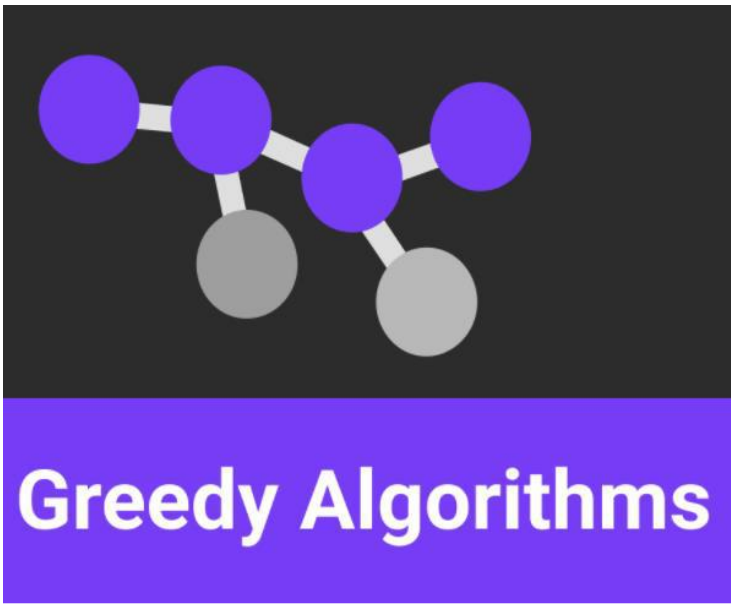


DAA-LAB7

GREEDY PROBLEM

Sr. No.	Admission No.	Team Members Names
1	U19CS011	RAJ JIKADRA
2	U19CS012	BHAGYA RANA
3	U19CS049	DEV JARIWALA
4	U19CS080	MUKTESHARYAN UPPALA

Design and Analysis of Algorithms (CS206)

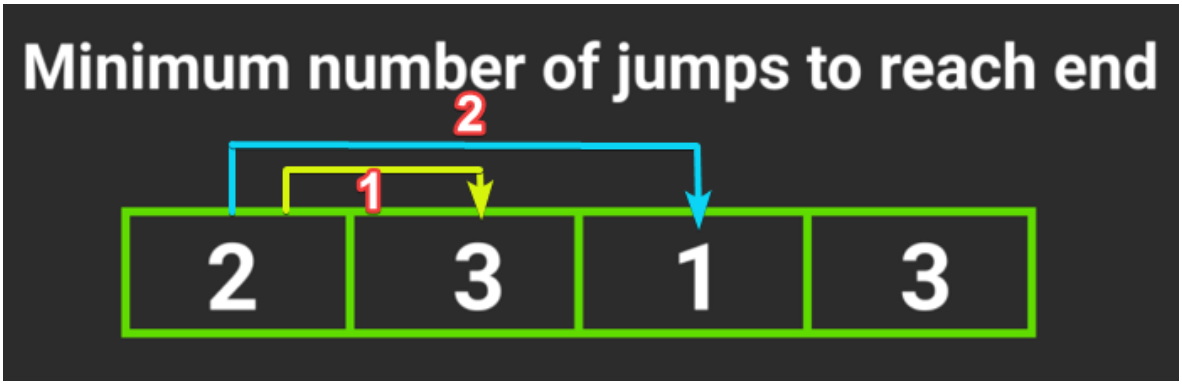
LAB Assignment – 7

1.1. (T) Problem Statement:

"Will Doraemon Help?"

As Usual, **Nobita** woke Up *Late for School* and Got Ready in Hurry. But, His **Mom**, wanted to teach Nobita a Lesson, So with Help of Smart **Suneo**, Designed a Problem to Reach School.

The **Path** from **Nobita's Home** to **School**, is Broken Down into Consecutive **Spring Jumper Tiles**, Which has Power Capacity of '**p**'. If Person stands on this Tile, than He can jump to Any Tile from 1 to 'p' ahead of current Tile. [See Image for Better Understanding]



Time Taken to Move from Current Tile to Next Tile takes **1** second.

