Analysis of Execution times for finding the maximum number of activities for Activity Selection Problem using Brute Force and Greedy Algorithms

Syed Ahtsham Ul Hassan

04071813015

Assignment No: 02

Submitted to: Sir Akmal Khattak

26-Dec-2020

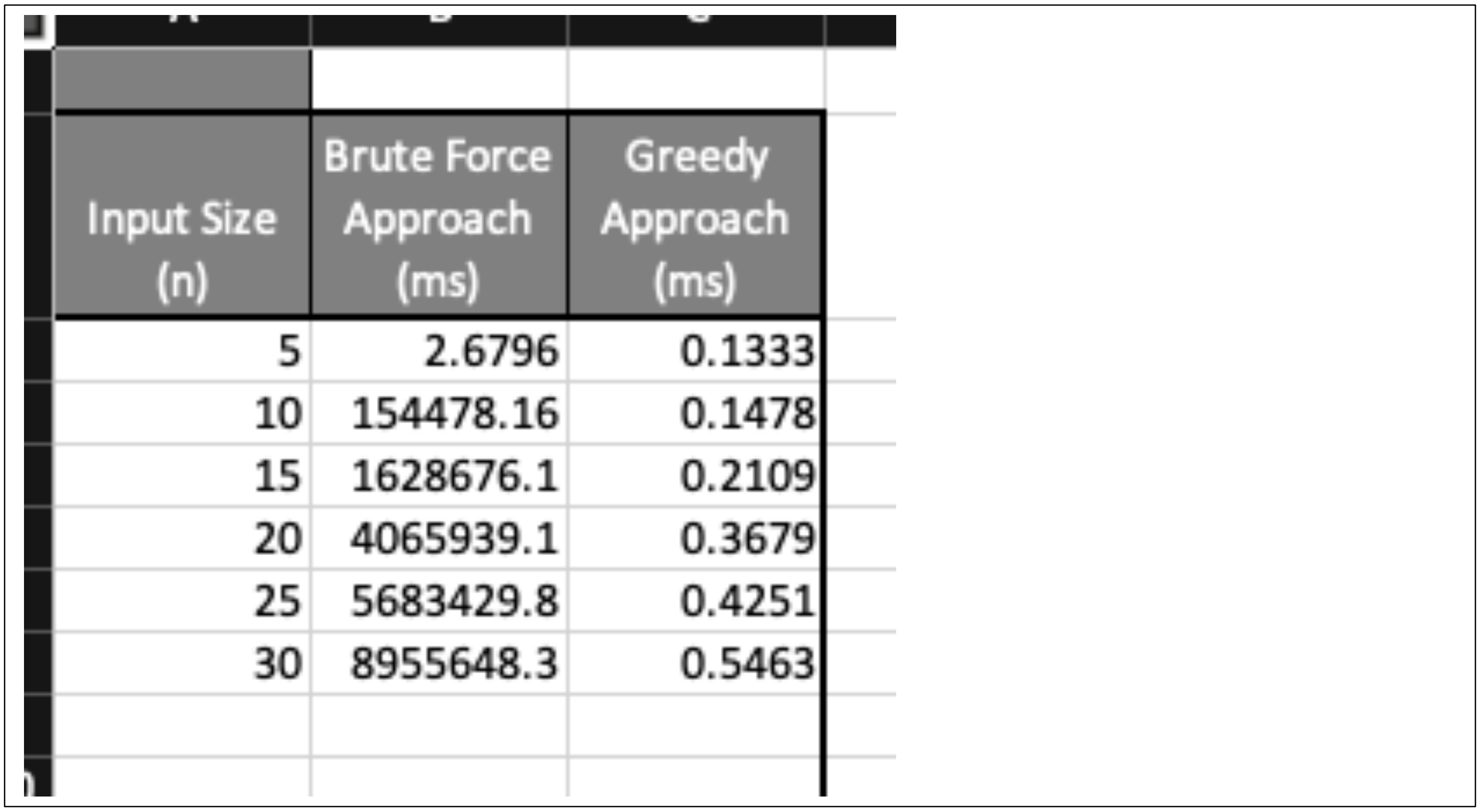
Analysis of the Brute Force and Greedy Approach for Activity Selection Problem

Execution Times for Brute Force and Greedy Approach for Activity Selection Problem is noted in the given following table. The program is run with the 6 set of different number of activities i.e., 5, 10, 15, 20, 25, and 30. The execution times are noted using time functions from chrono C++ libraries.

