Chase Huante
Shaochen Ren

Team Cactus Project 2 Report

Finally, produce a brief written project report ***in PDF format***. Your report should be submitted as a checked-in file in GitHub. Your report should include the following:

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

GREEDY

```
Std:unique_ptr<RideVector> greedy_max_time(const RideVector& rides, double total_cost)
  auto todo = rides; ---------------------------------------- 1
  Std::unique_ptr<RideVector> result(new RideVector); ------- 0
  double result_cost = 0; ----------------------------------- 1
  while (!todo.empty()) { ----------------------------------- ? (log(n)+1)
      auto start = todo[0] ---------------------------------- 1
      auto iter = todo.begin(); ----------------------------- 1
      auto current = iter; ---------------------------------- 1
      for (auto c: todo) { ---------------------------------- n
        if (c->rideTime()/c->cost() > start->rideTime()/start->cost()) { -- 3  If
            start = c; ---------------------------------------------------- 1
            Current = iter; ----------------------------------------------- 1
        }
        iter++ ------------------------------------------------------------ 1
      }
      todo.erase(current);
        if (result_cost + start->cost() <= total_cost) { ---------------- 2  IF
            result->push_back(start); ------------------------------------ 1
            resut_cost += start->cost(); --------------------------------- 1
        }
  }
```