Since, Nobita is in Hurry, He came Crying to **Doraemon** [You] to solve this Problem, So that he can reach the School as **Early as Possible**.
You need to Help Nobita, to Tell him if **He can Reach School or Not** (-1).
If He can **Reach School**, You need to inform him the **Minimum Time Taken** to Reach School.

Input Format:

Input File containing Integers, each representing the **Power** of Particular Tile

Constraints:

$$0 \leqslant A[i] \leqslant 10^6$$

Output Format:

Print "-1", If Nobita wont to be able to Reach School.
If it's Possible to Reach School, Then Print the **Minimum** Time Taken to Reach School.

Sample Input:

3 7 4 1 1 2 1 1 1

Sample Output:

2

Explanation:

The Path {3 ->7->1} will take Minimum Time

Sample Input 2:

3 2 1 1 0 7 9 1

Sample Output 2:

-1

Explanation:

No Matter What You Try You Won't to be able to Reach the End

1.2. (T) Write pseudocodes to design the algorithms for the above mentioned computational problem using the Greedy approach as well as dynamic programming.

Pseudo Code [Greedy]

```
// Greedy Method Of Solving the Problem

nums: Array of Numbers
n: Size of Array

Greedy_Jump(nums, n)

// 2 Pointers
1.  previous = 0
2.  current = 0
// No Of Jumps Taken
3.  jumps = 0

4.  for i = 0 to n - 1
5.      if (i > previous)
6.          jumps = jumps + 1
7.          previous = current
8.          if (i > current)
9.              return -1
        // Update the Current Position
10.     current = max(current, i + nums[i])

// Final Answer Containing Minimum Time
return jumps;
```

Pseudo Code [DP]

```
// DP Method of Solving the Problem

nums: Array of Numbers
n: Size of Array

DP_Jump(nums, n)

1.  jumps(n, 0)
2.  i = 0
3.  j = 0

    // Edge Cases
4.  if (n == 0 || nums[0] == 0)
5.      return -1

    // Minimum Jumps to Reach jumps[0] = 0 [You are Already Standing There!]
6.  jumps[0] = 0

    // Find the minimum number of jumps to reach nums[i] from nums[0],
    // and assign this value to jumps[i]
7.  for i = 1 to n-1
8.      jumps[i] = INT_MAX
9.      for j = 0 to i-1
10.     if (i <= j + nums[j] && jumps[j] != INT_MAX)
11.             jumps[i] = min(jumps[i], jumps[j] + 1)
                break

// If Final Answer is INT_MAX -> No Way to Reach End Otherwise Answer Exist
12. return (jumps[n - 1] == INT_MAX) ? -1 : jumps[n - 1]
```

1.3. (T) Analyze the time complexity of above algorithms.

	dp_jump(nums, n)	cost	Best Case	Worst Case
1.	Let jumps be an integer array of length n, initially 0s.	c_1	c_1	c_1
2.	if n = 0 or nums[0] = 0	c_2	c_2	c_2
3.	return -1	c_3	0	0
4.	jumps[0] = 0	c_4	c_4	c_4
5.	for i = 1 to n-1	c_5	$c_5 \cdot n$	$c_5 \cdot n$
6.	jumps[i] = ∞	c_6	$c_6(n-1)$	$c_6(n-1)$
7.	for j = 0 to i - 1	c_7	$2c_7$	$\left[\sum_{i=1}^{n-1}(i+1)\right]c_7$
8.	if (i ≤ j + nums[j] and jumps[i] = ∞)	c_8	c_8	$\left(\sum_{i=1}^{n-1} i\right)c_8$
9.	jumps[i] = min(jumps[i], j + jumps[j])	c_9	c_9	c_9
10.	break for loop	c_{10}	c_{10}	c_{10}
11.	if jumps[n-1] = ∞ return -1	c_{11}	c_{11}	c_{11}
12.	else return jumps[n-1]	c_{12}	0	0
		c_{13}	c_{13}	c_{13}
		c_{14}	c_{14}	c_{14}

