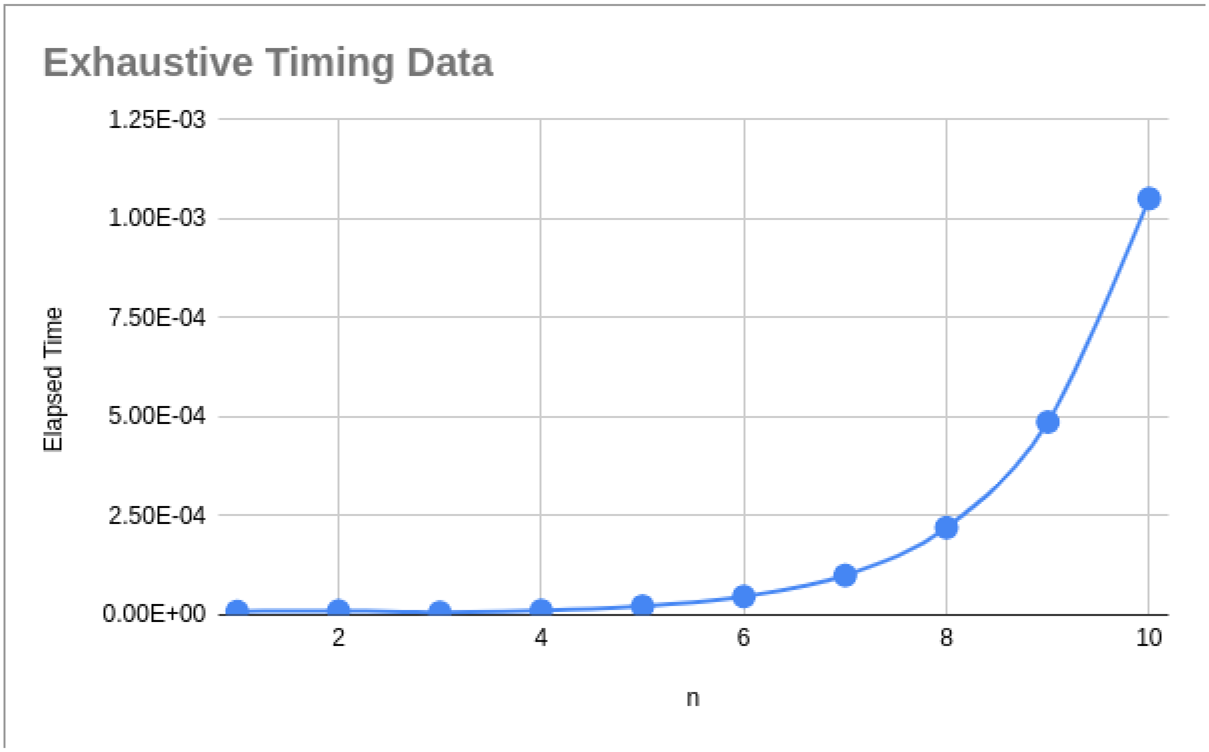
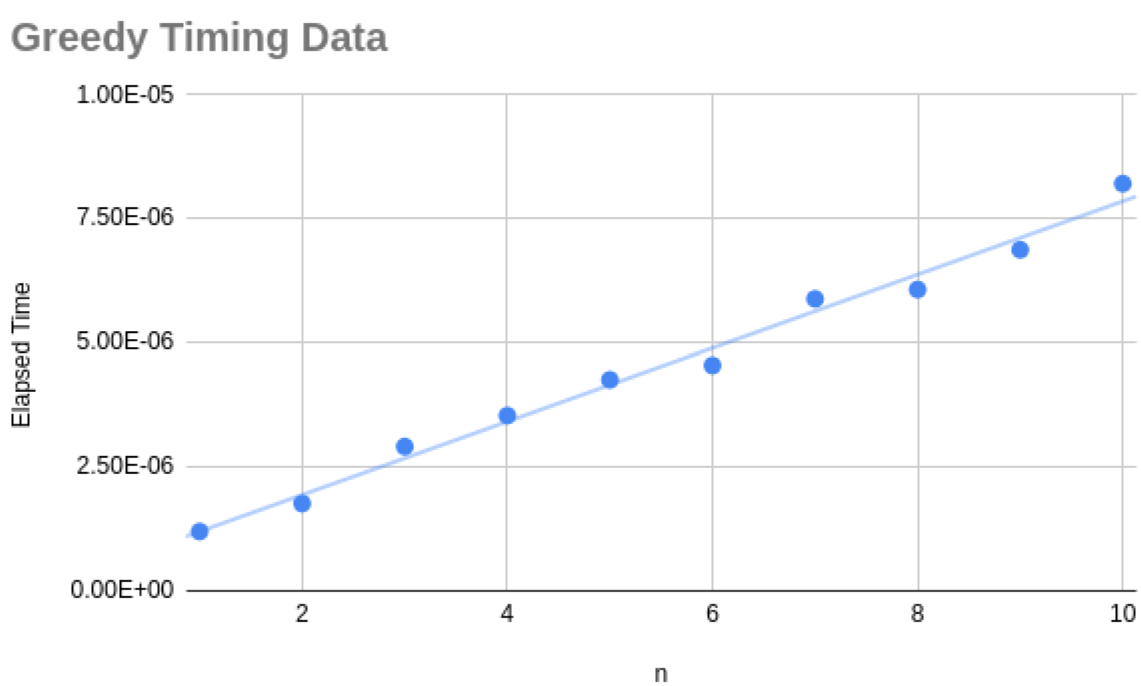
Project 2 report