Brute Force Function's Time Complexity: 2^n

Greedy Approach Function's Time Complexity: n(logn)

Table for Execution Times in milliseconds

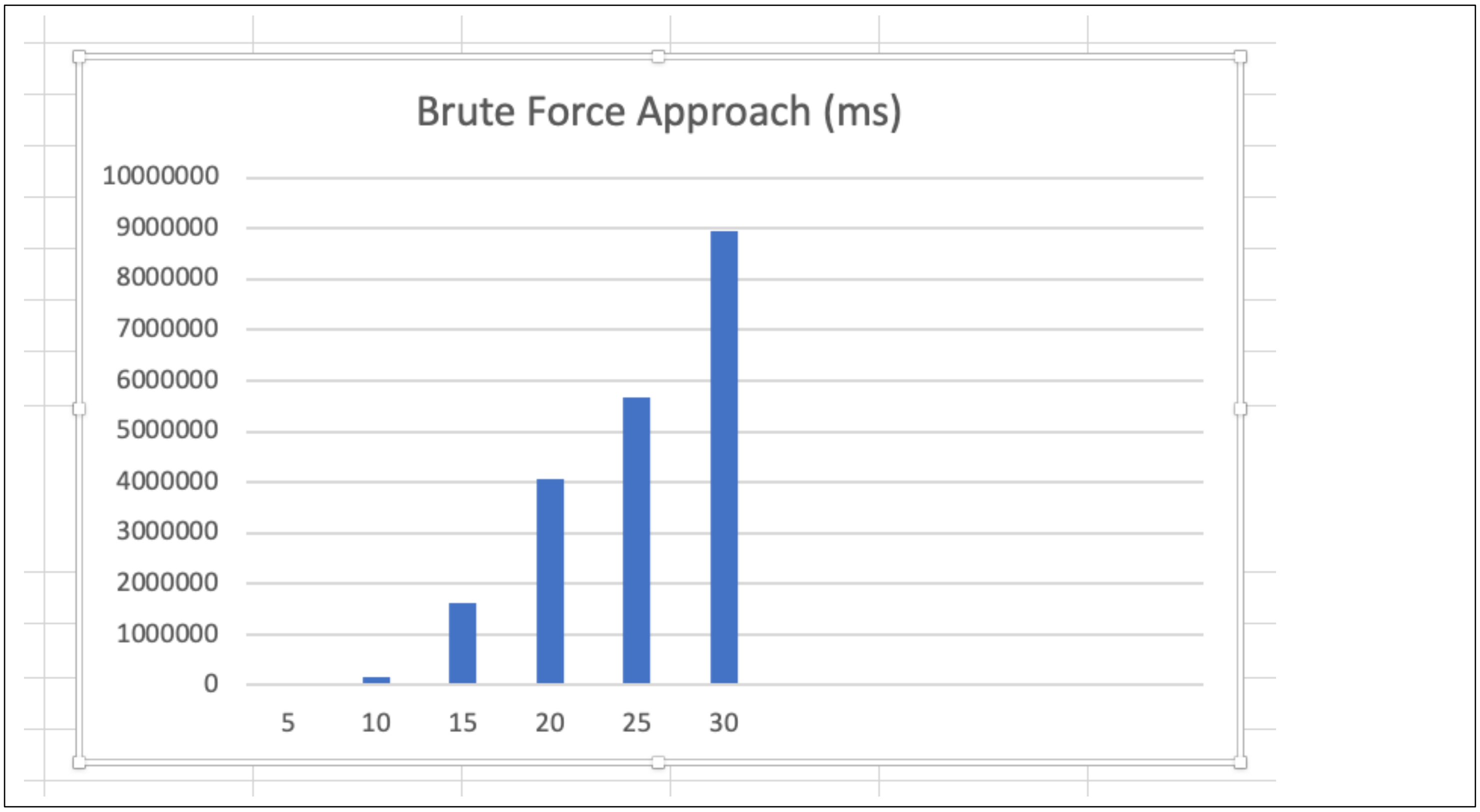


Input Size (n)	Brute Force Approach (ms)	Greedy Approach (ms)
5	2.6796	0.1333
10	154478.16	0.1478