* Best case complexity:

$$T(n) = c_1 + c_2 + c_4 + nc_5 + (n-1)c_6 + 2c_7 + c_8 + c_9 + c_{10} + c_{11} + c_{13} + c_{14}$$

$$= (c_5 + c_6)n + (c_1 + c_2 + c_4 - c_6 + 2c_7 + c_8 + c_9 + c_{10} + c_{11} + c_{13} + c_{14})$$

$$T(n) = an + b$$

$$\boxed{T(n) = \Theta(n)}$$

* Worst case complexity:

$$T(n) = c_1 + c_2 + c_4 + c_5 n + c_6(n-1) + \left(\sum_{i=1}^{n-1}(i+1)\right)c_7 + \left(\sum_{i=1}^{n-1} i\right)c_8 + c_9 + c_{10} + c_{11} + c_{13} + c_{14}$$

$$= (c_1 + c_2 + c_4 - c_6 + c_9 + c_{10} + c_{11} + c_{13} + c_{14}) + c_5 n + c_6 n + \frac{(n-1)(n+2)}{2} c_7 + \frac{n(n-1)}{2} c_8$$

$$= \left(\frac{c_7 + c_8}{2}\right)n^2 + \left(\frac{c_7 - c_8 + 2c_5 + 2c_6}{2}\right)n + (c_1 + c_2 + c_4 - c_6 + c_9 + c_{10} + c_{11} + c_{13} + c_{14})$$

$$= an^2 + bn + c$$

$$\boxed{T(n) = \Theta(n^2)}$$

Greedy Approach:

```
greedy_jump(nums, n)                          Cost    Best Case    Worst Case
1.  previous = 0                              C1      C1           C1
2.  current = 0                               C2      C2           C2
3.  jumps = 0                                 C3      C3           C3
4.  for i = 0 to n-1                          C4      C4(n+1)      C4(n+1)
5.      if i > previous                       C5      C5n          C5n
6.          jumps = jumps + 1                 C6      0            C6n
7.          previous = current                C7      0            C7n
8.          if i > current                    C8      0            C8n
9.              return -1                     C9      0            0
10.     current = max(current,                C10     C10n         C10·n
11.                   i + nums[i])
11. return jumps                              C11     C11          C11
```

* Best case complexity:

$$T(n) = c_1 + c_2 + c_3 + c_4(n+1) + c_5 \cdot n + c_{10} \cdot n + c_{11}$$
$$= (c_4 + c_5 + c_{10})n + (c_1 + c_2 + c_3 + c_4 + c_{11})$$
$$= an + b$$
$$\boxed{T(n) = \Theta(n)}$$

* Worst case complexity:

$$T(n) = c_1 + c_2 + c_3 + c_4(n+1) + c_5 \cdot n + c_6 \cdot n + c_7 \cdot n + c_8 \cdot n + c_{10} \cdot n + c_{11}$$
$$= (c_4 + c_5 + c_6 + c_7 + c_8 + c_{10})n + (c_1 + c_2 + c_3 + c_4 + c_{11})$$
$$= an + b$$
$$\boxed{T(n) = \Theta(n)}$$

1.4. (L) Provide the details of Hardware/Software you used to implement algorithms.

Hardware Details:

PARAMETER	LAPTOP CONFIGURATION
Operating System	Microsoft Windows **10**.0.19042
Processor	Intel(R) Core(TM) i5-10210U [Core **i5 10th Gen**]
CPU	**1.60GHz**, 2112 Mhz, **4** Core(s), 8 Logical Processor(s)
System Type	x64-based PC [**64 Bit**]
RAM	**8.00** GB
Hard Drive/SSD	512 GB **SSD**

Software Used:

PARAMETER	LAPTOP CONFIGURATION
Code Editor	**Visual Studio** Code [Version 1.52]
Compiler	gcc (MinGW.org **GCC-8.2.0-5**) 8.2.0
Time	Measured using **chrono** Library in C++
Programming Language Used	**C++**

1.5. (L) Implement the above algorithms and submit the code (complete programs).

