Question 1: Wavelets, Sketches (1 P.)

a) Calculate the wavelet transform of the following cumulative distribution over the numbers 0 to 7. Also calculate the Mean-Squared-Error (MSE) if you omit all coefficients ≤ 3.

$$[2, 6, 8, 12, 20, 25, 26, 29]$$

b) Given the following Table with elements $x \in X$ and the hash values for the hash function $h : X \to [0, 1]$.

Read Jensen's Inequality
E[g(x)] >= g(E[x])

x	$h(x)$
Toyota	0.05
Ford	0.489
Bughatti	0.678
BMW	0.281
Audi	0.81

x	$h(x)$
Mercedes	0.15
Dodge	0.565
Porsche	0.579
Nissan	0.395
Hyundai	0.91

Using the KMV Sketch algorithm for $k \in 1, 2, 3, 4, 5$, calculate the estimate as well as the absolute error. Will the error get smaller with a larger k? How does this generally relate to the KMV Sketch?

Question 2: Statistics in PostgreSQL (1 P.)

To improve the query optimization, PostgreSQL keeps track of various statistics about every column in every table. Consider the following statistics about the *lineitem* table from the TPC-H database:

`SELECT * FROM pg_stats WHERE tablename = 'lineitem'`

attname	**null_frac**	**n_distinct**	**most_common_vals**	**most_common_freqs**
l_linenumber	0	7	{1,2,3,4,5,6,7}	{0.25,0.21,0.17,0.14,0.10,0.07,0.03}
l_orderkey	0	419664	{1703939,3644579,507137, 703172,712326,770882, 971014}	{0.000133333,0.000133333,0.0001, 0.0001,0.0001,0.0001, 0.0001}
l_extendedprice	0	-0.11964	{13747.3,25165.2, 26677.8,32274.0, 33265.3,35938.8,50370.3 }	{0.0001,0.0001,0.0001,0.0001, 0.0001,0.0001,0.0001 }

a) What does each column of *pg_stats* represent?

b) Which of the following queries benefit from these statistics? The total size of the table is 6 001 215.

- ☐ `SELECT * FROM lineitem`
- ☐ `SELECT * FROM lineitem WHERE l_orderkey IS NOT NULL`
- ☐ `SELECT * FROM lineitem WHERE l_orderkey = 406631`
- ☐ `SELECT * FROM lineitem WHERE l_extendedprice = 32274.0`
- ☐ `SELECT * FROM lineitem WHERE l_extendedprice <= 9500.0`

c) Use the statistics to estimate the result size of each query in part (b).

In the table below an additional column of the *pg_stats* table is shown. It shows "A list of values that divide the column's values into groups of approximately equal population. The values in *most_common_vals*, if present, are omitted from this histogram calculation." – `https://www.postgresql.org/docs/10/static/view-pg-stats.html`

attname	**histogram_bounds**
l_extendedprice	{906.00,1528.60,2062.06,3035.16,3639.44,4444.04,5265.52,5920.60, 6751.85,7518.63,8252.80,9032.32,9777.68,10540.68,11267.46, 11921.76,12652.00,13413.40,14078.28 (...) }

d) Decide for which queries in (b) this additional column is useful and estimate the result set size.

Question 3: Join ordering (1 P.)

Given the following query on the TPC-H database:

when no join relation given, then selectivity is 1,
because of cross product.
when join key is there, then sel. is even small.

```
SELECT *
FROM orders o, customer c,  nation n,
WHERE o.o_custkey = c.c_custkey AND c.c_nationkey = n.n_nationkey
```

a) Draw the query graph.

b) Calculate the join selectivities.

c) Calculate the cost C_{out} for all possible join trees without cross products.

Question 4: Join Ordering (1 P.)

a) **Left-deep trees vs. Bushy trees**
Given the following relations: R_1, R_2, R_3 and R_4, with $|R_1| = 100$, $|R_2| = 50$, $|R_3| = 200$, $|R_4| = 10$, and the following join selectivities: $j_{1,2} = 0.04$, $j_{2,3} = 0.10$, $j_{2,4} = 0.05$.

|R1JR2|.|R4|.fR4,R1.fR4,R2
= 200 . 10 . 1. 0.05
100

   i) Draw the query graph.

   ii) List all left-deep trees without cross products.

   iii) Calculate the C_{out} cost for the join orderings created in (ii) where R_3 is joined last (on the top of the tree).

cost cout: 100 + cout(R1jR2) + cout(R4)
= 100 + 200 + 0
= 300

b) Is there a bushy tree (that is not a linear tree), without cross products, that has lower cost?

Question 5: Join Ordering Algorithms (1 P.)

Write a program which compares the different join ordering algorithms, presented in the lecture. You may use any language you like. You have two write one method for each greedy algorithm plus two methods that find the best and worst possible join ordering. For the last two functions you have to find all possible orderings. Use the C_{out} cost to compare the algorithms.

You may use the Java template provided in OLAT. The template provides all required boilerplate code and instructions on how to use it.

Use your program to generate plans for the following relations: $|R_1| = 200, |R_2| = 300, |R_3| = 20, |R_4| = 140$ with the selectivities: $j_{1,3} = 1/100$, $j_{2,3} = 1/50$, $j_{3,4} = 1/5$ Cross products are not allowed.

Submit the created plans with their C_{out} costs and the code that created them.

HINT: The best and worst plans range somewhere between $(8\,000, 11\,000)$.

Greedy 1: Cost = |R|
Greedy 2:
Greedy 3: is greedy 2 and more