1 = 2 + max(2,0) = 2+2 = 4

2 = 3 + max (2,0) = 3+2 = 5

3 = (n-64+1) x (5+1) = 63n x 6 = 378n

4 = (log n) x (63n x 6) = 63n x 6 log n

5 = 2 + 63n x 6 log(n) = $\boxed{O(n \log(n))}$

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

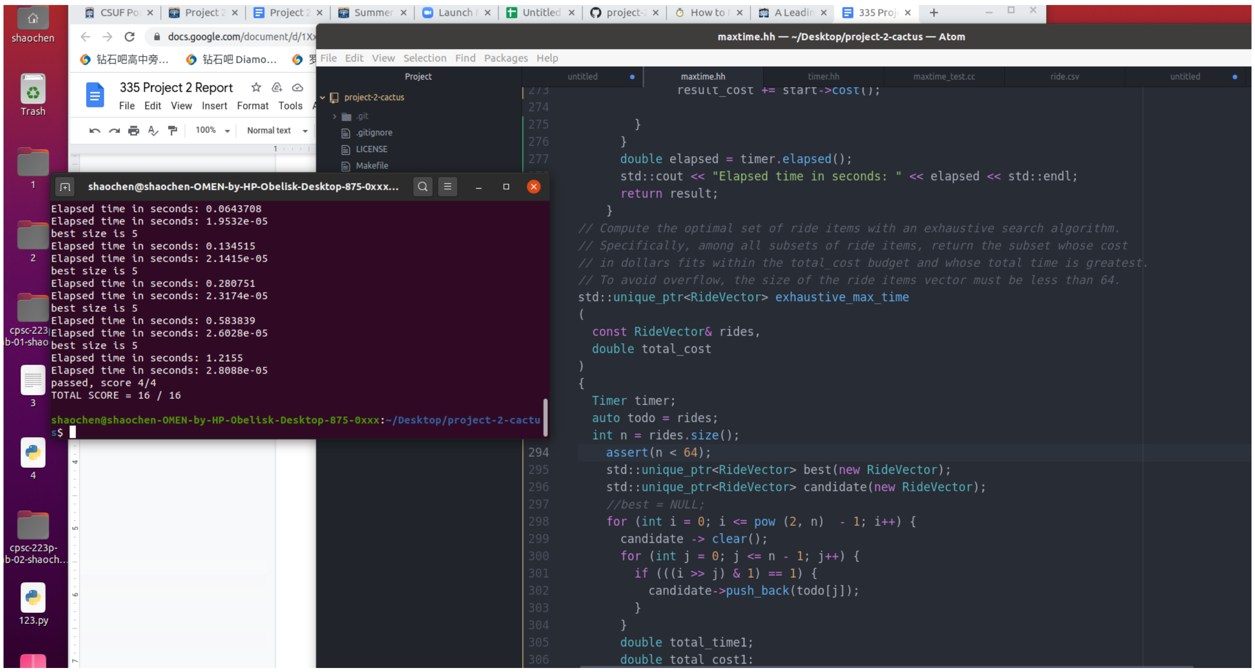
```
auto todo = rides; SC 1
int n= rides.size(); SC 1
if(n  >= 64){SC 1
        return nullptr;SC 0
else
        assert(n < 64);SC 1
        std::unique_ptr<RideVector> best(new RideVector); SC 0
        for  0 to 2^n doSC 2^n +1
                candidate = empty(); SC 1
                for 0 to n-1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

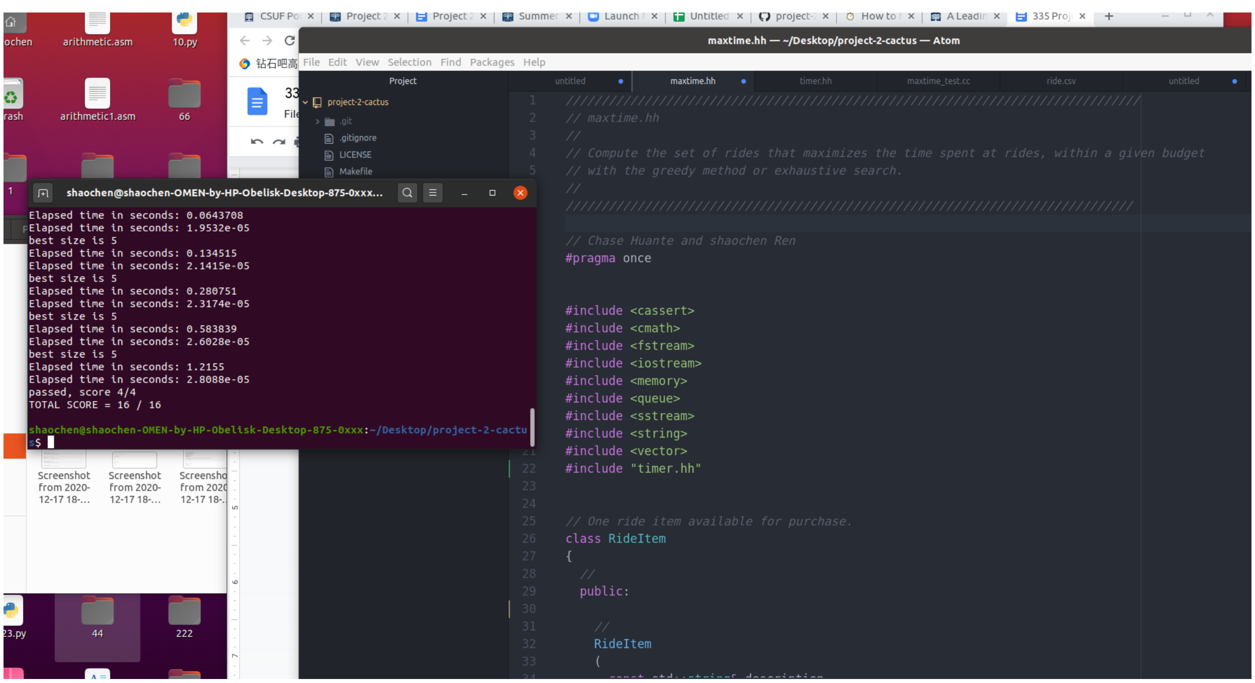
Step count: first for loop
(2^n +1)* 1 * n *(3+1)*(1+2+1)
= (2^n+1)*16n

Proof:
(2^n+1)*16n
16n*2^n+16n belongs to O(2^n*n)
Find c > 0 and n0> n st 16n*2^n+16n
Choose c = 16+16 = 32
16n*2^n+16n <= 32*2^n*n
16*2^n*n-16n*2^n+16*2^n*n-16n >=0 V n>=0 n0=1
Bu definition that 16n*2^n+16n belongs to O(2^n*n)

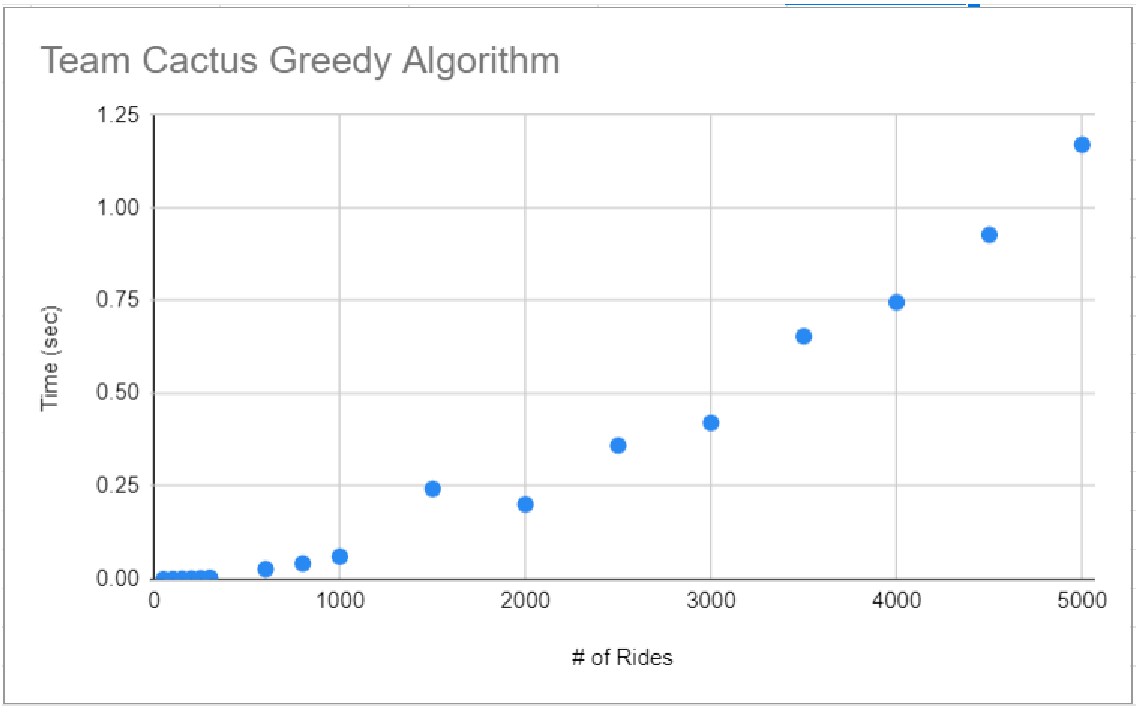
1. Your names, CSUF-supplied email address(es), and an indication that the submission is for project 2.

Chase Huante cdhuante@csu.fullerton.edu
