You're expected to work on the problems before coming to the lab. Discussion session is not meant to be a one-way lecture. The TA will lead the discussion and correct your solutions if needed. For many problems, we will not release 'official' solutions. If you're better prepared for discussion, you will learn more. The TA is allowed to give some bonus points to students who actively engage in discussion and report them to the instructor. The bonus points earned will be factored in the final grade.

1. (basic) Run Insertion Sort on input array $A = \langle 5, 8, 4, 2, 3, 1 \rangle$, show how the array looks before each iteration of the for loop in line 1; see page 18 for the pseudo-code.

   Sol.

   5 8 4 2 3 1
   4 5 8 2 3 1
   2 4 5 8 3 1
   2 3 4 5 8 1
   1 2 3 4 5 8

2. (basic) Show the operation of Merge sort on the array $A = \langle 7, 4, 2, 8, 3, 1, 5, 6, 9 \rangle$ as shown in Fig. 2.4.

3. (Intermediate) The following is a pseudo-code of Selection-Sort. Describe Selection-Sort in plain English.

```
Selection-Sort(A)
1.  n = A.length
2.  for j = 1 to n-1
3.   smallest = j
4.    for i = j+1 to n
5.      if A[i] < A[smallest]
6.        smallest = i
7.    exchange A[j] with A[smallest]
```

   Sol. In each iteration, by sequential scan we find the smallest one in the remaining elements, and append it to the sorted list.

   State the loop invariant and prove that the algorithm is correct. What is the running time?

   Sol. Loop invariant: at the start of each iteration of the outer for loop, the subarray $A[1 \cdots j-1]$ consists of the $j-1$ smallest elements in $A[1 \cdots n]$, and is sorted.

   Initialization, maintenance, termination? Omitted.

   RT: $O(n^2)$.

4. (basic) Suppose there are 2^n inputs to a certain problem. The running time of algorithm A is exactly 2^n for exactly one of the inputs, and 1 for any other input. Then, the running time is $O(1)$ since $2^n * \frac{1}{2^n} + 1 * (1 - \frac{1}{2^n}) \leq 2$. Is this statement correct?

   Sol. No. By default, we mean the worst case running time. And the worst case running time is defined as the maximum running time over all inputs (of size n). So, it is 2^n. Further, you need a distribution on the inputs to define average case running time, which is not given here.

5. (basic) Suppose the input sequence is already sorted. For this specific input, what is the running time of Insertion-sort and Merge-sort? Assume that they are implemented as in the textbook.

   Sol. $\Theta(n)$ and $\Theta(n \log n)$. You can say that we will revisit the RT of Merge sort in Chapter 4. and revisit asymptotic notations in chapter 3.

6. (basic) Answer the same question as above when the input is $\langle n, n-1, \dots, 1\rangle$.

   Sol. $\Theta(n^2)$ and $\Theta(n \log n)$. You can say that we will revisit the RT of Merge sort in Chapter 4. and revisit asymptotic notations in chapter 3. For Insertion sort, basically we need to count the number of comparisons made. In the previous case, one comparison is made in each iteration, so $\Theta(n)$. In this case, in j th iteration, roughty j comparisons are made. so, $\sum_{j=2}^{n} j = \Theta(n^2)$.

7. (basic) Briefly explain the computational model, RAM (Random Access Model).

8. (intermediate) Consider the pseudocode of $Merge(A, p, q, r)$ in the textbook. Given that $A[p...q]$ and $A[q+1...r]$ are both sorted, the function call merges the two sorted subarrays into a sorted subarray $A[p...r]$. Prove the correctness of Merge. For simplicity, you can assume that $L[1...n_1] = A[p...q]$ and $R[1...n_2] = A[q+1...r]$., and $L[n_1+1] = R[n_2+1] = \infty$. (So you only need to consider from lines 10). You can assume that all elements stored in the array have distinct values.

   Sol. Look Invariant: In the beginning of each for loop of iteration,

   (a) $A[p...k-1]$ is an sorted subarray of elements from $L[1...i-1]$ and $R[1...j-1]$, where $i \leq n_1 + 1$ and $j \leq n_2 + 1$; and

   (b) Any element in $L[i...n_1+1]$ and $R[j...n_2+1]$ is greater than any element in $A[p...k-1]$.

   Do you see why you should have (b)? Proof by initialization, maintenance, and termination is omitted.

