1 Comparison Sorts

a) How many inversions are there in $3, 2, 5, 7, 6, 4, 2$? 10

To count the number of inversions, for each element, count the number of elements that are larger than them and come before themselves.

The inversions: 3-2, 7-6, 5-4, 7-4, 6-4, 3-2, 5-2, 7-2, 6-2, 4-2

b) Run the **insertion sort** on $3, 2, 5, 7, 6, 4, 2$.
The first call of `swap` on this array is `swap(0, 1)` which means swap the elements in indices `0` and `1`. Fill in the next five `swap` operations to continue the insertion sort.

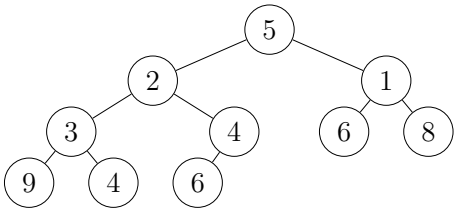
2nd Call: `swap(3, 4)`
3rd Call: `swap(4, 5)`
4th Call: `swap(3, 4)`
5th Call: `swap(2, 3)`
6th Call: `swap(5, 6)`

- How many more `swap` operations do we need to call to finish the insertion sort? 4

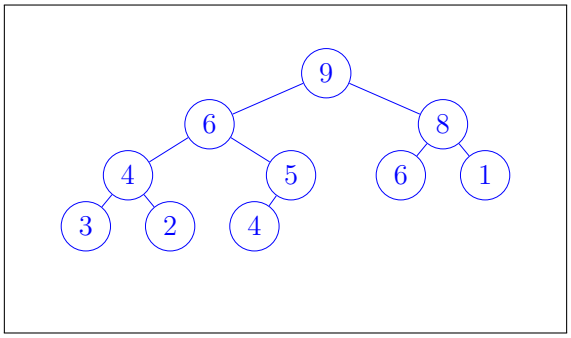
The intermediate results after each `swap` call are:

Initial: 3, 2, 5, 7, 6, 4, 2
1st Call: **2**, **3**, 5, 7, 6, 4, 2
2nd Call: 2, 3, 5, **6**, **7**, 4, 2
3rd Call: 2, 3, 5, 6, **4**, **7**, 2
4th Call: 2, 3, 5, **4**, **6**, 7, 2
5th Call: 2, 3, **4**, **5**, 6, 7, 2
6th Call: 2, 3, 5, 4, 6, **2**, **7**

c) Given an array and its representing tree below, perform **bottom-up max-heapification** to construct a max-heap. Show the heap and the array representation after the heapification.



Initial:	5	2	1	3	4	6	8	9	4	6
Result:	9	6	8	4	5	6	1	3	2	4



d) Give **two** reasons why bottom-up heapification is better than constructing the heap naively.

One possible answer:
1) bottom-up heapification runs in $O(N)$ while building heap naively runs in $O(NlogN)$.
2) bottom-up heapification does **not** require an additional array. (in-place)

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4

e) Using the idea of merging from **mergesort**, can these two arrays be merged? If so, give the merged result. If not, briefly explain why and fix the original arrays.

3	5	76	67

1	3	4	4

No, it **cannot** because the left array is not sorted. In order to perform merging, both arrays need to be sorted.

f) Given an array A of size $2N$ with N items in sorted order in indices 0 through $N-1$, and an array B of size N with N items in ascending order, describe an algorithm in English to merge the array B into A so that A contains all of the items in ascending order. Only $O(1)$ **extra memory** allowed. Also, justify the **runtime** of your algorithm.

One possible answer:
We can modify from the typical mergesort by merging from right to left instead of left to right. Hence, we would start filling the largest value at index $2N-1$ of array A and start filling the rest to the left in an descending order.
Since we consider each element at most once, the runtime is $O(N+N)$ which is simplified to $O(N)$.

g) Perform **Hoare partitioning** on 3,7,**5**,8,6,2,2 with 5 as a pivot. Fill in `swap` operations and intermediate results until we finish the partitioning. The first `swap` is provided for you to keep the pivot safe at the leftmost index.

1) `swap(0, 2)`	**5**	7	**3**	8	6	2	2
2) `swap(1, 6)`	5	**2**	3	8	6	2	**7**
3) `swap(3, 5)`	5	2	3	**2**	6	**8**	7
4) `swap(0, 3)`	**2**	2	3	**5**	6	8	7

h) Give one advantage and one disadvantage of Hoare partitioning, compared to naive partitioning.

Advantage: One possible answer

- It does **not** require an additional array. (in-place)

Disadvantage: One possible answer

- It does **not** ensure the stability. (unstable)
- Harder to implement

i) Assuming a weird partitioning runs in $\Theta(N^2)$ where N is the size of an array to be partitioned, give the recurrences representing the best-case runtime and the worst-case runtime of **quicksort** using this weird partitioning.

Best-case: $T(N) = 2T(\frac{N}{2}) + N^2$ Worst-case: $T(N) = T(N-1) + N^2$

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4

2 Radix Sorts

a) Show the output after **three** passes in least significant digit (**LSD**) radix sort for the input $[5423, 452_A, 3156, 299, 956, 452_B]$, i.e., after applying counting sorts three times starting from the rightmost digit. Note that, A and B just tell the order of which one comes first.

3156 299 5423 452_A 452_B 956

b) One TA wants to experiment with LSD radix sort and MSD radix sort by replacing counting sort with different sorting algorithms and run each radix sort as usual using the chosen sorting algorithm. Select sorting algorithm(s) to replace counting sort such that each radix sort will give correct result in any given input.

- LSD radix sort:
  - ☐ Selection Sort ■ Insertion Sort ☐ Heap Sort ■ Mergesort
  - ■ Naive Quicksort ☐ Quicksort with Hoare Partitioning

  Any **STABLE** sorting algorithms would work as a replacement of counting sort for LSD radix sort.
- MSD radix sort:
  - ■ Selection Sort ■ Insertion Sort ■ Heap Sort ■ Mergesort
  - ■ Naive Quicksort ■ Quicksort with Hoare Partitioning

  Since MSD radix sort break elements in group(s) at each step, there is no need for the sorting algorithm to be stable.

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4