]} - \left[\frac{\sum_{i=0}^{4} i(4-i)\left[\binom{4}{i}\binom{6}{4-i}\right]}{\sum_{i=0}^{4}(4-i)\left[\binom{4}{i}\binom{6}{4-i}\right]}\right]^2 \\
&= \frac{5}{9}.
\end{aligned}
$$

Appendix

The following is the matlab code used for exercise 1.1 and it's associated output:

Zachary Kaplan

Math 448 9/20/18 HW 1

Contents

Common Data

```
% Global declarations for ease of reference.
% Allows helper functions to modify global state without a lot of parameter
% passing.
global A S n D i t rc rs;

A = [12 31 63 95 99 154 198 221 304 346 411 455 537];  % Arrival Times.
S = [40 32 55 48 18 50 47 18 28 54 40 72 12];          % Service Times.
n = length(A);    % Number of Customers.
D = zeros(1, n);  % Pre-allocate departure times.
```

Algorithm 1: Find the departure time for each customer with a single server.

```
t = 0;  % Current time.
for i = 1:n
    if t < A(i)
        % If the next customer has not arrived, advance time.
        t = A(i);
    end

    % Serve the next customer and advance time.
    t = t + S(i);
    % Mark departure time.
    D(i) = t;
end

% Print out the departure times.
disp("Algorithm 1: D = ")
disp(D)
```

```
Algorithm 1: D =
    52    84   139   187   205   255   302   320   348   402   451   527   549
```

Algorithm 2: Find the departure time for each customer with two servers

```
i = 1;  % Next customer.
t = 0;  % Current time.
rc = [-1 -1];  % Indices of customers being helped.
rs = [0 0];    % Remaining service time of customers heing helped.
% Helper function for sending the next customer to server j:
% See definition of AcceptNextCustomer(j) in section "Helper Methods".

% While there are more customers or any server is busy, continue.
while i <= n || any(rc ~= -1)
    if all(rc == -1)  % All servers are free.
        % Take the next customer, advancing time as needed.
        AcceptNextCustomer(1);
    else
        % Check if one of the servers are free.
        j = find(rc == -1, 1);
        if isempty(j)  % No servers are free.
            % Find the lesser of the service times remaining, and which
            % server min_idx has that time.
            [min_rs, min_idx] = min(rs);
            other_idx = mod(min_idx, 2) + 1;  % Index of other server.

            t = t + min_rs;  % Advance the remaining service time.
            % Deduct this time from the other server's time.
            rs(other_idx) = rs(other_idx) - min_rs;
            % Mark customer that we just finished serving as done.
            D(rc(min_idx)) = t;
            rc(min_idx) = -1;  % Mark server as open.
        else  % The j'th server is free.
            other_idx = mod(j, 2) + 1;  % Index of other (not free) server.

            if i <= n && A(i) < t
                % If the next customer is alredy here,
                AcceptNextCustomer(j)  % Accept them.
            elseif i <= n && A(i) - t < rs(other_idx)
                % If the next customer will arrive before the other server
                % is finished, deduct the wait time.
                rs(other_idx) = rs(other_idx) - (A(i) - t);
                % And accept the next customer.
                AcceptNextCustomer(j)
            else
                % If the other server will finish before the next customer,
                % then finish serving.
                t = t + rs(other_idx);  % Advance time.
                D(rc(other_idx)) = t;   % Mark customer as done.
                rc(other_idx) = -1;     % Mark server as open.
            end
        end
    end
end

% Print out the departure times.
disp("Algorithm 2: D = ")
```

```
disp(D)
```

```
Algorithm 2: D =
    52    63   118   143   136   204   245   239   332   400   451   527   549
```

Algorithm 3: Find the departure time for each customer with a single server, using stack priority.

```
t = 0;                % Current time
D(1:length(D)) = -1;  % Initialize D to all -1's.
while true
    % Find the most recently arrived customer who has not been helped.
    i = find(A <= t & D == -1, 1, 'last');
    if isempty(i)
        % No such customer exists, instead serve next customer to arrive.
        i = find(A > t, 1);
        if isempty(i); break; end  % No next customer, we are done.

        t = A(i);  % Advance time until they arrive.
    end

    t = t + S(i);  % Serve the current customer.
    D(i) = t;      % Mark customer as departed.
end

% Print out the departure times.
disp("Algorithm 3: D = ")
disp(D)
```

```
Algorithm 3: D =
    52    84   139   320   157   207   254   272   348   402   451   527   549
```

Helper Methods: Matlab requires local helper functions be defined after the rest of the script.

```
function AcceptNextCustomer(j)
    % We are using the global i, t, rc, rs, S, and A in this method.
    global i t rc rs S A;
    if t < A(i)
        t = A(i);  % Advance time if needed.
    end

    rs(j) = S(i);  % Mark down the service time of next customer.
    rc(j) = i;     % Mark down the index of next customer.
    i = i + 1;     % Increment next customer index.
end
```