9. (intermediate/advanced) Consider the searching problem: The input is an array $A[1 \cdots n]$ of n numbers and a value v. You're asked to find an index i such that $A[i] = v$. If there's no such index, return NIL. The following is a pseudocode of linear search, which scans through the sequence, looking for v. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties. What is the asymptotic worst case running time?

```
1. For i = 1 to n
2.    If A[i] == v then return i
3. return NIL.
```

   Sol. Loop Invariant: If the For loop hasn't ended, just before the start of each iteration of the loop, $A[1 \cdots i-1]$ doesn't contain v.

   Initialization: $i = 1$. Clearly, $A[1 \cdots 0]$ doesn't contain v.
   Maintenance: We know that $A[1 \cdots i-1]$ doesn't contain v. If $A[i] == v$, then it returns i and ends. So, the invariant is satisfied. Otherwise, we know that $A[1 \cdots i]$ doesn't contain

v, satisfying the invariant again in the next iteration.
Termination: The algorithm always terminates. There're two cases. The first case is when $i = n+1$. In this case, by the invariant, we know that $A[1 \cdots n]$ doesn't include v, and the algorithm returns NIL as desired. If $i \leq n$, it means that the For loop has ended by the return in line 2. And the algorithm returns i and we know that $A[i] = v$ from Line 2.

10. (advanced) You're given an array $A[1 \cdots n]$ of n numbers, and would like to know if the array includes two equal numbers. Describe an algorithm that does this job for you. The running time must be $O(n \log n)$.

    Sol. A naive algorithm would be to compare every pair of numbers, which will take a a quadratic time. If we sort the numbers (say using merge-sort) and compare every pair of adjacent numbers, it will take $O(n \log n)$ time. If we use hashing, we only need $O(n)$ time on average, but we will discuss this later when we learn hashing.

11. (advanced) We're given three sorted arrays $A[1 \cdots x]$, $B[1 \cdots y]$, $C[1 \cdots z]$. Describe an algorithm that merges the given three sorted (in increasing order) arrays into one (sorted array). What is the running time?

    Sol. Keep a pointer for each array, starting at 1. Let's use a, b, c for the pointers. Create another array D of size $a+b+c$. Also keep a pointer d for the array D. Set $d = 1$. Compare $A[a], B[b], C[c]$ and take the smallest number and copy it to $D[d]$... update pointers...

12. (basic) Merge sort is always faster than insertion sort. True or false?. Justify your answer.

    Sol. When the input is large yes. Otherwise, not necessarily. Particularly merge sort can be slower because it uses an auxiliary array for merging and the recursive calls overhead. Also, if the input is already almost sorted, insertion sort can be faster.

13. (intermediate) Observe that if the sequence A is sorted, we can check the midpoint of the sequence against v (the element we're looking for) and eliminate half of the sequence from further consideration. The binary search algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time. The following is a recursion-based pseudo-code is supposed the following: returns an index i such that $p \leq i \leq r$ and $A[i] = v$ if such an index exists, and returns -1 otherwise. Prove the correctness via induction.

```
BSearch(A, p, r, v)
   If r < p, return -1
   q = the floor of (p + r)/2
   if A[q] == v return q
   else if v < A[q] return BSearch(A, p, q-1, v)
   else return BSearch(A, q+1, r, v)
```

    Sol. Induction on n, the number of element in the subarray $A[p...r]$

    Base case. When the subarray $A[p...r]$ includes no elements, i.e. $r < p$. In this case, clearly the array includes no element of value v, and the algorithm return -1 as desired.

    Induction step. Suppose the algorithm is correct when $n = 0, 1, 2, ...k$ for some $k \geq 0$. Now we want show that it is correct also when $n = k+1$. If $A[q] == v$, then the algorithm is obviously correct as it returns q. Otherwise, v must appear in either $A[p...q-1]$ or $A[q+1,r]$ depending on whether $v < A[q]$ or not, if it does in the $A[p...q]$. In the former case, as $A[p...q-1]$ includes at most k elements, so by induction hypothesis, we know that $BSearch(A,p,q-1,v)$ correctly returns the index of A whose value is v if such an index exists. The latter case is similar.

14. (intermediate) Can you use binary search to improve the running time of insertion sort to $O(n \log n)$?

    Sol. No. You will have to move $\Theta(i)$ elements that are greater than $A[i]$ in the ith iteration.