Advanced sorting algorithms

A – Theory part

A.1. Identify the Big-O time complexity for the **best case** and **worst case** of *heap sort*. Repeat the question for *quick sort*, *merge sort, radix sort,* and *counting sort*.

A.2. What is the worst case of *quick sort,* where the corresponding time complexity is $\mathbf{O(n^2)}$? Among sorting algorithms with time complexity (in average case) $O(n\log_2 n)$, *quick sort* is most popular though its complexity deteriorates to $\mathbf{O(n^2)}$ in the **worst case**. Explain why.

A.3. What is the **running time** of *heap sort* on an array of *n* elements **already sorted**? Explain.

A.4. What is the **running time** of *quick sort* on an array of *n* **identical elements**? Explain.

A.5. What are the differences among *radix sort, counting sort*, and *bucket sort*?

A.6. Consider an array of positive integers, A = {100, 77, 49, 1, 29, 51, 7, 15, 90}. Sort the given array in **ascending order** using *heap sort*. Present the resulting array after heap building and each heap correction. Mark the boundary between sorted and unsorted regions.

A.7. Consider an array of positive integers, A = {100, 77, 49, 1, 29, 51, 7, 15, 90}. Sort the given array in **ascending order** using *quick sort,* where the pivot is chosen following the *median-of-three* strategy. Present the resulting array after each partition with the pivot in its correct position. Mark the boundary between sorted and unsorted regions.

A.8. Consider an array of positive integers, A = {100, 77, 49, 1, 29, 51, 7, 15, 90}. Sort the given array in **ascending order** using *merge sort*. Draw a diagram depicting the split and merge.

A.9. Consider an array of positive integers, A = {100, 77, 49, 1, 29, 51, 7, 15, 90}. Sort the given array in **ascending order** using *counting sort*. Present the frequency of occurrence for every distinct element and the accumulated values.

A.10. For each of the following situations, choose the sorting algorithm that will perform the best. The candidates are: *selection sort, bubble sort, insertion sort, interchange sort, heap sort, merge sort*, and *quick sort*. Note that an algorithm may be chosen several times, while more than one answer may be acceptable for some questions.

a. I have a fast computer with many processors and lots of memory. I want to choose a sorting algorithm that is fast and can also be parallelized easily to use all of the processors to help sort the data.

b. You can only use the sorting algorithm with time complexity of $O(n\log_2 n)$, even in the worst case. You also may not use extra space, except for a few local variables.
c. The array is mostly sorted already (a few elements are in the wrong place).
d. Instead of sorting the entire data set, you only need the k smallest elements where k is an input to the algorithm but is likely to be much smaller than the size of the entire data set.

B – Coding part

B.1. Provide appropriate modifications to *heap sort* so that it can be used for **descending order**. Repeat the question for *merge sort*, *quick sort, radix sort* and *counting sort.*

B.2. Implement the three versions of *quicksort* shown in the lecture slide.

B.3. Implement *merge sort* without using the additional temporary array.

B.4. Implement an $O(n\log_2 n)$ sorting algorithm that is different from those in the lecture.