1(35%). Consider the following directed graph shown below.

a). Show the adjacency matrix for this graph.

**Key:**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>inf</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>0</td>
<td>inf</td>
<td>inf</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>inf</td>
<td>0</td>
<td>inf</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
<td>inf</td>
<td>inf</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>inf</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

b). Using Prim’s algorithm, create a shortest spanning tree with “a” as the root.

**Key:**
Iteration 1: \( V_T = \{a\}, d[] = \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>inf</td>
</tr>
</tbody>
</table>

Select b to add to tree.

Iteration 2: \( V_T = \{a, b\}, d[] = \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Select e to add

Iteration 3: \( V_T = \{a, b, e\}, d[] = \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Select c to add

Iteration 4: \( V_T = \{a, b, e, c\}, d[] = \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Select d to add

Iteration 5: \( V_T = \{a, b, e, c, d\}, d[] = \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- a
- b
- c
- d
- e

\[ \begin{align*}
\text{Iteration 1: } V_T & = \{a\}, d[] = \\
\text{Iteration 2: } V_T & = \{a, b\}, d[] = \\
\text{Iteration 3: } V_T & = \{a, b, e\}, d[] = \\
\text{Iteration 4: } V_T & = \{a, b, e, c\}, d[] = \\
\text{Iteration 5: } V_T & = \{a, b, e, c, d\}, d[] =
\end{align*} \]
2. (30%). We have seen how a bitonic sort works. Apply the method to the following sequence of numbers: 12, 2, 11, 4, 9, 1, 10, 15, 5, 7, 14, 3, 8, 13, 6, 16

Note: you must first convert the above sequence into a bitonic sequence and then sort the bitonic sequence. Show all steps clearly.

Key:
Need to construct Bitonic sequence first.

\{12, 2, 11, 4, 9, 1, 10, 15, 5, 7, 14, 3, 8, 13, 6, 16\} =
\{2, 12, 11, 4, 1, 9, 10, 15, 5, 7, 14, 3, 8, 13, 16, 6\} =
\{2, 4, 11, 12, 15, 10, 9, 1, 3, 5, 7, 14, 16, 13, 8, 6\} =
\{1, 2, 4, 9, 10, 11, 12, 15, 16, 14, 13, 8, 7, 6, 5, 3\}

Now to create a sorted list
\{1, 2, 4, 9, 10, 11, 12, 15, 16, 14, 13, 8, 7, 6, 5, 3\} =

<table>
<thead>
<tr>
<th>Initial</th>
<th>Rnd 1</th>
<th>Rnd 2</th>
<th>Rnd 3</th>
<th>Rnd 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>10</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>11</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>13</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

3 (35%). We discussed bucket sort in class. Consider that you have p processors and n values to sort. Assume that you can use p buckets to sort the numbers.
a). Describe how you will use the p processors to sort
b). Outline how you will implement your algorithm using MPI.
c). Compute the parallel execution time.

Key: a). We have two alternatives

Alternative 1:
Step1: Each processor receives n/p elements. Places its inputs into p buckets.

Step 2: Each processor sorts one bucket

Pseudo code: Scatter n/p elements
Each processor locally places n/p elements into p buckets
Each processor gather one bucket
We can use all to all gather
Sort a bucket

Alternative 2: All processors receive n elements
Each processor selects inputs belonging to one bucket
Sort each bucket

Pseudo code: Broadcast n elements to all processors
Each processor selects numbers in one bucket
Sort

Key: b). MPI code

Alt 1:
....{standard MPI INIT and variable declaration etc
Sendbuffer contains n input values
Rcvbuffer contains data received
Send_count = receive_count =n/p}

MPI_Scatter (&sendbuffer, send_count, MPI_INT, &rcvbuffer,
receive_count, MPI_INT, 0, MPI_COMM_WORLD);

Sort_inputs_into_p_buckets;

for (i=0; i<p; i++){

/* since each processor may have different number of elements in buckets
/* processor pj need to collect how many each processor has and create two arrays
/* needed for GatherV
MPI_Gather(.....);  /* gather how many values each processor  
/* has in bucket i

}

Create_arrays_for_gather;  /* processor i creates its two arrays
for (i=0; i<p; i++){
    /* now processor i will gather bucket i
    MPI_Gatherv(.....);
}

Sort_bucket_i;

/* final gather at root

/* need to find out how many elements there are in each bucket
/* create two arrays at root
/* use Gather V

MPI_Gather(.....);  /* gather how many values there are in each bucket
Create_arrays_for_gather;
MPI_Gatherv(.....);

MPI Code for alt 2

MPI_Bcast(....);  /* all n values to all p processors

Select_values_for_bucket_i;
    /* processor i will select values for bucket i

Sort_bucket_i;

/* final gather at root

/* need to find out how many elements there are in each bucket
/* create two arrays at root
/* use Gather V

MPI_Gather(.....);  /* gather how many values there are in each bucket
Create_arrays_for_gather;

MPI_Gatherv(......);

Key c). Complexity analysis

Alt 1:

Step 1: Scatter n/p values  
\[ = t_s \log p + t_w \left(\frac{n}{p}\right)(p-1) \]

Step 2: Each processor places n/p elements in p buckets  
\[ = t_c \left(\frac{n}{p}\right) \]

Step 3: We repeat this step of collecting how many elements in each bucket  
\[ = p^*(t_s \log p + t_w (p-1)) \hspace{1cm} \text{-- only one value sent by each processor} \]

Step 4: We repeat this step also p times  
\[ \text{MPI}_\text{GatherV} \hspace{1cm} \text{--worst case each processor may have n/p in a bucket} \]
\[ = p(t_s \log p + t_w \left(\frac{n}{p}\right)(p-1)) \]

Step 5: Each processor sorts its bucket, each bucket may have all n values  
\[ = t_c (n \log n) \]

Step 6: Need to gather how many from each processor  
\[ = t_s \log p + t_w (p-1) \]

Step 7: GatherV  
\[ = (t_s \log p + t_w \left(\frac{n}{p}\right)(p-1)) \]

Total execution time:

Computation cost = \( t_c \left(\frac{n}{p}\right) + t_c (n \log n) \)

Communication cost:
\[ = t_s \log p + t_w \left(\frac{n}{p}\right)(p-1) + p^*(t_s \log p + t_w (p-1)) + p(t_s \log p + t_w \left(\frac{n}{p}\right)(p-1)) + t_s \log p + t_w (p-1) + t_s \log p + t_w \left(\frac{n}{p}\right)(p-1) \]

Complexity of Alt 2

Step 1: one-to-all-broadcast  
\[ = t_s + t_w (n) (log p) \]

Step 2: Each processor selects items in its bucket  
\[ = t_c (n) \]

Step 3: Each processor sorts its bucket and each bucket may have all n elements  
\[ = t_c (n \log n) \]

Step 4: Need to gather how many from each processor
Step 5: \[
    \text{GatherV} = t_s \log p + t_w (p-1)
\]

Total Execution time:

- Computation time = \( t_c(n) + t_c(n \log n) \)
- Communication time = \( t_s + t_w(n) (\log p) + t_s \log p + t_w(p-1) + t_s \log p + t_w(n/p) (p-1) \)

Note: Alternative 2 has much less communication cost.