**Slide 1-4: Title Slide**

**Title:** Skyline Search Algorithms for Real Estate Analysis  
**Subtitle:** Comparing Sequential Scan, BBS, and Divide-and-Conquer Approaches

**Speaker Notes (30 seconds):** "Good morning everyone. Today I'll be presenting my analysis of skyline search algorithms applied to real estate data. I implemented three different algorithms to find optimal houses based on cost and size criteria. This is particularly relevant for multi-criteria decision making in real-world applications like property selection."

**Slide 5: What is the Skyline Problem?**

**Title:** Understanding the Skyline Problem

**Content:**

* **Real Estate Context:** Finding best houses to buy
* **Two Criteria:** Cost (lower is better) + Size (higher is better)
* **Skyline Points:** Houses not dominated by any other house
* **Domination:** House A dominates House B if A is cheaper AND larger

**Speaker Notes (90 seconds):** "Let me start by explaining what we mean by the skyline problem. Imagine you're house hunting and care about two things: you want the house to be as cheap as possible and as large as possible. Now, some houses are clearly better choices than others. For example, if House A costs $150k for 100 square meters, and House B costs $200k for 90 square meters, then House A dominates House B because it's both cheaper and larger. The skyline consists of all houses that aren't dominated by any other house - these represent your best options. In my dataset of 300,000 houses, I found 8 skyline points that represent the optimal trade-offs between cost and size."

**Slide 6: Algorithm 1 - Sequential Scan**

**Title:** Sequential Scan: The Brute Force Approach

**Content:**

* **Method:** Compare every house against every other house
* **Time Complexity:** O(n²) - very slow for large datasets
* **Process:** For each house, check if any other house dominates it
* **Result:** Simple but inefficient baseline algorithm

**Code Snippet:** Show key domination checking loop

**Speaker Notes (90 seconds):** "The first algorithm I implemented is Sequential Scan, which is the brute force approach. For every house in the dataset, I compare it against every other house to see if it's dominated. This means for 300,000 houses, I'm doing 90 billion comparisons! The algorithm is straightforward - for each house, I loop through all others and check if any house has both lower cost and higher size. If I find such a house, the current house is eliminated from the skyline. While this approach is simple to understand and implement, it's extremely slow with O(n²) time complexity. In my testing, this took over 0.9 seconds, which serves as our baseline for comparison."

**Slide 7,8: Algorithm 2 - Branch and Bound Skyline (BBS)**

**Title:** BBS with R-tree: The Smart Spatial Approach

**Content:**

* **Key Innovation:** R-tree spatial indexing for efficient pruning
* **MBR Concept:** Group houses into spatial regions with boundaries
* **Pruning Strategy:** Skip entire regions dominated by skyline points
* **Process:** Build tree → traverse with priority queue → prune aggressively

**Visual Suggestion:** Simple diagram showing tree structure with MBR boxes

**Speaker Notes (120 seconds):** "The second algorithm, Branch and Bound Skyline, uses a much smarter approach. Instead of checking houses individually, I organize them into a spatial tree structure called an R-tree. Each node in this tree has a Minimum Bounding Rectangle that defines the cost and size ranges of all houses in that region. Here's the key insight: if I find a skyline house that costs $100k with 200 square meters, I can immediately skip any tree node labeled 'houses costing $150k-$300k' because all houses in that region are dominated. This spatial pruning dramatically reduces the number of comparisons needed. I use a priority queue to always examine the most promising regions first, based on a distance heuristic. The R-tree is built using bulk loading, which sorts houses by cost and groups them efficiently. This algorithm achieved a 9x speedup over sequential scan."

**Slide 9: Algorithm 3 - Divide and Conquer BBS**

**Title:** Divide-and-Conquer: Split, Solve, Merge

**Content:**

* **Strategy:** Divide dataset by cost, solve each half separately
* **Process:** Sort → split at median → apply BBS to each half → merge results
* **Advantage:** Consistent performance, parallelizable
* **Final Step:** 1D dominance screening to merge skylines

**Visual Suggestion:** Flow diagram showing split → solve → merge process

**Speaker Notes (90 seconds):** "The third algorithm combines divide-and-conquer strategy with BBS. I sort all houses by cost and split them at the median into 'cheap houses' and 'expensive houses'. Then I apply the BBS algorithm to each half independently, finding the skyline for each subspace. The interesting part is the merging step - I can't just combine the two skylines because a cheap house might dominate an expensive one. After merging both skylines, I need to filter the results because a cheap house from the left half might dominate an expensive house from the right half. So I apply our standard skyline checking to the merged candidates to ensure the final result contains only non-dominated points."

In my implementation, this achieved a 6x speedup over sequential scan."