Code:

```cpp
// HEADERS AND NAMESPACE
#include <bits/stdc++.h>
// INSTEAD OF ALL THESE
#include <iostream>
// For Creating File
#include <fstream>
#include <vector>
// For set - precision
#include <iomanip>
// For Time Calculation
#include <chrono>
// For File Name and Output File Name
#include <string>
```

```cpp
using namespace std;
using namespace std::chrono;

// COMMONLY USED TYPES
typedef long long ll;
typedef vector<ll> vll;

#define max(a, b) (a < b ? b : a)
#define min(a, b) ((a > b) ? b : a)

// Greedy Method Of Solving the Problem
ll Greedy_Jump(vll nums, ll n)
{
    int previous = 0;
    int current = 0;
    int jumps = 0;
    for (int i = 0; i < n; i++)
    {
        if (i > previous)
        {
            jumps = jumps + 1;
            previous = current;
            if (i > current)
                return -1;
        }
        current = max(current, i + nums[i]);
    }
    return jumps;
}

// DP Method of Solving the Problem
ll DP_Jump(vll nums, ll n)
{
    vll jumps(n, 0);
    int i, j;

    // Edge Cases
    if (n == 0 || nums[0] == 0)
        return -1;

    // Minimum Jumps to Reach jumps[0] = 0 [You are Already Standing There!]
    jumps[0] = 0;

    // Find the minimum number of jumps to reach nums[i] from nums[0],
    // and assign this value to jumps[i]
    for (i = 1; i < n; i++)
    {
        jumps[i] = INT_MAX;
        for (j = 0; j < i; j++)
        {
```