15	1628676.1	0.2109
20	4065939.1	0.3679
25	5683429.8	0.4251
30	8955648.3	0.5463

Where, n = Input Size = Number of Activities

Brute Force Graph:

<u>Graph of Execution time (ms) vs Input Size (n)</u>



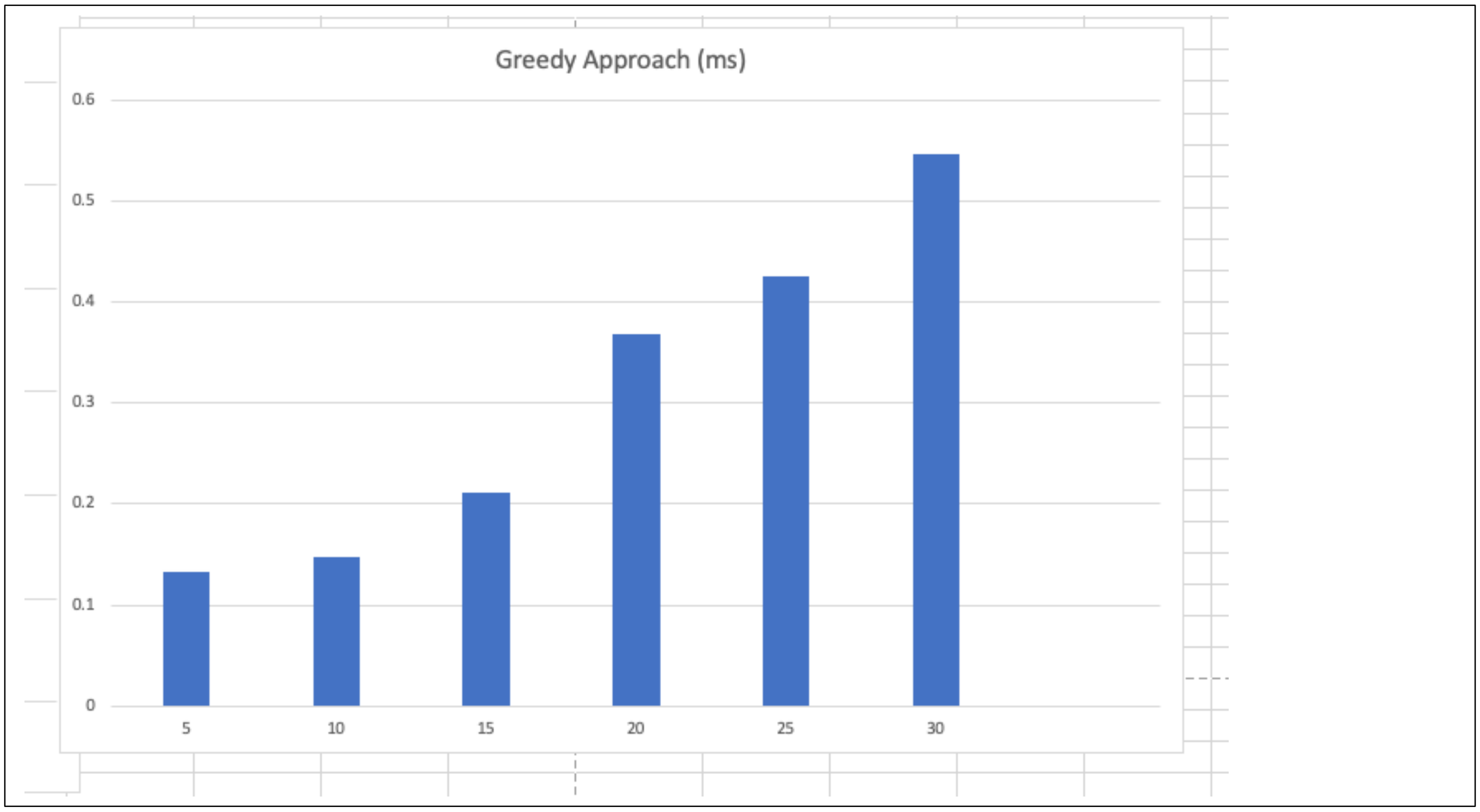
x-axis = Total number of activities (n)