**Slide 6: Implementation Details**

**Title:** Technical Implementation Highlights

**Content:**

* **Data Structure:** Tuples (ID, cost, size) for efficiency
* **R-tree Construction:** Bulk loading with 100 entries per node
* **Priority Queue:** Python heapq for optimal node processing
* **Error Handling:** Comprehensive validation and file I/O safety
* **No External Dependencies:** Pure Python implementation

**Speaker Notes (75 seconds):** "Let me briefly cover some key implementation details. I used simple tuples to represent houses, which is more memory-efficient than custom classes. For the R-tree, I chose bulk loading over incremental insertion because it's optimal for static datasets like ours. I set the node capacity to 100 entries, which provides good balance between tree depth and pruning effectiveness. Python's heapq module handles the priority queue efficiently, always giving me the most promising node to examine next. I included comprehensive error handling for file operations and data validation. Importantly, my implementation uses only standard Python libraries - no external dependencies like the rtree library, making it completely self-contained and portable."

**Slide 7: Performance Results**

**Title:** Performance Analysis & Comparison

**Content:**

* **Sequential Scan:** 0.9105 seconds (baseline)
* **BBS Algorithm:** 0.1005 seconds (9.1x faster)
* **Divide-and-Conquer:** 0.1474 seconds (6.2x faster)
* **All algorithms found same 8 skyline points**
* **Dataset:** 300,000 real estate points

**Visual Suggestion:** Bar chart showing execution times and speedup ratios

**Speaker Notes (90 seconds):** "Now for the results that validate our approach. All three algorithms correctly identified the same 8 skyline points, confirming algorithmic correctness. The performance differences are dramatic: Sequential scan took 0.91 seconds as our baseline. BBS achieved 0.10 seconds, which is a 9.1x speedup - excellent performance that demonstrates the power of spatial indexing. Divide-and-conquer achieved 0.15 seconds for a 6.2x speedup, which is still very good and shows the benefit of problem decomposition. These speedup ratios are realistic for a 300,000 point dataset and demonstrate that spatial optimization techniques can provide significant performance improvements. The fact that all algorithms found identical skyline points gives us confidence in the correctness of our implementations."

**Slide 8: Real-World Skyline Points**

**Title:** Discovered Skyline Houses

**Content:**

* **House 14825:** $100k, 331m² (cheapest option)
* **House 50171:** $100k, 424m² (great value)
* **House 88124:** $413k, 500m² (largest house)
* **House 105741:** $100k, 500m² (best overall value)
* **4 other optimal trade-off points**

**Speaker Notes (60 seconds):** "Let me show you the actual skyline points we discovered. House 14825 represents the absolute cheapest option at $100k for 331 square meters. House 105741 offers incredible value at $100k for 500 square meters - nearly the largest size at the lowest price point. At the other extreme, House 88124 gives you the maximum size of 500 square meters, but you'll pay $413k for it. Each of these 8 houses represents a different optimal trade-off point. No other house in the entire 300,000-point dataset can beat these houses in both dimensions simultaneously."

**Slide 9: Real-World Applications** **Title:** Real-World Applications

**Content:**

* **Car Shopping:** Price vs fuel efficiency vs safety rating
* **Restaurant Selection:** Cost vs rating vs distance from home
* **Job Selection:** Salary vs work-life balance vs career growth potential
* **University Choice:** Tuition cost vs academic ranking vs location preference

**Speaker Notes (75 seconds):** "The skyline algorithm solves real optimization problems we face daily. When buying a car, you want low price, high fuel efficiency, and good safety ratings - skyline algorithms find cars that aren't beaten by others in all three criteria. For restaurants, you might consider cost, rating, and distance, finding places that offer the best trade-offs. Job seekers face similar decisions, balancing salary offers against work-life balance and career growth opportunities - skyline analysis would show only jobs that aren't dominated in all three areas. Students choosing universities optimize between tuition costs, academic rankings, and location preferences. In each case, skyline algorithms eliminate dominated options and present only the truly competitive choices, making decision-making much easier when facing multiple competing criteria."

**Slide 10: Conclusions**

**Title:** Key Takeaways

**Content:**

* **Spatial indexing dramatically improves performance** (9x speedup)
* **R-trees enable effective pruning** for multi-dimensional problems
* **Algorithm choice matters** based on dataset characteristics
* **Implementation simplicity** can be more valuable than complexity
* **Skyline algorithms solve real optimization problems**

**Speaker Notes (60 seconds):** "To conclude, this project demonstrated several key insights. First, spatial indexing techniques like R-trees can provide dramatic performance improvements - our 9x speedup shows the value of smart data organization. Second, the choice of algorithm matters and should match your dataset characteristics and requirements. Third, sometimes simpler implementations are better - my bulk-loaded R-tree outperformed more complex dynamic approaches. Finally, skyline algorithms solve genuine real-world optimization problems and provide a principled approach to multi-criteria decision making. The techniques we explored here have applications far beyond real estate and represent fundamental concepts in computational geometry and database optimization."