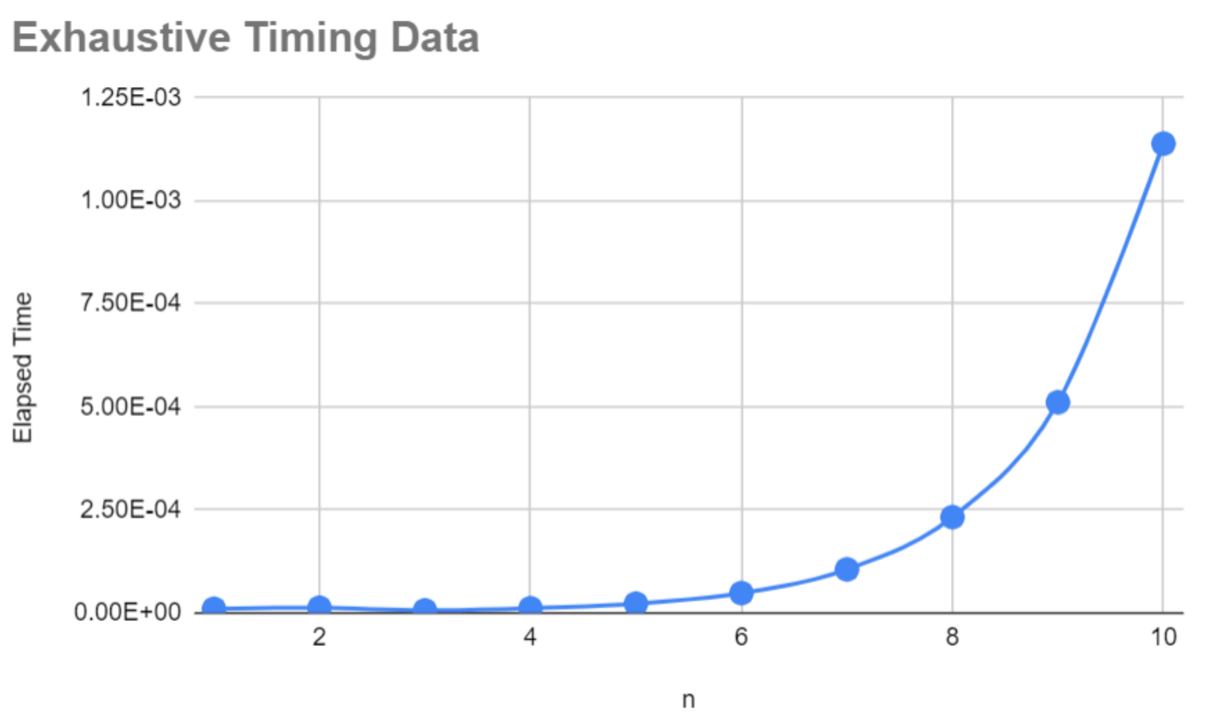
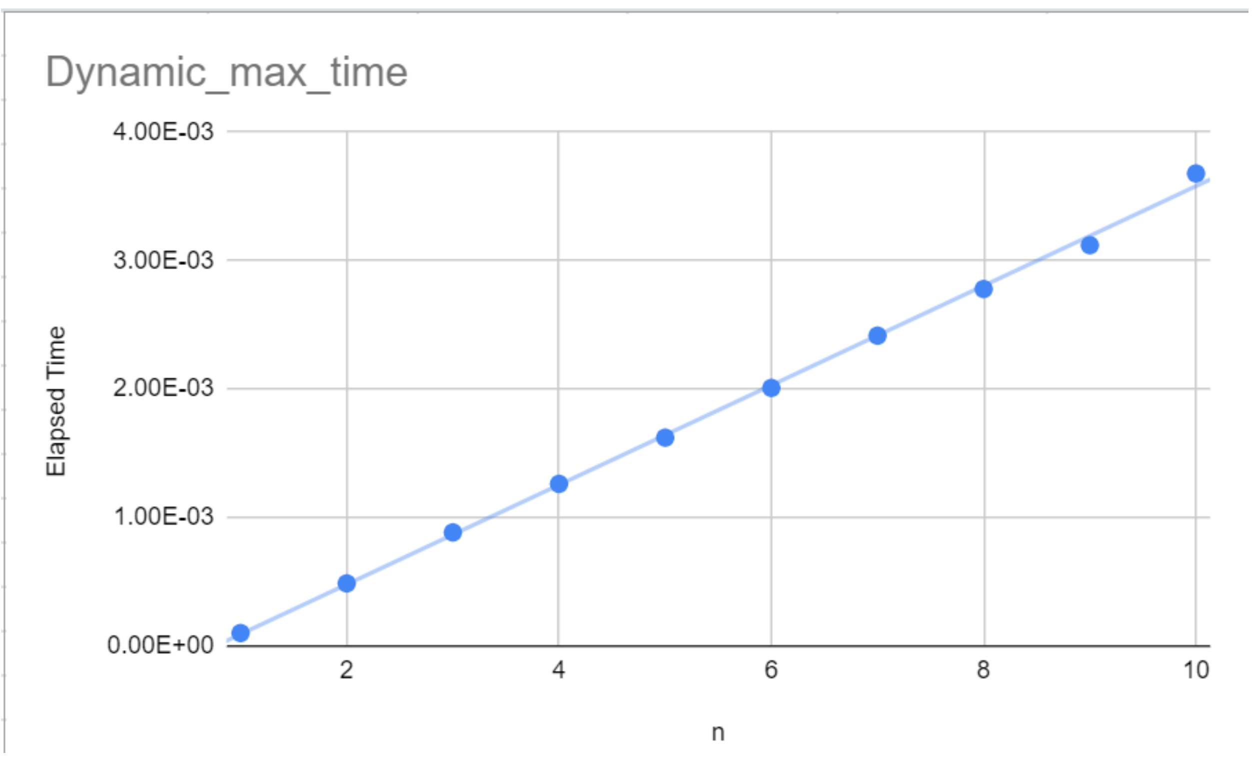
Yesh Patel
Arqum Ahmed
Project 4 report





