Selection sort, Insertion sort, Indirect sorting, Binary search - Solution

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: …**O**……(…… **lgN**……..)

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. No. In the first 3 iterations, Insertion sort will touch 5,1, and 3 and would not move 2. But here 2 is moved.

// 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
Orig Array:	8	6	9	3	7	4
	6	**8**	9	3	7	4
	6	8	**9**	3	7	4
	3	**6**	**8**	**9**	7	4

Ins4. a) Write code or pseudo-code for insertion sort. See above or in slides.

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.

The answer here should show what student code does. I will show the class insertion sort.

	4	**9**	**1**	**6**	**11**	**0**	**7**
After the **first** execution of the outer loop.	**4**	**9**	**1**	**6**	**11**	**0**	**7**
After the **second** execution of the outer loop.	**1**	**4**	**9**	**6**	**11**	**0**	**7**
After the **third** execution of the outer loop.	**1**	**4**	**6**	**9**	**11**	**0**	**7**
4^{th}	**1**	**4**	**6**	**9**	**11**	**0**	**7**
5^{th}	**0**	**1**	**4**	**6**	**9**	**11**	**7**
	0	**1**	**4**	**6**	**7**	**9**	**11**
Items moved in that iteration are highlighted in yellow.							

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

I will show how it got from the 4^{th} to the 5^{th}. This is what the code literally does.

Array after 4^{th} outer loop iteration, BEFORE the inner loop started.	**1**	**4**	**6**	**9**	**11**	**0**	**7**
After the **first** execution of the inner loop.	**1**	**4**	**6**	**9**	**11**	**11**	**7**
After the **second** execution of the inner loop.	**1**	**4**	**6**	**9**	**9**	**11**	**7**
After the **third** execution of the inner loop.	**1**	**4**	**6**	**6**	**9**	**11**	**7**
After the 4^{th} execution of the inner loop.	**1**	**4**	**4**	**6**	**9**	**11**	**7**
After the 5^{th} execution of the inner loop. This is the LAST iteration of the loop.	**1**	**1**	**4**	**6**	**9**	**11**	**7**
After instruction : A[i+1]=key	**0**	**1**	**4**	**6**	**9**	**11**	**7**
Data UPDATED in that iteration is highlighted in yellow.							

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. -> $\Theta(N)$.
- Insert (in the correct place) every element of B in C. You can assume C is large enough to hold both A and B. --> in worst case: $N+(N+1)+(N+2)+\ldots+(N+M-1)=$

$$N+\ldots+N+1+2+\ldots+(M-1) =$$

$$NM+M(M-1)/2$$

$$\Rightarrow \Theta(NM+M^2).$$

a) (6 points) Give the Θ time complexity for this process. Justify your answer. $\Theta(NM+M^2)$. See above.

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).

A= [1,2,3,4,5,6] B = [7,8,9,10]

In this case to insert the elements of B in C, just copies them at the end of C (they will be in the correct place) => inserting B in C will take Θ(M) =>

Total time to build array C is: Θ(N+M)

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? insertion sort

Justify your answer. Insertion sort is adaptive. It goes through all items, but since items are mainly sorted, it would do either no sliding at all or slide only a few items (so few that we assume they are a constant number). It should take close to Θ(N). Selection sort will not take advantage of the fact that the array is mainly sorted and run in $\Theta(N^2)$.

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.
N=8 => lg8 = 3 => 3 numbers out of order.
My alg: insertion sort. Worst for it these 3 must travel as much as possible back => they must the the smallest 3 values, placed in decreasing order at the end:
4,5,6,7,8,***3,2,1***

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)?

lgN must travel between (N-lgN) and (N-1) positions =>

total time complexity: (N-lgN)+(N-lg+1)+ (N-lg+2)+…+ [N-lg+(lgN-1)]

rewrite it as: (N-1) + (N-2) +…+(N-lgN) =

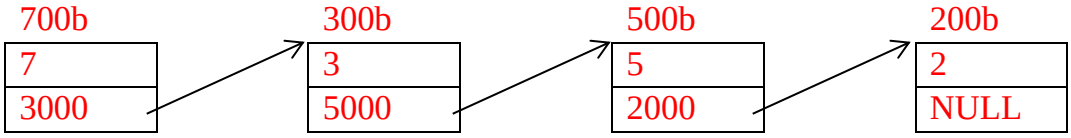
N+N+..+N –(1+2+3+…+lgN) = NlgN - (lgN)*[(lgN)-1]/2 = Θ(NlgN)

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: **builds an array to access into original data that gives the original data in sorted order. Here the array will keep the memory adr of nodes.**

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.

Initial A	**700b**	**300b**	**500b**	**200b**
A after indirect sorting	**200b**	**300b**	**500b**	**700b**

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]
0	**7**	**3**	**7**
4	**7**	**5**	**14**
4	**4**	**4**	**10**
5	**4**	**Cross over**	**stop**

Selection Sort

Sel1

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

Sel2. Is selection sort stable? If yes, justify, if no, give a counter example. **No. The array [4,*4*,2] after sorting will be [2,*4*,4]**

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
	0	1	3	8	4	6	2

	0	1	3	8	4	6	2
	0	1	2	8	4	6	3
	0	1	2	3	4	6	8
	0	1	2	3	4	6	8
	0	1	2	3	4	6	8
	0	1	2	3	4	6	8

Sel4. Below is the code for selection sort. 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.)

Modify this code, to not swap, A[j] with A[min_idx], but to push A[min_idx] to position j through consecutive swaps (same as insertion sort does in its inner 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).

	Original alg	Stable alg
Original array	3a,3b,1	3a,3b,1
After 1st iteration	**1,**3b,**3a**	**1,3a,3b**
After 2nd iteration	1,3b,3a	1,3a,3b