I. Deadlocks (30 points)

(a) Is a cycle considered to be a necessary and/or a sufficient condition for deadlock? Explain. Is the answer dependent on any particular characteristic of the resource allocation graph being considered? Justify your answer; provide as well-reasoned a justification as you can based on the concepts discussed in class related to deadlock, the class textbook, and the class slides.

(b) Suppose a system has 5 processes, \( P = [P_1, P_2, P_3, P_4, P_5] \), and 4 resource types, \( R = [R_1, R_2, R_3, R_4] \). Further, assume one instance of each resource type. The current state of the system is defined by the following requests and assignments:

   (i) \( P_1 \) is requesting \( R_3 \) and has been assigned \( R_2 \);  
   (ii) \( P_2 \) is requesting \( R_1 \) and has been assigned \( R_3 \);  
   (iii) \( P_3 \) is requesting \( R_4 \) and has been assigned \( R_1 \);  
   (iv) \( P_4 \) is requesting \( R_3 \);  
   (v) \( P_5 \) is requesting \( R_2 \) and has been assigned \( R_4 \).

Is there deadlock in this system? (Explicitly state yes or no.) If applicable, draw a wait-for graph or resource-allocation graph, and use the appropriate graph to justify your answer. If you conclude there is no deadlock, your justification should contain a sequence of execution for the processes that shows that all processes can execute to completion.

(c) Suppose a system has 5 processes, \( P = [P_1, P_2, P_3, P_4, P_5] \), and 4 resources, \( R = [R_1, R_2, R_3, R_4] \), where the vector \([2, 1, 3, 3]\) gives the number of instances of each type of resources \([R_1, R_2, R_3, R_4]\), respectively. (There are 2 instances of resource \( R_1 \), 1 instance of resource \( R_2 \), and so on.) The current state of the system is defined by the following requests and assignments:

   (i) \( P_1 \) is requesting an instance of \( R_3 \) and has been assigned an instance of \( R_2 \);  
   (ii) \( P_2 \) is requesting an instance of \( R_1 \) and has been assigned an instance of \( R_3 \);  
   (iii) \( P_3 \) is requesting an instance of \( R_4 \) and has been assigned an instance of \( R_1 \) and an instance of \( R_3 \);  
   (iv) \( P_4 \) is requesting an instance of \( R_3 \);  
   (v) \( P_5 \) is requesting an instance of \( R_2 \) and has been assigned an instance of \( R_4 \).

Is there deadlock in this system? (Explicitly state yes or no.) If applicable, draw a wait-for graph or resource-allocation graph and use the appropriate graph to justify your answer. If you conclude there is no deadlock, your justification should contain a sequence of execution for the processes which shows that all processes can execute to completion.
II. **Swapping (20 points)**

Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB, and 15 KB. Which hole is taken for successive segment requests of:

(a) 12 KB;
(b) 10 KB; and
(c) 9 KB

for first fit? Repeat the question for best fit, worst fit, and next fit.

III. **Virtual Memory (30 points)**

A computer provides each process with 65,536 bytes of address space divided into pages of 4096 bytes. A particular program has a text size of 32,768 bytes, a data size of 16,386 bytes, and a stack size of 15,870 bytes. Will this program fit in the address space? If the page size were 512 bytes, would it fit? Explain your answers to get full credit. Remember that a page may not contain parts of two different segments.

IV. **Segmentation (20 points)**

Consider the segment table shown in Table 1. What are the physical addresses for the following logical address (segment #, offset)?

(a) (0, 430)
(b) (1, 10)
(c) (2, 500)
(d) (3, 400)
(e) (4, 112)

<table>
<thead>
<tr>
<th>Segment</th>
<th>Base</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>219</td>
<td>600</td>
</tr>
<tr>
<td>1</td>
<td>2300</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>1327</td>
<td>580</td>
</tr>
<tr>
<td>4</td>
<td>1952</td>
<td>96</td>
</tr>
</tbody>
</table>

Table 1: Segment table.