Shaochen Ren renleo@csu.fullerton.edu

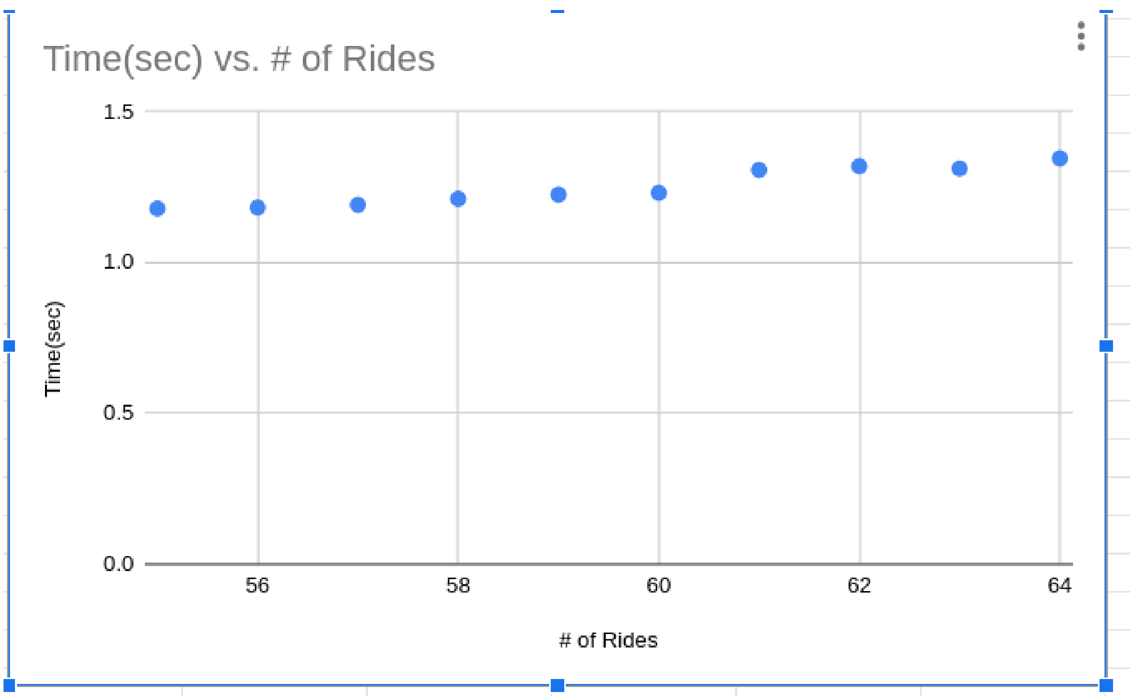




2. Two scatter plots meeting the requirements stated above.
Greedy



exhaustive_max_time



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?

When comparing the 2 functions, greedy is considerably faster than exhaustive search. Reason being is because of exhaustive's exponential power to n which is why we had to bring the list down to 64. Anything longer and the search could be too demanding. Although this would be a surprising moment, we've been anticipating these results the whole time so no we aren't surprised by the time discrepancy.

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(nlogn) search which can relate to a graph when substituting enough n values for o(nlog(n)) with a slope of zero.

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

Consistent to some degree but ideally inconsistent. The reason being is because exhaustive search produces a more correct answer than greedy but with real world implications we cannot always use exhaustive search whether it be time or resource constraints.

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.