Yesh Patel yesh@fullerton.edu

Arqum Ahmed ArqumAhmed@fullerton.edu


Exhaustive Timing Data
1.25E-03
1.00E-03
7.50E-04
5.00E-04
2.50E-04
0.00E+00
Elapsed Time
2
4
6
8
10
n


Greedy Timing Data
1.00E-05
7.50E-06
5.00E-06
2.50E-06
0.00E+00
Elapsed Time
2
4
6
8
10
n

335 Project 2 Answers

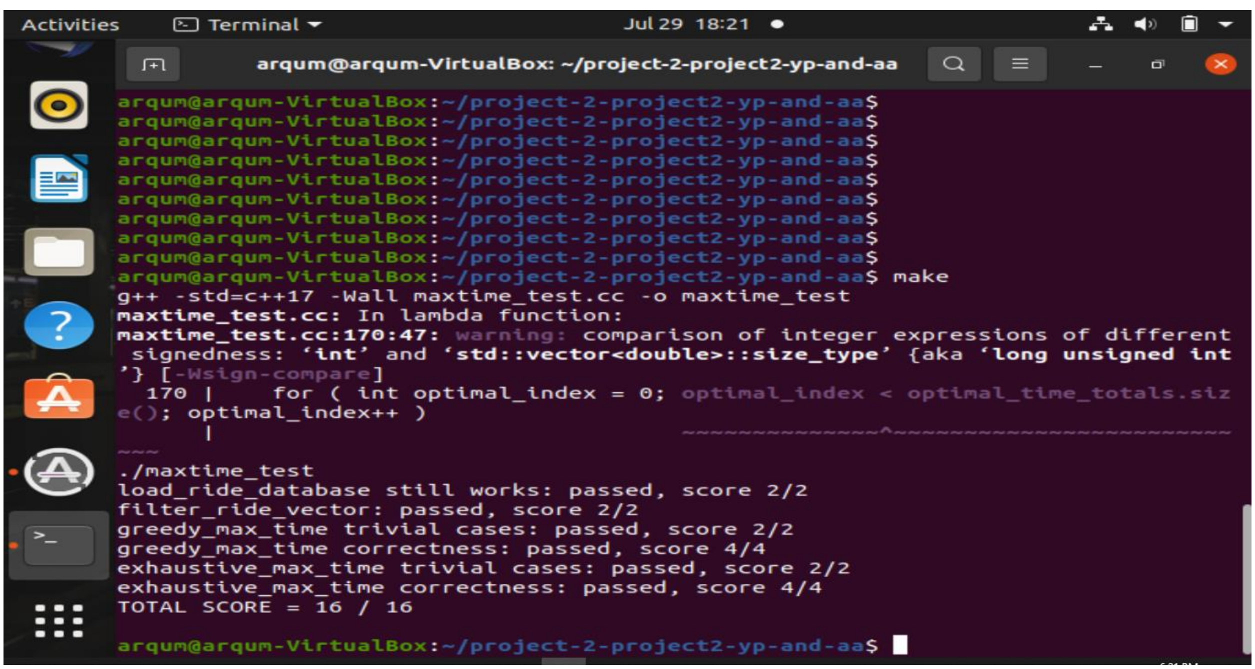
3a. There is a noticeable difference between the performance of two algorithms. The greedy algorithm is faster. Greedy algorithm runs in a quadratic manner while, exhaustive algorithm runs in an exponential manner. It is not a surprise at all.

