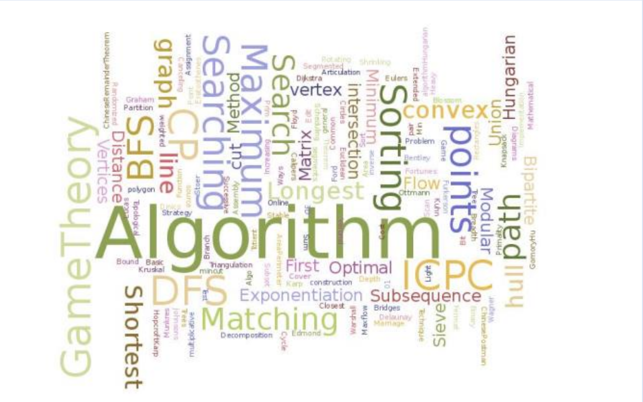
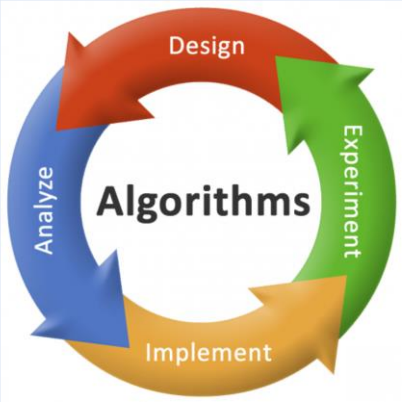
Divide and Conquer

COP 3503
Fall 2021
Department of Computer Science
University of Central Florida
Dr. Steinberg