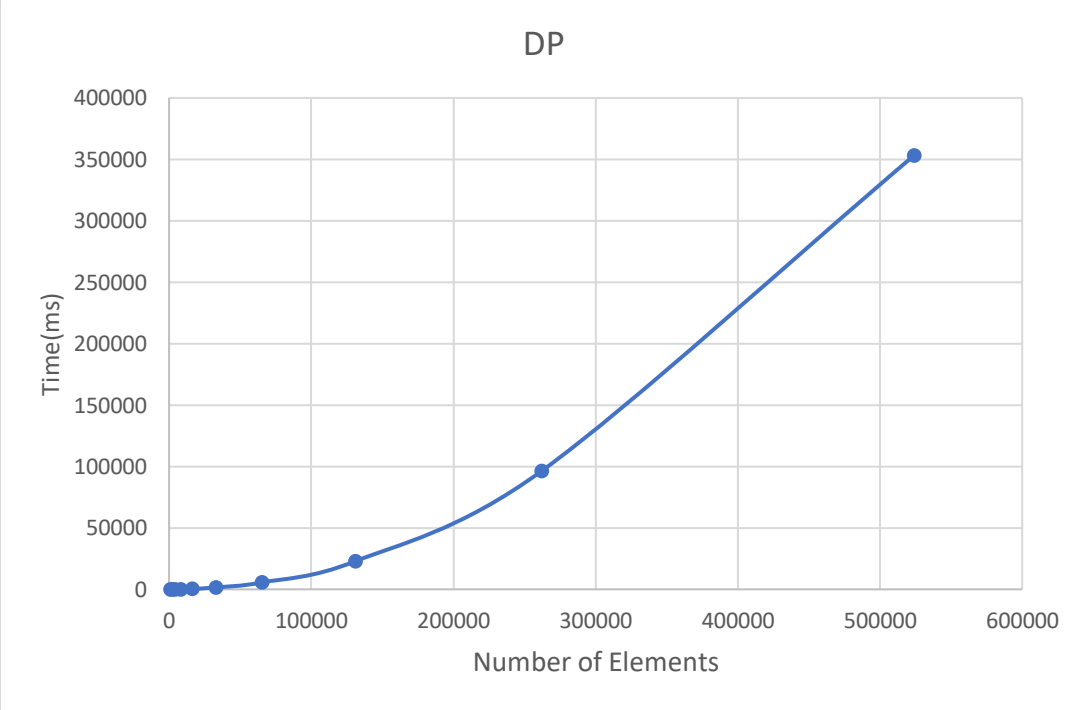
```
            if (i <= j + nums[j] && jumps[j] != INT_MAX)
            {
                jumps[i] = min(jumps[i], jumps[j] + 1);
                break;
            }
        }
    }

    // If Final Answer is INT_MAX -> No Way to Reach End Otherwise Answer Exist
    return (jumps[n - 1] == INT_MAX) ? -1 : jumps[n - 1];
}

int main()
{
    // For Read & Write from "Input File" and  Return Output to "Output" File
    freopen("output.txt", "w", stdout);

    // EDIT THIS FILE NUMBER , LIMIT and Number of Times File Runs
    int file_no = 1;
    int limit = 10;
    int each_file_runs = 2;

    for (; file_no <= limit; file_no++)
    {
        string inp_file = "File";
        string num = to_string(file_no);
        string ext = ".txt";
        inp_file += num;
        inp_file += ext;

        ifstream File;
        File.open(inp_file);

        vector<ll> arr;

        ll number;
        while (!File.eof())
        {
            File >> number;
            arr.push_back(number);
        }

        ll DP_Duration = 0;
        ll Greedy_Duration = 0;
        auto start = high_resolution_clock::now();
        auto end = high_resolution_clock::now();
        auto time_taken = duration_cast<nanoseconds>(end - start);

        ll n = arr.size();
```

```cpp
        for (int f = 0; f < each_file_runs; f++)
        {
            // DP
            start = high_resolution_clock::now();
            // Function Here
            ll dp_ans = DP_Jump(arr, n);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            DP_Duration += time_taken.count();

            // Greedy
            start = high_resolution_clock::now();
            // Function Here
            ll greedy_ans = Greedy_Jump(arr, n);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Greedy_Duration += time_taken.count();

            // For Checking Purpose
            if (dp_ans != greedy_ans)
            {
                cout << inp_file << endl;
                cout << "WRONG ANSWER!" << endl;
            }
        }

        cout << "-------------------------------------------------------" << endl;
        cout << inp_file << endl;

        cout << "TIME TAKEN (DP): ";
        double avg = (double)DP_Duration / (double)each_file_runs;
        avg *= 1e-9;
        cout << fixed << avg << setprecision(9);
        cout << " seconds" << endl;

        cout << "TIME TAKEN (GREEDY): ";
        double avg2 = (double)Greedy_Duration / (double)each_file_runs;
        avg2 *= 1e-9;
        cout << fixed << avg2 << setprecision(9);
        cout << " seconds" << endl;
    }

    return 0;
}
```

1.6. (L) Analyze the performance of both the implemented algorithms (performance of algorithms on your computers). Plot a graph.