Questions

Answers to the following questions, using complete sentences.

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

There is a noticeable difference between the performance of two algorithms. Dynamic algorithm is faster because it is Quadratic growth while exhaustive is exponential. It is not a surprise at all.

is Exhaustive is faster because it goes exponentially while dynamic runs in a linear manner. It is not a surprise at all.

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

The empirical analysis is consistent with mathematical analysis because empirical analysis shows that the Dynamic algorithm is faster with number of inputs and big-oh representation compared to the Exhaustive algorithms.

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

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

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

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.

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
```

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

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

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

Dynamic max time

```
Ridevector result                / 1
Vector T                         / 1

for i=0 to n                     / (n-0+1) = n+1
   T.Push_back (vector double)   / 2
   for J=0 to n                  / (n-0+1) = n+1
      T.at(i).Push_back(0.0)     / 2
   endfor
endfor

for i=1 to n                     (n-1+1) = n
 for J=1 to W                    (n-1+1) = n
  if (J >= rides[i-1] -> cost())  / 3+max(15, 2) = 18
     if (T[i-1][J] >= T[i-1][J-rides[i-1] -> cost()] +
           rides[i-1] -> time())  / 8 + max(2, 7) = 15
         T[i][J] = T[i-1][J]  / 2
    else
         T[i][J] = T[i-1][J-rides[i-1] -> cost()] +
         rides[i-1] -> time()    / 7
     endif
  else
     T[i][J] = T[i-1][J]   / 2
```

```
      endif
    endfor
  endfor

col_shift = total_cost                      / 1
for n to 1 / ( (1-n)/(-1) + 1 ) = n
   if (T[i][col_shift] != T[i-1][col_shift]) / 2+max(s)
      result -> push_back (rides[i-1])      / 2          = 7
      col_shift -= rides[i-1] -> cost()     / 3
   endif
endfor
return result
```

$S.C = 2 + (n+1)(2n+2) + n(18n) + 1 + 7n$

$= 2 + 2n^2 + 2n + 2n + 2 + 18n^2 + 1 + 7n$

$= 20n^2 + 11n + 5$

Proof: $20n^2 + 11n + 5 \in n^2$

$$\lim_{n \to \infty} \frac{(20n^2)' + (11n)' + (5)'}{(n^2)'} \to \lim_{n \to \infty} \frac{(40n)' + (11)'}{(2n)'}$$

$$\to \lim_{n \to \infty} \frac{40}{2} = 20 \geq 0$$ Therefore, exist in n^2