y-axis = execution time in milliseconds (ms)

The time taken for 5 number of activities is very less as compare to the time taken for larger number of activities because the factor 2^n INCREASES exponentially with the increase of input size (n).

Greedy Approach Graph:

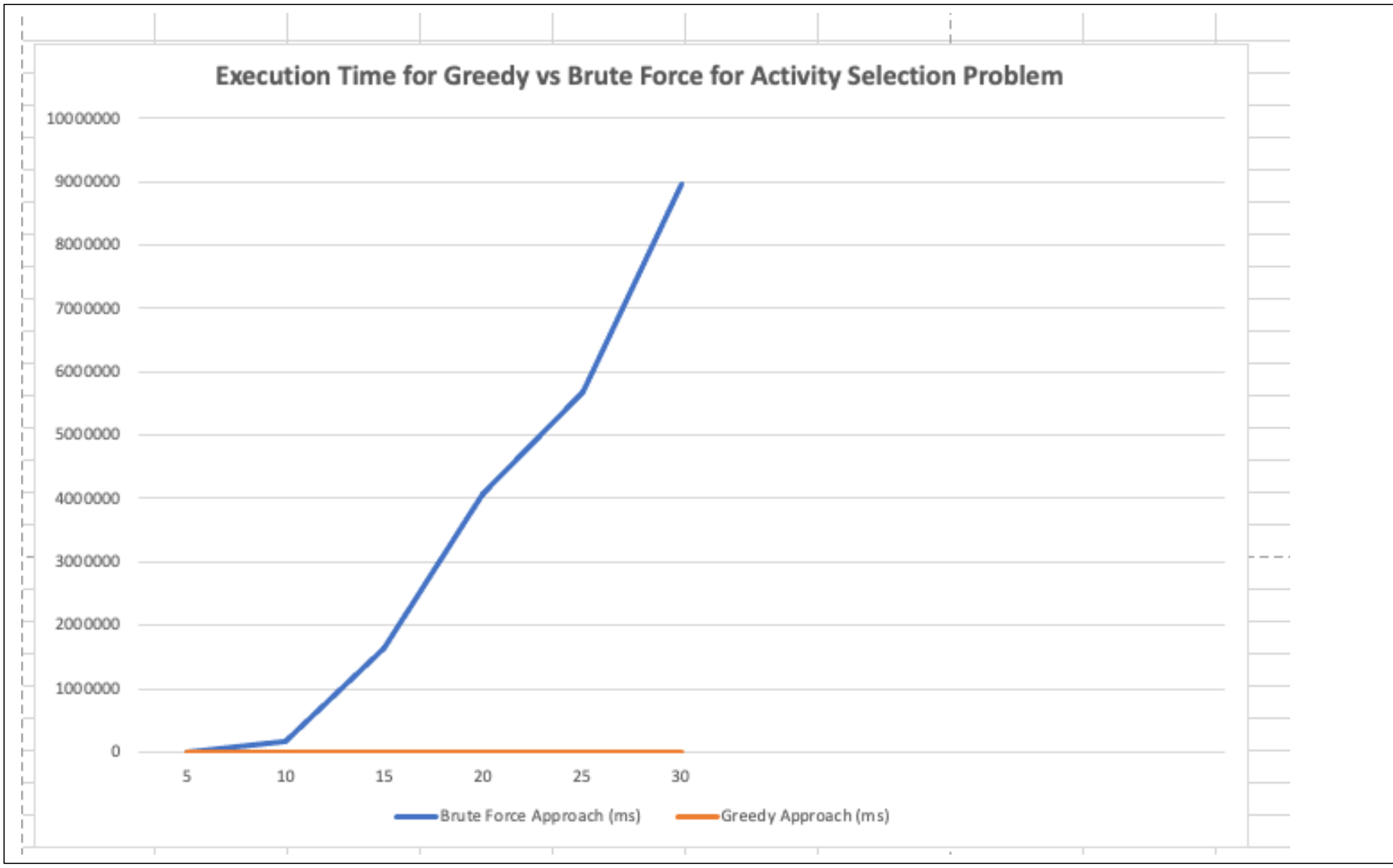
<u>Graph of Execution time (ms) vs Input Size (n)</u>