DP METHOD

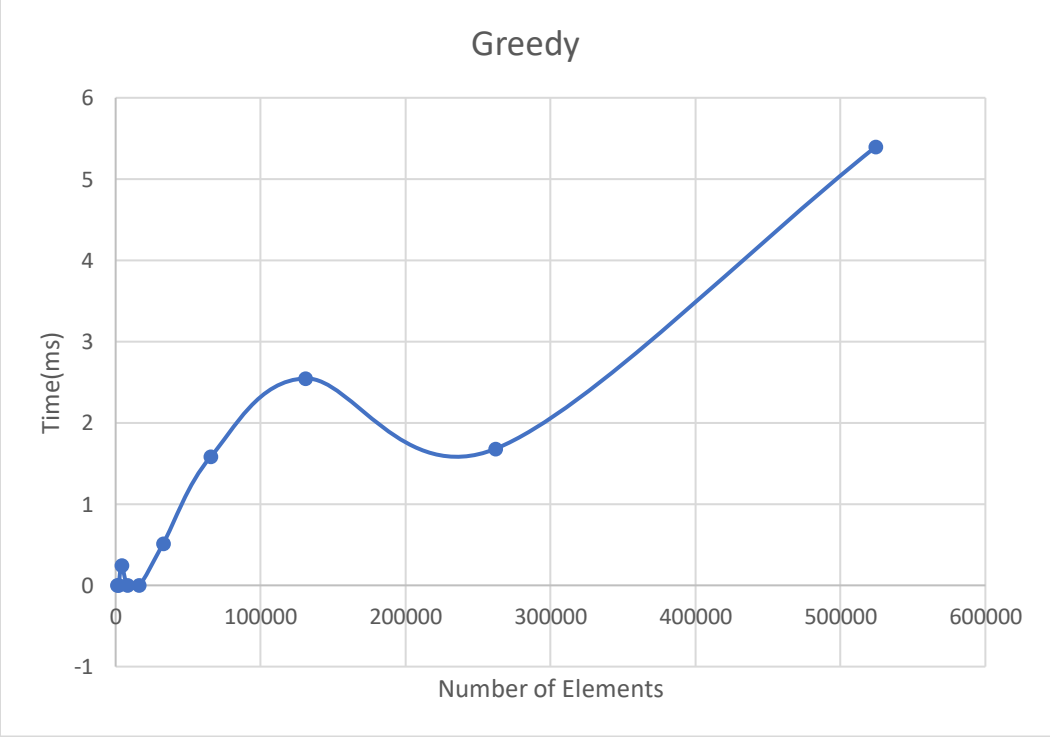
Sr. No.	No of Elements	DP (ms)
1	1024	0.00
2	2048	12.683500
3	4096	31.872000
4	8192	118.410500
5	16384	391.449000
6	32768	1561.782000
7	65536	5659.320500
8	131072	22958.085000
9	262144	96446.760500
10	524288	353511.590500



$$T(n) \approx 1.2075296762154\ldots \times 10^{-6}\, n^2 + 0.0435634n - 750.681$$

GREEDY METHOD

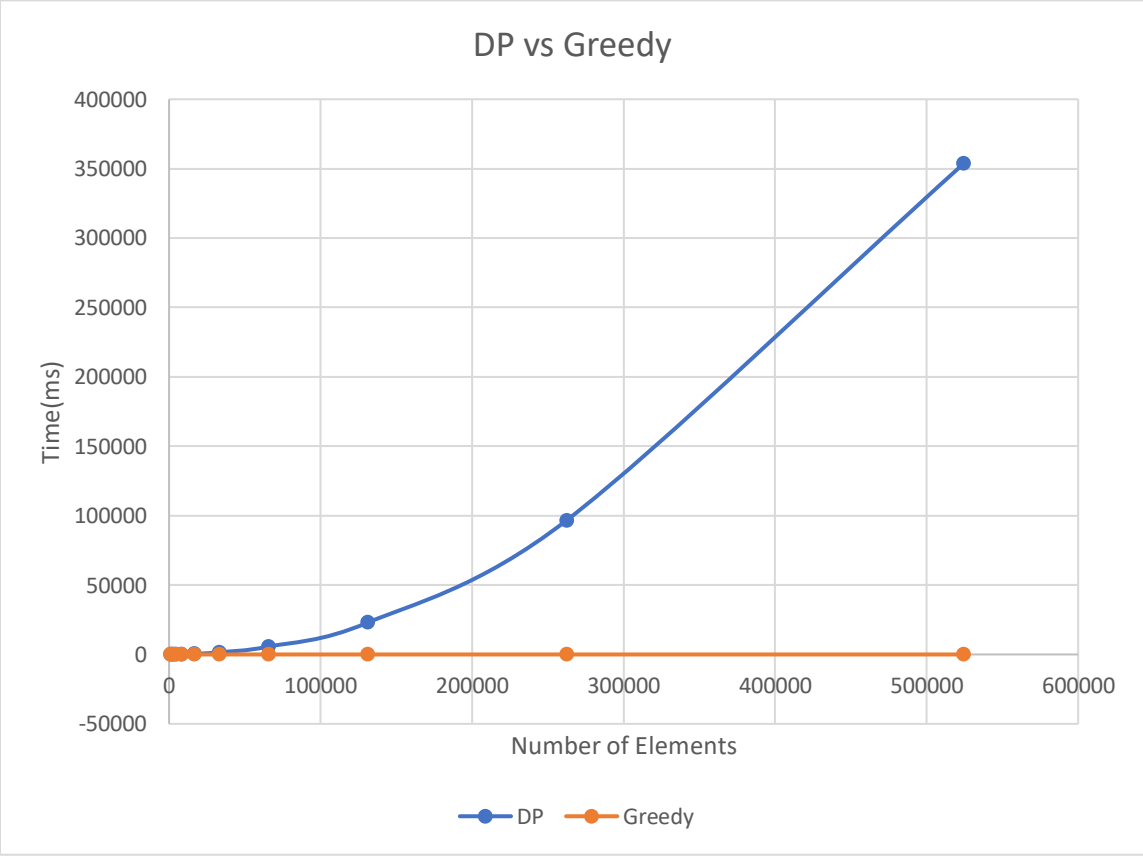
Sr. No.	No of Elements	Greedy (ms)
1	1024	0
2	2048	0
3	4096	0.2472
4	8192	0
5	16384	0
6	32768	0.513
7	65536	1.5814
8	131072	2.5466
9	262144	1.6814
10	524288	5.395835

