Selection sort, Insertion sort, Indirect sorting, Binary search

Remember to also practice: Homework, quizzes, class examples, slides, reading materials.

Insertion Sort

Ins1

- **a)** Insertion sort is $O(N^2)$. True or False.
- **b)** Insertion sort is stable. True or False.
- **c)** Binary search in a sorted array with N items has time complexity: ……(……..)

Ins2. Let A = [9, 5, 1, 3, 2, 7]. After 3 complete passes of **insertion sort** (3 iterations of the outer loop) the partially sorted array A is now: [1, 2, 3, 9, 5, 7] .

Is the above statement true or false? Justify your answer.

// Insertion sort pseudo-code provided for reference.

```
for (j = 1; j<= N-1; j++)  // indexes start from 0
    key = A[j]
    i = j-1
    while (i>=0) and (A[i]>key)
         A[i+1] = A[i]
        i = i-1
    A[i+1] = key
```

Ins3. (5 pts). Show the array, A, at the end of each iteration of **the outer loop** of **insertion sort**, for the **first 3 iterations**. Circle the changes in the array.

Index:	0	1	2	3	4	5
Original Array:	8	6	9	3	7	4

Ins4. a) Write code or pseudo-code for insertion sort.

b) Given the array below, show how it is being processed by insertion sort (as described by you above). In particular, on each row, fill in the values of the array at the end of the outer loop of insertion sort.

	4	**9**	**1**	**6**	**11**	**0**	**7**
After the **first** execution of the outer loop.							
After the **second** execution of the outer loop.							
After the **third** execution of the outer loop.							
…(and so on) …							

c) Show the array **after each iteration of the inner** loop.

Ins5. You have a **sorted** array, A, and an **unsorted** array, B. You need the data from A and B sorted together. Array A has N elements and array B has M elements. You do the following:

- Copy A in a third array, C.
- Insert (in the correct place) every element of B in C. You can assume C is large enough to hold both A and B.

a) (6 points) Give the Θ time complexity for this process. Justify your answer.

b) (6 points) Give an example that shows the best case (use N = 6 and M =4). Give the **values in A and B** (before the processing starts). Give the **Θ time complexity** for your example (which may be different than your answer in part a).

Ins6. You have to sort an array of bank transactions by date. Most of them are in order (by date), only a few are out of order.

Which sorting algorithm will you use between **insertion sort, selection sort ~~and merge sort~~** in order to take advantage of the fact that the array is almost sorted?

Justify your answer.

Assume that lgN elements are out of order and these lgN elements are placed in the worst possible way for your chosen algorithm. **Both describe the placement** and **give an example** of the placement for an array with N = 8 integers.

What is the time complexity (as a function of N) for your algorithm for the above worst case (of lgN out-of-order elements placed in the worst way in an array of N)?

Indirect sorting

Ind1 We discussed **indirect** sorting for an array. How do you do indirect sorting for a list?

1) Give a BRIEF explanation of indirect sorting,
2) Draw the list , L, 7 -> 3 -> 5 -> 2
3) Assume you use an extra **array**, A, for indirect sorting. Show
- what you initialize A to be
- the contents A **after** indirect sorting this list.

Make sure you show all memory addresses, all pieces of data and all types. Do NOT write any code for sorting. Do not draw data moving around.

Binary Search

BS1 (5 pts) Show the indexes: left, right, m when searching for value ***12*** in the sorted array below. If needed, assume rounding down.

Indexes	0	1	2	3	4	5	6	7
Array, A	1	3	4	7	10	14	16	20

left	right	m	A[m]

Selection sort

Here S stands for Selection sort problem

Sel1 Fill in the answer:

a) Selection sort does **Θ**(……..) **data moves**.

Sel2. Is selection sort stable? If yes, justify, if no, give a counter example.

Sel3. Show the array, A, at the end of each iteration of the outer loop of **selection sort**.

Index:	0	1	2	3	4	5	6
Original Array:	2	1	3	8	4	6	0

Sel4. Below is the code for selection sort. There was a suggestion in class for making it stable. Based on that suggestion or your own ideas, modify this algorithm to become stable. (The modified algorithm can behave somewhat different than selection sort, but it should still place the j-th smallest element its final position, j, after the j-th iteration of the outer loop.)

```
int i, j, temp;
for (j = 0; j < N-1; j++)  {
   int min_idx = j;
   for (i = j+1; i < N; i++){
      if (A[i] < A[min_idx])
         min_idx = i;
   }
   temp = A[min_idx];
   A[min_idx] = A[j];
   A[j] = temp;
}
```

b) Give an example of A and show what the original selection sort would do (and be unstable) and what your algorithm does (that it stable).