1) The following are multiple choice questions. Only one answer is correct. Circle the correct answer. Each problem is worth 2 pts.

For the best implementation of bubble sort discussed in class, an approximation of the number of comparisons and exchanges to sort an array of size $N$ is:

a) $N^2/4$ comparisons, $N^2/4$ exchanges
b) $N^2/2$ comparisons, $N^2/2$ exchanges
c) $N^2/2$ comparisons, $N^2/4$ exchanges
d) $N^2/4$ comparisons, $N^2/2$ exchanges
e) $N^2/2$ comparisons, $N$ exchanges

For the best implementation of selection sort discussed in class, an approximation of the number of comparisons and exchanges to sort an array of size $N$ is:

a) $N^2/4$ comparisons, $N^2/4$ exchanges
b) $N^2/2$ comparisons, $N^2/2$ exchanges
c) $N^2/2$ comparisons, $N^2/4$ exchanges
d) $N^2/4$ comparisons, $N^2/2$ exchanges
e) $N^2/2$ comparisons, $N$ exchanges

For the best implementation of insertion sort discussed in class, an approximation of the number of comparisons and exchanges to sort an array of size $N$ is:

a) $N^2/4$ comparisons, $N^2/4$ exchanges
b) $N^2/2$ comparisons, $N^2/2$ exchanges
c) $N^2/2$ comparisons, $N^2/4$ exchanges
d) $N^2/4$ comparisons, $N^2/2$ exchanges
e) $N^2/2$ comparisons, $N$ exchanges

For the implementation of quick sort discussed in class, the worst case complexity for sorting an array of size $N$ is:

a) $O(1)$

b) $O(n \log n)$
c) $O(n)$
d) $O(n \log n)$
e) $O(n^2)$
f) $O(2^n)$

For the implementation of quick sort discussed in class, the average case complexity for sorting an array of size $N$ is:

a) $O(1)$

b) $O(n \log n)$
c) $O(n)$
d) $O(n \log n)$
e) $O(n^2)$
f) $O(2^n)$

For the implementation of heap sort discussed in class, the worst case complexity for sorting an array of size $N$ is:

a) $O(1)$

b) $O(n \log n)$
c) $O(n)$
d) $O(n \log n)$
e) $O(n^2)$
f) $O(2^n)$

For the implementation of a heap discussed in class, the worst case complexity for inserting a single item into the heap of size $N$ is:

a) $O(1)$

b) $O(n \log n)$
c) $O(n)$
d) $O(n \log n)$
e) $O(n^2)$
f) $O(2^n)$
2) Assume the following pseudocode implementation of **Partition** and **QuickSort**:

```
Algorithm Partition(a, left, right)
    Input: array a of distinct elements, integers left and right
    Output: integer
    p <- a[right], lhs <- left, rhs <- right-1
    while lhs ≤ rhs do
        while lhs ≤ rhs and a[lhs] ≤ p do
            lhs = lhs + 1
        while rhs ≥ lhs and a[rhs] ≥ p do
            rhs = rhs - 1
        if lhs < rhs then
            swap(a[lhs],a[rhs])
        swap(a[lhs],a[right])
    return lhs

Algorithm QuickSort(a[], left, right)
    Input: array a of distinct elements, integers left and right
    Output: sorted array a
    if left ≥ right then return
    pivot <- Partition(a, left, right)
    QuickSort(a, left, pivot-1)
    QuickSort(a, pivot+1, right)
```

Show the effect of **QuickSort** on the following array `a`. Show the array after each call to **Partition** and indicate the final placement of **pivot**s.

```
a = {13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21}
```

- gray box indicates pivot placement
- following solution assumes recursion down lhs and rhs subtrees proceeds concurrently

<table>
<thead>
<tr>
<th>Ini</th>
<th>13</th>
<th>19</th>
<th>9</th>
<th>5</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>11</th>
<th>2</th>
<th>6</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>It1</td>
<td>13</td>
<td>19</td>
<td>9</td>
<td>5</td>
<td>12</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td>It2</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>12</td>
<td>8</td>
<td>7</td>
<td>19</td>
<td>11</td>
<td>13</td>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td>It3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>19</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>21</td>
</tr>
<tr>
<td>It4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>It5</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>It6</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>19</td>
<td>21</td>
</tr>
</tbody>
</table>

3) Assume that Quicksort uses the last item in the list as the pivot (as described earlier):

a) Give a list of 10 integers representing the worst-case scenario.
Any example with already sorted data, e.g., `a = {1,2,3,4,5,6,7,8,9,10}`

b) Give a list of 10 integers representing the best-case scenario.
Any example with pivot as median of each partition, e.g., `a = {2,1,5,4,3,7,8,10,9,6}`
Note this problem is difficult to show exact correct answer
4) Text, Problem C-10.10, Page 528. For simplicity, assume that each candidate receives a unique number of votes. The function description is as follows:

```plaintext
Algorithm findPrez(S,n)
Input: integer array S of size n, integer n
Output: integer winner

QuickSort(S, 0, n-1)
//assume we use Quicksort found later in homework. Note
QuickSort() is O(n^2) worst case. Therefore, might want to use
O(nlgn) best/avg/worst case sorting algorithm, like mergesort or
heapsort.

contender <- S[0]
votes <- 1
maxvotes <- 1

for i <- 1 to n-1 do //O(n)
    if S[i] = contender then
        votes <- votes + 1
    else
        if votes > maxvotes then
            maxvotes <- votes
            winner <- contender
            contender <- S[i]
            votes <- 1
```

Idea is as follows:
- sort S using O(nlgn) sort
- find S[i] with most occurrences in sorted S