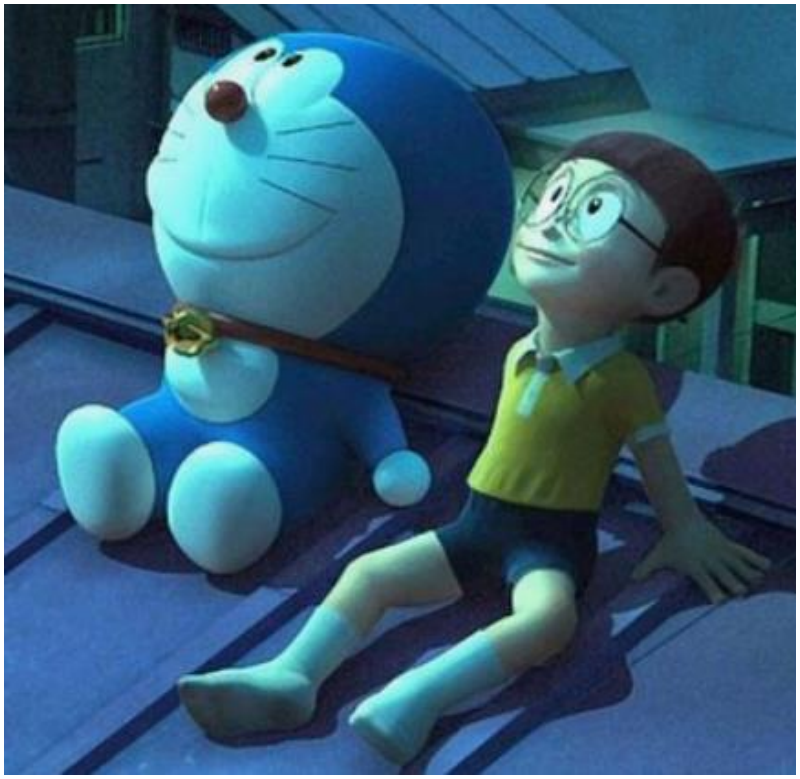


$$T(n) \approx 0.637577\,n - 18720.3$$

1.7. (L) Comparatively Analyze the performance of above algorithms and plot a graph.

Sr. No.	No of Elements	Greedy (ms)	Divide and Conquer (ms)
1	1024	0	0
2	2048	0	12.6835
3	4096	0.2472	31.872
4	8192	0	118.4105
5	16384	0	391.449
6	32768	0.513	1561.782
7	65536	1.5814	5659.3205
8	131072	2.5466	22958.085
9	262144	1.6814	96446.7605





The **Most Optimal Solution** = Nobita can ask Doraemon for **Bamboo Copter** [OR **Anywhere Door**]

SUBMITTED BY:

Sr. No.	Admission No.
1	U19CS011
2	U19CS012
3	U19CS049
4	U19CS080