

Project 4 Report

1. Analyze your dynamic programming algorithm code mathematically to determine its big-O efficiency class, probably $O(n^2)$ or $O(n \log n)$.

Handwritten analysis of a dynamic programming algorithm for finding the minimum cost path in a grid. The code is annotated with line numbers and complexity values.

```

Std::unique_ptr<RideVector> dynamic_max_time(const RideVector &rides, int total_cost) {
    Std::unique_ptr<RideVector> final(new RideVector);
    int n = rides.size();
    Std::vector<Std::vector<double>> T(n+1, Std::vector<double>(total_cost+1));

    for (int x=0; x<=n; x++) {
        for (int y=0; y<=total_cost; y++) {
            T[x][y] = 0;
        }
    }

    for (int x=1; x<=n; x++) {
        for (int y=1; y<=total_cost; y++) {
            if (rides[x-1].cost() <= y) {
                T[x][y] = std::max(T[x-1][y], T[x-1][y-rides[x-1].cost()] + rides[x-1].time());
            } else {
                T[x][y] = T[x-1][y];
            }
        }
    }

    double result = T[n][total_cost];
    int cost = total_cost;

    for (int i=n; i>0 && result > 0 && cost > 0; i--) {
        if (result != T[i-1][cost]) {
            answer.push_back(rides[i-1]);
            result = T[i-1][cost-rides[i-1].cost()];
            cost = cost-rides[i-1].cost();
        }
    }

    Return answer;
}

```

Complexity analysis (S.C.):

- 1 = 1 x 8065 = 8065
- 2 = 8065 x 8065 = 65,044,225
- 3 = 2 + max(7, 2) = 2 + 7 = 9
- 4 = n x 9 = 9n
- 5 = n x 9n = 9n²
- 6 = 2 + max(8, 0) = 2 + 8 = 10
- 7 = n x 10 = 10n
- 8 = 5 + 10n + 9n² + 65,044,225 ⇒ $O(n^2)$ Time Complexity

2. Analyze your exhaustive optimization algorithm code mathematically to determine its big-O efficiency class, probably $O(2^n \cdot n)$.

```

auto todo = rides; SC 1
int n= rides.size(); SC 1
if(n >= 64){SC 1
    return nullptr; SC 0
else
    assert(n < 64); SC 1
    std::unique_ptr<RideVector> best(new RideVector); SC 0
    for 0 to 2^n do SC 2^n + 1
        candidate = empty(); SC 1
        for 0 to n-1 do { SC n
            if(((i >> j) & 1) == 1) SC 3
                candidate.push_back(todo[j]); SC 1
            if (total_cost(candidate) <= total_cost(best)) SC 1
                if(best.empty() || total_time(candidate) > total_time(best)) SC 2
                    *best = *candidate; SC 1

```

Step count: first for loop

$$(2^n + 1) \cdot 1 \cdot n \cdot (3 + 1) \cdot (1 + 2 + 1) \\ = (2^n + 1) \cdot 16n$$

Proof:

