1. This question deals with `SelectionSort`,

   Each of following vectors is in the middle of being sorted by SelectionSort. Right now, the loop condition for the outer loop (for loop indexed by i) is about to be checked. For each vector, what is the maximum number of times the loop could have executed so far? The answer might be different for each vector. After you have determined how many iterations have happened, perform the next few iterations and show what the vector will look like after each.

   (a)

15	27	30	36	48	87	98	52	55	57	86	85	78	53	58

   At most 5 iterations of SelectionSort

   (b)

12	31	33	36	64	79	51	69	70	75	81	78	92	65	67

   At most 4 iterations of SelectionSort

   (c)

11	12	16	18	22	26	25	63	59	75	32	58	64	57	29

   At most 5 iterations of SelectionSort

   (d)

17	18	24	28	29	30	41	77	43	53	88	71	87	82	72

   At most 7 iterations of SelectionSort

2. This question deals with `InsertionSort`, the relevant code of which is reproduced for your convenience:

   for $j \leftarrow 2$ to n **do**
   key $\leftarrow A[j]$
   $i \leftarrow j - 1$
   while $i > 0$ and $A[i] >$key **do**
   $A[i+1] \leftarrow A[i]$
   $i = i - 1$
   $A[i+1] \leftarrow$ key

   The following vectors in the middle of being sorted by InsertionSort. Right now, the next line of code to execute is for j to be incremented.

   For each, what is the maximum value that j can have right now (before the increment)? Recall that when we discuss pseudo-code, we treat arrays as indexed $1 \ldots n$, so if your belief is that the outer loop has executed exactly once, your answer should be two. After you have determined how many iterations have happened, perform the next few iterations and show what the vector will look like after each.

   (a)

20	34	35	67	77	80	94	54	70	26	47	40	56	81	92

   7

   (b)

37	71	93	45	38	92	42	52	22	82	47	90	54	21	19

   3

   (c)

37	56	61	64	74	81	63	69	82	68	80	60	76	18	48

   6

   (d)

15	30	43	45	54	65	70	97	79	83	34	13	74	67	88

   8

3. This question deals with `BubbleSort`, the relevant code of which is reproduced for your convenience:

```
for i ← 1 to n − 1 do
  for j ← 1 to n − i do
    if A[j + 1] < A[j] then
      Swap A[j] and A[j + 1]
```

The following array is in the middle of being sorted by BubbleSort. Right now, the next line of code to execute is for i to be incremented. What is the maximum value that i can have right now (before the increment)? Recall that when we discuss pseudo-code, we treat arrays as indexed $1 \ldots n$. After you have determined how many iterations have happened, perform the next few iterations and show what the vector will look like after each.

(a)

14	25	42	10	32	21	15	41	30	48	51	62	67	85	93

6

(b)

22	40	43	81	74	76	72	33	66	69	52	82	85	88	90

4

(c)

16	42	76	40	64	66	70	71	44	77	78	81	83	94	97

6

(d)

14	33	40	29	32	68	24	65	17	55	72	84	86	89	95

5

(e)

12	20	23	34	50	38	46	40	56	58	73	79	80	84	94

4. This question deals with `HeapSort`. Recall that this algorithm converts a vector into a max heap and then performs $n-1$ instances of the `extractMax` procedure.

   The following array is in the middle of being sorted by HeapSort. We are already past the portion of the code that converts the vector to a max heap. We may have performed some number of the extractMax procedure. How many times has extractMax been performed?

   (a)

75	49	48	31	33	25	24	15	22	13	17	23	83	89	93

   We have done 3 instances of extractMax

   (b)

74	69	43	22	64	37	32	12	14	17	79	80	88	91	96

   We have done 5 instances of extractMax

   (c)

38	30	26	11	19	46	47	55	73	78	82	89	91	92	97

   we have done 10 instances of extractMax

   (d)

30	28	26	18	19	21	31	38	41	43	58	70	71	79	88

   We have done 9 instances of extractMax

   (e)

24	23	13	15	27	29	35	42	53	58	69	70	72	85

   We have done 10 instances of extractMax

   (f)

13	11	23	33	34	42	53	59	60	67	71	75	91	93

   We have done 12 instances of extractMax

   (g)

39	36	27	25	17	12	41	42	44	45	49	66	71	81

   We have done 8 instances of extractMax

   (h)

60	49	36	42	46	10	11	12	65	66	72	86	88	93

   We have done 6 instances of extractMax

   (i)

62	60	56	59	42	55	35	28	29	39	12	10	72	76	80

   We have done 3 instances of extractMax

   (j)

56	48	50	44	46	24	41	21	22	23	58	71	92	97	98

   We have done 5 instances of extractMax

   (k)

43	34	29	16	14	44	48	54	58	66	71	84	85	87	97

   We have done 10 instances of extractMax

   (l)

45	42	27	11	17	16	48	57	77	78	79	90	91	92	93

   We have done 9 instances of extractMax

   (m)

70	67	47	50	65	36	27	38	49	19	34	28	87	94	98

   We have done 3 instances of extractMax

   (n)

77	56	71	52	48	17	19	43	31	29	82	87	90	96	98

   We have done 5 instances of extractMax

   (o)

43	37	32	18	25	60	64	66	67	69	72	81	82	83	93

   We have done 10 instances of extractMax

   (p)

27	26	24	16	13	19	50	55	60	70	84	88	92	95	96

   We have done 9 instances of extractMax

5. (2 points) This question deals with `QuickSort`, the relevant code of which is reproduced below for your convenience. Recall that the element at position q returned by the partition function is known as the *pivot*.

```
QuickSort(A, start, end)
  if start < end then
    q = partition(A, start, end)
    QuickSort(A, start, q − 1)
    QuickSort(A, q + 1, end)
```

We are sorting an array by QuickSort. A pivot was selected, the array was partitioned, and the pivot was placed in its correct position. Then, the algorithm was called recursively on the array in the range start to $q - 1$. That sub-array was partitioned, and the pivot placed in its proper position.