x-axis = Total number of activities (n)

y-axis = execution time in milliseconds (ms)

<u>Comparison Chart for Two Approaches</u>



x-axis = Total number of activities (n)

y-axis = execution time in milliseconds (ms)

We can see how the execution times for Brute Force changes exponentially with respect to the input size.

Time Complexity of Greedy Approach Function:

```
int maxActivitiesGreedyApproach(Activity arr[], int n)
{
   int i, j, counter = 0;  ————→ 1

   mergeSort(arr, 0, n);  ————→  n * logn
   i = 0;  ————→ 1
   for (j = 1; j < n; j++).  ————→  n+1
   {

     if (arr[j].start >= arr[i].finish)  ————→ 1
     {
        counter++;  ————→ 1
        i = j;  ————→ 1
     }
   }
   return counter+1;  ————→ 1
}
```

$$\mathbf{T(n) =} \ 1 + 1+1+1+1+1 + n + n*\log n$$

$$\mathbf{T(n) =} \ 6 + n + n*\log n$$

$$\mathbf{T(n) = O(n*\log n)}$$

Time Complexity of Brute Force Approach Function:

The time complexity calculated for the Brute Force Approach for Activity selection problem is 2^n.

$T(n) = O(2^n)$

Conclusions:

We asymptotically found that the time complexity of **Brute force is O(**2^n) whereas **for the Greedy Approach it is O(n log n)**. And we have implemented the algorithms as functions for finding the **maxActivities** for a given Array of activities of certain size. We found a significant difference between the execution times for both approached for 6 different input sizes.

For example: for input size **n=25**, the **Brute force** is taking around **5683429.76 milliseconds** while **the Divide and Conquer** is taking only 0.4251 **milliseconds**.

Hence, the Greedy Approach approach is the best to find out the maximum number of Activities because it performed very well for all input sizes.

Conclusions:

We asymptotically found that the time complexity of **Brute force is O(n^2)** whereas **for the divide and conquer it is O(n log n)**. And we have implemented the algorithms as functions for finding the **Maximum Subarray** from a given Array of certain size. We found a significant difference between the execution times for both approached for 10 different input sizes.

For example: for input size **n=9000**, the **Brute force** is taking around **99.4947 milliseconds** while **the Divide and Conquer** is taking only **0.711194 milliseconds**.

Hence, the Divide and Conquer approach is the best to find out the maximum subarray of a given size because it performed very well for all input sizes.