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) You're given an array $A[1 \cdots 8] = \langle 3, 1, 6, 8, 4, 2, 5, 7 \rangle$. Is this a max-heap? If not, make it a max-heap by iteratively applying the Max-Heapify function. In other words, illustrate the operation of Build-Max-Heap on the array. What is $A[1 \cdots 8]$ at the end?

   Sol. Here we only show $A[1 \cdots 8]$ at the end: $\langle 8, 7, 6, 3, 4, 2, 5, 7 \rangle$.

2. (Basic) Illustrate the operation of Max-Heap-Insert(A, 9) on the heap you obtained in problem 1.

   Sol. Here we only show the array after the completion of the operation: $A[1 \cdots 9] = \langle 9, 8, 6, 7, 4, 2, 5, 1, 3 \rangle$.

3. (Basic) Illustrate the operation of Heap-Sort on the max-heap you've obtained in problem 2. Note that you don't need to execute line 1 of Heap-Sort, as you already have a max-heap A. **Sol.** Omitted

4. (Basic) You're given an array $A[1 \cdots 8] = \langle 3, 1, 6, 8, 4, 2, 5, 7 \rangle$. Is this a min-heap? If not, make it a min-heap by iteratively applying the Min-Heapify function. **Sol.** In other words, it is asking to a min-heap. As a result, we obtain $A[1 \cdots 8] = \langle 1, 3, 2, 7, 4, 6, 5, 8 \rangle$.

5. (Basic) Is an array that is sorted in increasing order a min-heap?

   Sol. Yes. Do you see why every node has a lower index than its child(ren)?

6. (Basic) Is Heap-Sort an in-place sorting algorithm? We say that a sorting algorithm is in-place if only $O(1)$ elements are stored outside the input array at any time – in other words, at most $O(1)$ extra memory is used in the whole process. Do you see why Insertion-Sort is in-place but Merge-Sort is not?

   Sol. Heap-sort is an in-place sorting algorithm

7. (Intermediate) Give a pseudo-code to test if a given binary heap is a max-heap or not.

   Sol. For each node, we examine if it has a value no smaller than its children, if there are any. We leave it as a exercise to change this into a pseudocode.

8. (Intermediate) We want to show that it takes $O(n)$ time to build a max-heap. Towards this end, we will take the following steps:

   (a) Suppose the binary heap has height H. Explain that $2^H \leq n < 2^{H+1}$.

   (b) Explain that there are at most 2^{H-h} nodes of height h in the heap, for all $0 \leq h \leq H$.

   (c) Show that there are $O(n/2^h)$ nodes of height h.

   (d) What is the running time of Max-Heapify on a node of height h?

   (e) From above, the running time for building a max-heap is at most $\sum_{h=1}^{H} O(h) \cdot O(n/2^h)$.

   (f) Show that $\sum_{h=1}^{\infty} h/2^h = O(1)$.

Sol. See lecture slides or textbook.

9. (Intermediate) You want to add a new operation to the max-priority-queue, namely Heap-Second-Maximum(A). The function returns the second largest value (key) stored in the heap A. Give a pseudo-code of this operation. What's the running time of your algorithm? The faster, the better.

   Sol. Hint: Return the second largest value among $A[1], A[2], A[3]$.

10. (Advanced) The operation Heap-Delete(A, i) deletes (the item in) node i from heap A. Give a pseudocode of Heap-Delete that runs in $O(\lg n)$ time for an n-element max-heap.

    Sol. Hint: You can implement it from scratch. But by recycling existing operations, you can have a simpler implementation: Heap-Increase-Key(A, i, ∞);. Heap-Extract-Max(A);

11. (Advanced) Give an $O(n \lg k)$-time algorithm to merge k sorted lists into an sorted list, where n is the total number of elements in all the input lists. Hint: Use a min-heap. Explain the running time.

    Sol. Maintain a pointer for each sorted list, which initially points to the first element in the list. Build a min-heap using the first elements in the lists. ($O(k)$ time). In each iteration, extract the minimum from the min-heap and append it to the result list. Consider the list from which the min element originates. Add to the min-heap the value in the list pointed to by the pointer, and update the pointer. (Each iteration takes $O(\log k)$ time. Here extract-min and insert each take $O(\log k)$ time.).