<table>
<thead>
<tr>
<th></th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>100</td>
<td>14</td>
</tr>
<tr>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>
Q1  (c) In a heap, there is no information which of the node's two children is larger. But in a binary search tree this is determined during insertion. In a heap, the root is always the largest (or smallest). In a binary search tree, the largest is always the leftmost (or rightmost).

(d) The complexity is $O(n)$. This is because there is no information which of the node's children the smaller. In the worst case, the search need to visit all nodes.

Q2.

```java
public void trickUP(int index){
    int parent = (index-1)/2;
    Node bottom = heapArray[index];

    while(index>0 && heapArray[parent].getKey()> bottom.getKey()){
        heapArray[index]=heapArray[parent];
        index=parent;
        parent =(parent-1)/2;
    }
    heapArray[index]=bottom;
}
```

```java
public void trickDown(int index){
    int smallerChild;
    Node top=heapArray[index];
    while(index<currentSize/2){
        int leftChild=2*index+1;
        int rightChild=leftChild+1;
        if(rightChild<current && heapArray[leftChild].getKey()>heapArray[rightChild].getKey())
            smallerChild=rightChild;
        else
            smallerChild=leftChild;
        if(top.getKey()<=heapArray[smallerChild].getKey())
            break;
        heapArray[index]=heapArray[smallerChild];
        index=smallerChild;
    }
    heapArray[index]=top;
}
```

Q3  (a)

|       | 4371 | 1323 | 6173 | 4344 | 4197 | 9677 | 1987 |

(b) ```java
public void inserQ(Node newN){
    int h=hast(newN.value);
    int p=0;
    while(hashArray_b[h].Occupied){
        p++;
        h+=p*p;
    }
    hashArray_b[h]=newN;
}
```