(a) What element must have been the first pivot selected?

(b) What element must have been the second pivot selected?

At the moment, the array looks like this:

(a)

36	11	21	32	39	48	45	55	52	56	44	53	59	91	95	84	92	83	93	85

The pivots were 59 and 39

(b)

11	39	21	36	32	44	84	55	95	53	93	91	56	48	52	45	92	83	59	85

The pivots were 44 and 11

(c)

21	11	36	44	39	32	48	45	52	55	84	91	56	53	93	59	92	83	85	95

The pivots were 95 and 52

(d)

14	33	10	34	49	44	58	62	38	66	91	92	75	73	68

The pivots were 66 and 34

(e)

44	27	19	42	36	51	80	60	62	76	69	84	93	89	92

The pivots were 84 and 51

(f)

33	20	34	16	30	35	54	44	69	43	72	78	88	94	76

The pivots were 72 and 35

(g)

16	11	13	18	34	33	39	37	29	46	60	82	74	54	70

The pivots were 46 and 18

(h)

17	15	22	51	28	70	53	23	44	55	71	89	97	85	74

The pivots were 71 and 22

(i)

40	11	31	15	42	80	73	46	79	50	78	83	98	95	97

The pivots were 83 and 42

Short Quiz

1. This question deals with `SelectionSort`, the relevant code of which is reproduced for your convenience:

   for $i \leftarrow 1$ to $n-1$ **do**
   min $\leftarrow i$
   for $j \leftarrow i+1$ to n **do**
   if $A[j] < A[\text{min}]$ **then**
   min $\leftarrow j$
   Swap $A[i]$ and $A[\text{min}]$

   Each of following vector is in the middle of being sorted by SelectionSort. Right now, the loop condition for the outer loop (for loop indexed by i) is about to be checked. What is the maximum number of times the loop could have executed so far?

22	24	25	26	48	31	38	29	42	72	71	75	28	59	95

   At most 4 iterations of SelectionSort

2. This question deals with `InsertionSort`, the relevant code of which is reproduced for your convenience:

   for $j \leftarrow 2$ to n **do**
   key $\leftarrow A[j]$
   $i \leftarrow j-1$
   while $i > 0$ and $A[i] >$key **do**
   $A[i+1] \leftarrow A[i]$
   $i = i-1$
   $A[i+1] \leftarrow$ key

   The following vector is in the middle of being sorted by InsertionSort. Right now, the next line of code to execute is for j to be incremented. What is the maximum value that j can have right now (before the increment)? Recall that when we discuss pseudo-code, we treat arrays as indexed $1 \ldots n$, so if your belief is that the outer loop has executed exactly once, your answer should be two.

18	68	70	44	40	61	94	59	93	22	31	69	87	82	55

   3

3. This question deals with `HeapSort`. Recall that this algorithm converts a vector into a max heap and then performs $n-1$ instances of the `extractMax` procedure.

   The following array is in the middle of being sorted by HeapSort. We are already past the portion of the code that converts the vector to a max heap. We may have performed some number of the extractMax procedure. How many times has extractMax been performed?

75	49	48	31	33	25	24	15	22	13	17	23	83	89	93

   We have done 3 instances of extractMax

4. This question deals with `BubbleSort`, the relevant code of which is reproduced for your convenience:

   for $i \leftarrow 1$ to $n-1$ **do**
     for $j \leftarrow 1$ to $n-i$ **do**
       if $A[j+1] < A[j]$ **then**
         Swap $A[j]$ and $A[j+1]$

   The following array is in the middle of being sorted by BubbleSort. Right now, the next line of code to execute is for i to be incremented. What is the maximum value that i can have right now (before the increment)? Recall that when we discuss pseudo-code, we treat arrays as indexed $1 \ldots n$.

10	18	22	27	34	15	52	42	64	74	73	79	89	94	95

   4

5. This question deals with `QuickSort`, the relevant code of which is reproduced below for your convenience. Recall that the element at position q returned by the partition function is known as the *pivot*.

   ```
   QuickSort(A, start, end)
     if start < end then
       q = partition(A, start, end)
       QuickSort(A, start, q - 1)
       QuickSort(A, q + 1, end)
   ```

   We are sorting an array by QuickSort. A pivot was selected, the array was partitioned, and the pivot was placed in its correct position. Then, the algorithm was called recursively on the array in the range start to $q-1$. That sub-array was partitioned, and the pivot placed in its proper position.

   (a) What element must have been the first pivot selected?

   (b) What element must have been the second pivot selected?

   At the moment, the array looks like this:

21	11	32	44	39	36	45	55	95	53	93	91	56	48	52	59	92	83	85	84

   The pivots were 45 and 32

6. Suppose you have a linked list where every key is an integer type (int, long, unsigned, or a similarly created type, etc). Each key is distinct. The list is NOT sorted right now, but you want it to be.

   (a) Which sorting algorithm that we saw in class would you use to sort the linked list? Explain why you think it is a good choice.

   (b) Which sorting algorithm that we saw in class would you NOT USE to sort the linked list? Explain why you think it would be a bad choice.

   There is not a single correct answer. On a test, selecting a correct answer and justifying its selection is the important part.