**Karlo Robert C. Wagan Real- World Applications of Algorithm Strategies**

**BSCS - 3 ITP 6**

**Final Requirements**

**Real-World Applications of Algorithm Strategies**

**Executive Summary**

This report analyzes five algorithmic strategies through implementation of classic problems, examining their performance characteristics and real-world applications.

**1. Brute Force: Traveling Salesman Problem**

**Implementation:** Generates all possible city permutations to find the minimum distance path.

**Results:**

A graph with a line going up

AI-generated content may be incorrect.

**Complexity:**

* Time: O(n!) - factorial growth
* Space: O(n)

**Performance:** Execution time increases dramatically with city count:

* 5 cities: 0.0001s
* 9 cities: 0.1022s

**When Most Useful:**

* Small problem instances (≤12 elements)
* When exact optimal solutions are required

**Real-World Applications:**

* PCB manufacturing drill path optimization
* Local delivery route planning
* Quality control inspection sequencing

**2. Divide and Conquer: Merge Sort**

**Implementation:** Recursively divides arrays in half, sorts, and merges them.

**Results:**

A graph with a line and a blue line

AI-generated content may be incorrect.

**Complexity:**

* Time: O(n log n) consistently
* Space: O(n)

**Performance:** Shows consistent scaling regardless of input pattern:

* 10,000 elements: ~0.052s
* 100,000 elements: ~0.16s

**When Most Useful:**

* Large datasets requiring consistent performance
* Applications where sort stability matters
* Parallelizable processing

**Real-World Applications:**

* Database query result sorting
* External file sorting
* Version control system merging

**3. Decrease and Conquer: Binary Search**

**Implementation:** Repeatedly divides search interval in half to locate target value.

**Results:**

A graph with a line and a line

AI-generated content may be incorrect.

**Complexity:**

* Time: O(log n)
* Space: O(1)

**Performance:** Dramatically outperforms linear search as data size increases:

* 1,000,000 elements: Binary search (0.00001144s) vs. Linear search (0.032s)

**When Most Useful:**

* Frequent lookups in sorted datasets
* Applications with strict timing requirements
* Memory-constrained environments

**Real-World Applications:**

* Dictionary implementations
* Database indexing structures
* IP routing tables
* Autocomplete systems

**4. Transform and Conquer: Gaussian Elimination**

**Implementation:** Transforms linear equation system through row operations with partial pivoting.

**Results:**

A graph with a line going up

AI-generated content may be incorrect.

**Complexity:**

* Time: O(n³)
* Space: O(n²)

**Performance:** Execution time increases cubically with system size:

* 10×10 system: 0.0003s
* 200×200 system: 0.0405s

**When Most Useful:**

* Systems of linear equations requiring exact solutions
* Problems that can be modeled as linear systems
* Applications requiring high numerical precision

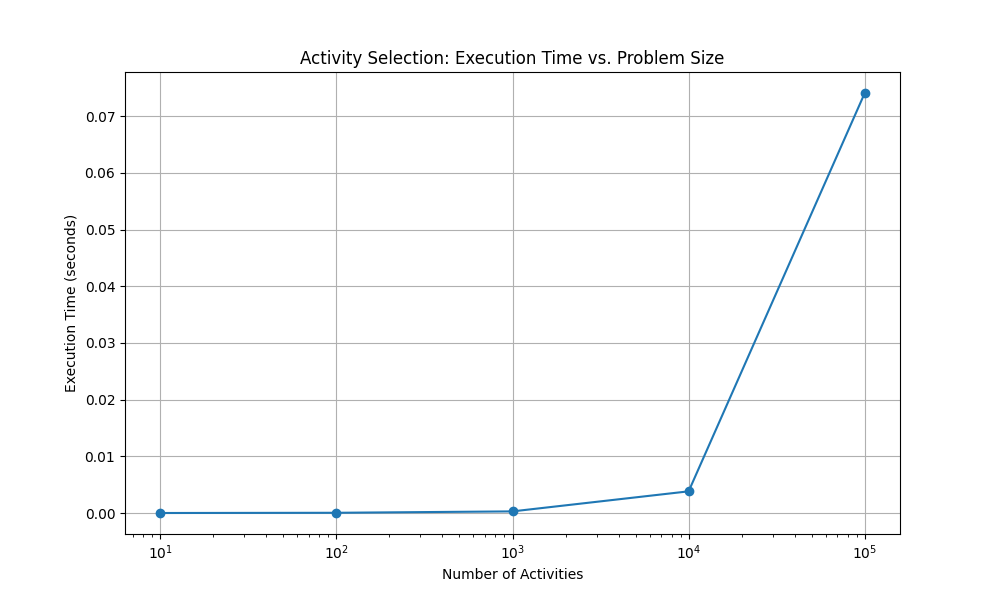
**Real-World Applications:**

* Structural engineering analysis
* Electrical circuit analysis
* Computer graphics transformations
* Economic modeling

**5. Greedy Algorithm: Activity Selection**

**Implementation:** Sorts activities by finish time and selects compatible activities.

**Results:**



A graph with a line

AI-generated content may be incorrect.

**Complexity:**

* Time: O(n log n) - dominated by sorting
* Space: O(n)

**Performance:** Scales efficiently with problem size:

* 10,000 activities: 0.0038s, selecting ~30% of activities

**When Most Useful:**

* Optimization with sequencing constraints
* Resource allocation with time limitations
* Problems where local optimality leads to global optimality

**Real-World Applications:**

* Meeting room scheduling
* CPU task scheduling
* Network packet management
* Transportation timetabling

**Comparative Analysis**

| **Strategy** | **Algorithm** | **Time Complexity** | **Space Complexity** | **Key Advantage** | **Size Limit** |
| --- | --- | --- | --- | --- | --- |
| Brute Force | TSP | O(n!) | O(n) | Optimal solution | ≤12 elements |
| Divide & Conquer | Merge Sort | O(n log n) | O(n) | Consistent performance | Millions |
| Decrease & Conquer | Binary Search | O(log n) | O(1) | Extremely efficient | Billions |
| Transform & Conquer | Gaussian Elimination | O(n³) | O(n²) | Handles complex systems | Thousands |
| Greedy | Activity Selection | O(n log n) | O(n) | Fast scheduling | Millions |

**Conclusion**

Each algorithmic strategy presents distinct advantages:

* **Brute Force** provides guaranteed optimal solutions for small problems
* **Divide and Conquer** handles large datasets consistently and enables parallelization
* **Decrease and Conquer** achieves exceptional efficiency for search operations
* **Transform and Conquer** excels at mathematical problems through reformulation
* **Greedy Algorithms** offer practical solutions balancing performance with computational needs

The optimal choice depends on problem characteristics, dataset size, performance requirements, and implementation context. Real-world applications often benefit from hybrid approaches combining multiple strategies.