$$(2^n + 1) \cdot 16n$$

$16n \cdot 2^n + 16n$ belongs to $O(2^n \cdot n)$

Find $c > 0$ and $n_0 > n$ st $16n \cdot 2^n + 16n$

Choose $c = 16 + 16 = 32$

$$16n \cdot 2^n + 16n \leq 32 \cdot 2^n \cdot n$$

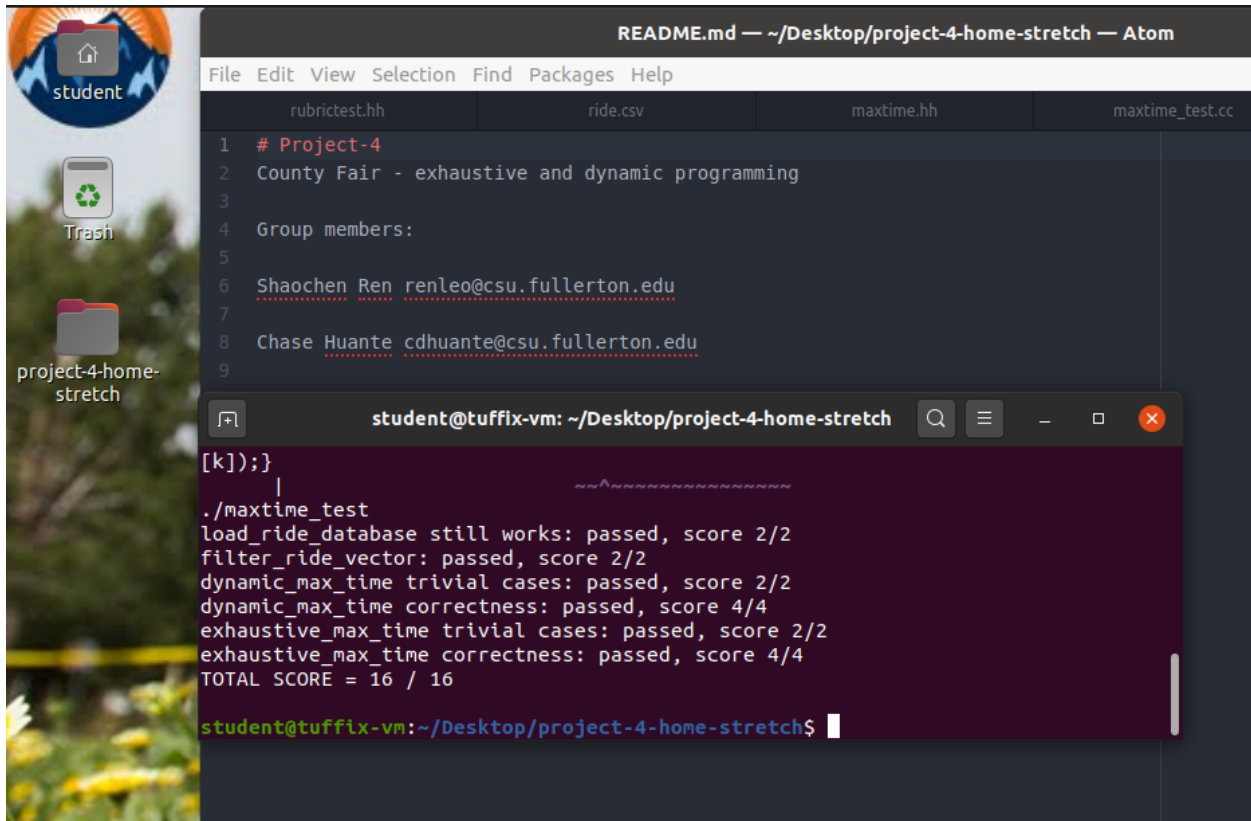
$$16 \cdot 2^n \cdot n - 16n \cdot 2^n + 16 \cdot 2^n \cdot n - 16n \geq 0 \quad \forall n \geq 0 \quad n_0 = 1$$

By definition that $16n \cdot 2^n + 16n$ belongs to $O(2^n \cdot n)$

1. Your names, CSUF-supplied email address(es), and an indication that the submission is for project 4.

Shaochen Ren renleo@csu.fullerton.edu

Chase Huante cdhuante@csu.fullerton.edu



The screenshot shows a Linux desktop environment. On the left, there is a sidebar with icons for 'student' (a blue circle with a white house icon), 'Trash' (a green recycling symbol), and 'project-4-home-stretch' (a folder icon). The main area displays a code editor window titled 'README.md — ~/Desktop/project-4-home-stretch — Atom'. The editor shows a file named 'README.md' with the following content:

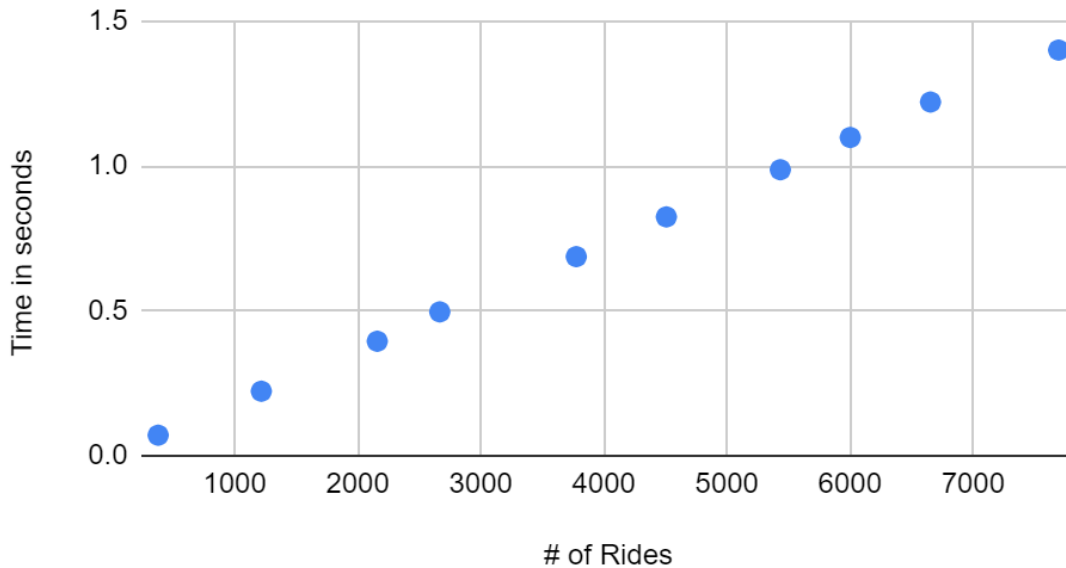
```
1 # Project-4
2 County Fair - exhaustive and dynamic programming
3
4 Group members:
5
6 Shaochen Ren renleo@csu.fullerton.edu
7
8 Chase Huante cdhuante@csu.fullerton.edu
9
```