3b. The empirical analysis is consistent with mathematical analysis because empirical analysis shows that greedy algorithm is faster with number of inputs and big-oh for exhaustive algorithm 2^n*n grows faster than the big-oh x^2. Both tells us the same thing that greedy algorithm is faster than exhaustive algorithm with growing number of input sizes.

3c. The evidence is consistent with hypothesis 1 because they provide the optimal solution for the problem by permutating over each set and finding the one that is the best for the scenario making it a feasible algorithm.

3d. The evidence is consistent with hypothesis 2 because with increasing number of input size the algorithm takes more time meaning if the input size is big enough it will be slow enough to provide results in any kind of reasonable time making it unreasonable to use it for practical use.

Screenshot of the result:



Psuedocode and Mathematical Analysis

Greedy Algorithm

Input: A positive "dollar amount" budget C (integer number of dollar coins), and a vector V of n "ride" objects, containing one or more ride objects where each ride object a = (c, t) has an integer cost of dollars c > 0 and time in minutes t >= 0

Output: A vector K of ride objects drawn from V, such that the sum of costs of the ride items from K is within the prescribed dollar budget C and the sum of the rides time is maximized.

```
result = None                                  / 1
result_cost = 0                                / 1
time_cost = 0                                  / 1
index = 0                                      / 1
while ( n > 0 )                                / n times
    for i=0 to n-1 do                          / (n-1+1) = n
        if ((V[i] -> t / V[i] -> c) > time_cost)  / 4+4=8
            time_cost = a[i] -> t / a[i] -> c  / 3 ] 4
            index = i                          / 1 ]
        endif
```

```
    endfor
    time_cost = 0                                   / 1
    if ((result_cost + V[index] -> c) <= c)         / 3+3 = 6
        result -> push_back(v[index])               / 1 ]
        result_cost += V[index] -> c                / 2 ] 3
    endif
    V.erase(v.begin() + index)                      / 2
endwhile
Return K
```

$S.C = 4 + n \cdot (8n + 1 + 6 + 2)$

$= 4 + n \cdot (8n + 9)$

$= 4 + 8n^2 + 9n$

$= 8n^2 + 9n + 4$

Proof: $8n^2 + 9n + 4$

$\lim_{n \to \infty} \frac{(8n^2 + 9n + 4)'}{(n^2)'} \rightarrow \lim_{n \to \infty} \frac{(16n + 9)'}{(2n)'}$

$\lim_{n \to \infty} \frac{16}{2} = 8 \geq 0$ therefore $8n^2 + 9n + 4 \in O(n^2)$ and defined

Exhaustive Algorithm

Input: A positive "dollar amount" budget C (integer number of dollar coins), and a vector V of n "ride" Objects, containing one or more ride objects where each ride object a = (c, t) has an integer cost of dollars c > 0 and time in minutes t >= 0

Output: A vector K of ride objects drawn from V, such that the sum of costs of the ride items from K is within the prescribed dollar budget C and the sum of the rides time is maximized.

```
best: None                                / 1
for bits:=0 to 2^n - 1                    / (2n-1+1) = 2^n
  Candidate = None                        / 1
    for J:=0 to n-1                       (n-1+1) = n
      if(((bits >> J) & 1) == 1)          / 3 + max (1, 0) = 4
          Candidate.push_back(rides[J])   / 1
      endif
    endfor
      Candidate_cost = 0                  / 1
      Candidate_time = 0                  / 1
      best_Cost = 0                       / 1
      best_time = 0                       / 1
```

```
Sum_ride-vector(candidate, candidate.cost, candidate_time) // 1
Sum_ride_vector(best, best.cost, best_time) // 1
if (candidate_cost <= total_cost)    1+max(4,0)=5
    if (best -> empty || candidate_time > 3+max(1)=4
            best_time)
        best = candidate  // 1
    endif
endif
endfor
Return best
```

$S.C = 1 + 2^n(1+4n+4+2+5)$

$= 1 + 2^n(4n+12)$

$= 2^n 4n + 2^n 12 + 1$