Below the code editor, there is a terminal window titled 'student@tuffix-vm: ~/Desktop/project-4-home-stretch'. The terminal shows the following output:

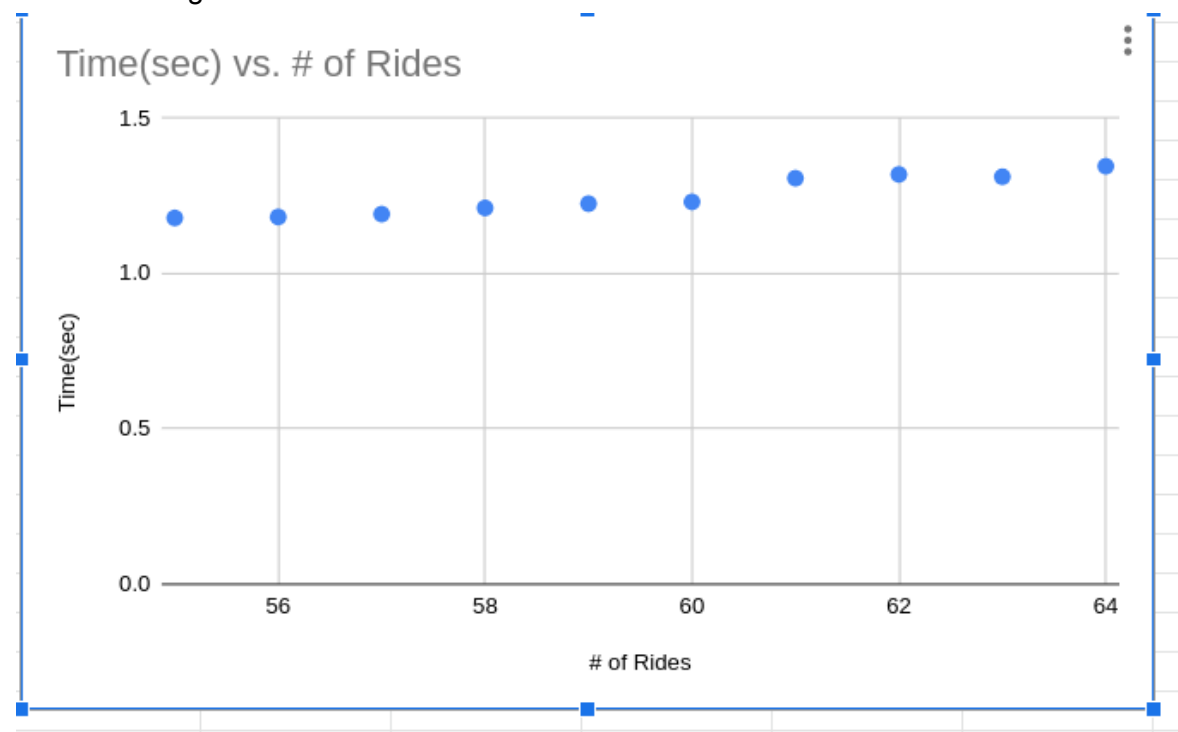
```
[k]);}
|
./maxtime_test
load_ride_database still works: passed, score 2/2
filter_ride_vector: passed, score 2/2
dynamic_max_time trivial cases: passed, score 2/2
dynamic_max_time correctness: passed, score 4/4
exhaustive_max_time trivial cases: passed, score 2/2
exhaustive_max_time correctness: passed, score 4/4
TOTAL SCORE = 16 / 16
student@tuffix-vm:~/Desktop/project-4-home-stretch$
```

2. Two scatter plots meeting the requirements stated above.

Dynamic Algorithm



Exhaustive Algorithm



3. Answers to the following questions, using complete sentences.

- a. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?

To no surprise the dynamic algorithm computes faster than the exhaustive search. The reason being is that $O(2^n \cdot n)$ is exponentially slower than $O(n^2)$. The reason why this isn't a surprise

is because this project is teaching how successful polynomial time with dynamic programming can be used.

- b. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.

When comparing the empirical analysis to the mathematical analysis they both correlate to a similar median with a few outside outliers. An example is how our greedy function is an $O(n \log n)$ search which can relate to a graph when substituting enough n values for $o(n \log(n))$ with a slope of zero.

- c. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.

This is correct in the sense of providing a correct output but ultimately not feasible to implement. What project 4 shows is that the same problem can be solved and provided the same answer without the exponential time complexity

- d. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.

Hypothesis 2 is correct. Exhaustive for example. Once we entered an exponential time complexity when looping all subsets of an n -element sequence, the time considerably jumped per ride. To the knowledge given in the class, $O(c^n)$ is the highest time complexity.