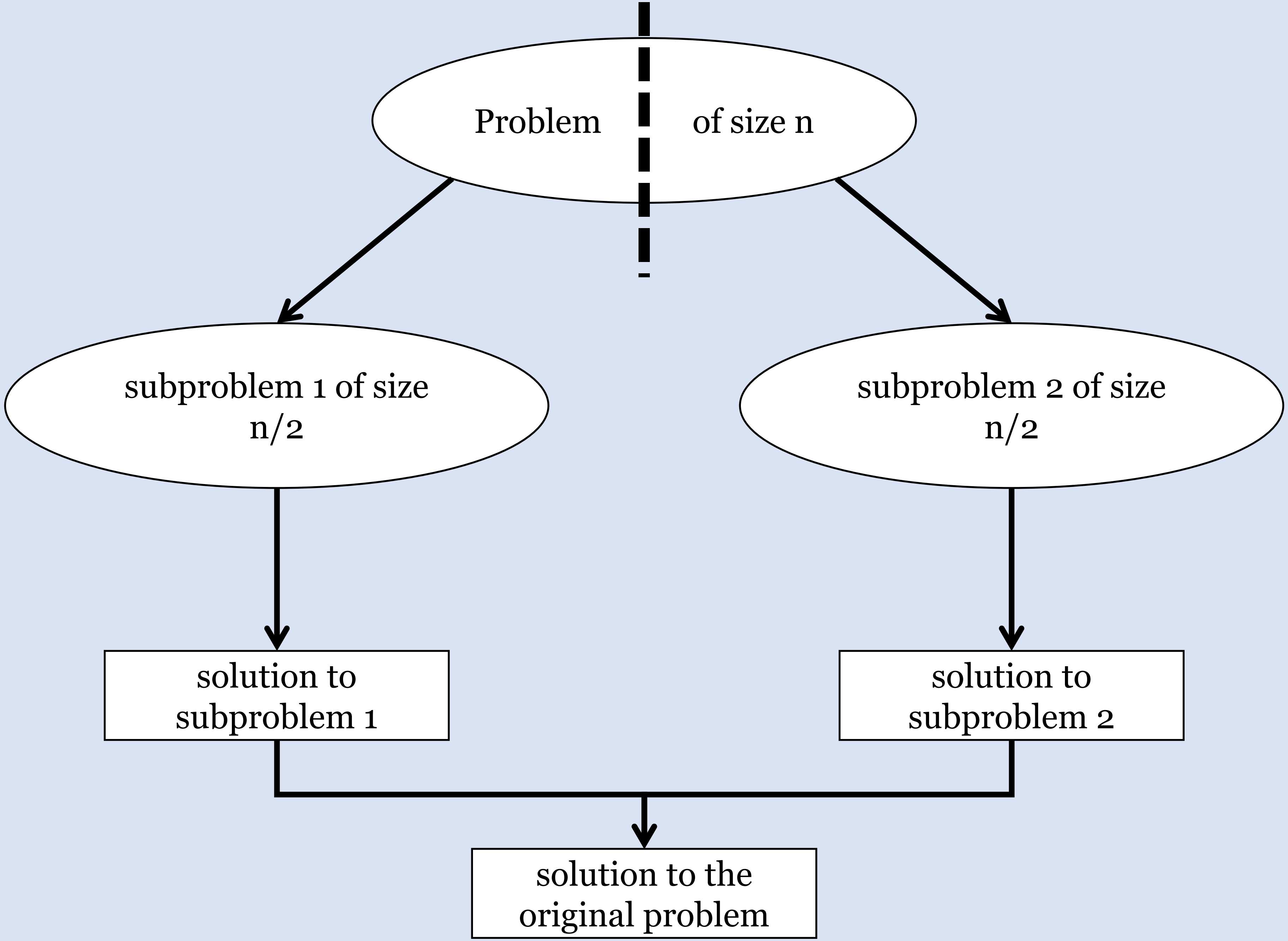


Introduction

- We just observed one of the first common approaches to designing algorithms
  - Brute Force
- While Brute Force yields correct results and is easy to implement, the algorithms have a high running time cost.
- There are multiple approaches to designing algorithms.
- Our goal in this lesson is to observe a technique that can help achieve a better running time.

Divide & Conquer

- A popular technique that used for general algorithm design
- Divide & Conquer Plan
  - A problem is divided into several subproblems of the same type, ideally about size.
  - The subproblems are solved (typically recursively, through sometimes a different algorithm is employed, especially when subproblems become small enough)
  - If necessary, the solutions to the subproblems are combined to get a solution to the original problem.
- Base Case: When the size of the problem is small enough, solve using Brute Force


Problem of size n
subproblem 1 of size n/2
subproblem 2 of size n/2
solution to subproblem 1
solution to subproblem 2
solution to the original problem

How could we represent a divide and conquer using a recurrence?

- First we observe the input size of n for the algorithm
- Then number of instances b per subproblem
- This gives us $\frac{n}{b}$ subproblems
- We also need to consider how long the dividing takes for the algorithms. We can represent as $f(n)$
- Last we need to know how many problems are needed to be solved. This can be denoted as a
- Now if we put all of this together, we are going to recognize a certain recurrence.

The General Divide-and-Conquer

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Remember The Sorting Problem Again?
Lets look at two of the solutions.
Merge-Sort and Quick-Sort
They both use Divide and Conquer

Remember Merge-Sort

<u>Merge-Sort(A,p,q,r)</u>
if p < r
    q = $\left\lfloor \frac{p+r}{2} \right\rfloor$
    Merge-Sort(A, p, q)
    Merge-Sort(A, q + 1, r)
    Merge(A, p, q, r)

<u>Merge(A,p,q,r)</u>
$n_1 = q - p + 1$
$n_2 = r - q$
//create two arrays
L[1…n_1+1] and R[1…n_2+1]

$\Theta(1)$

for i = 1 to n_1
    L[i] = A[p + i – 1]
for j = 1 to n_2
    R[j] = A[q + j]

$\Theta(n_1) = \Theta\left(\frac{n}{2}\right) = \Theta(n)$

$\Theta(n_2) = \Theta\left(\frac{n}{2}\right) = \Theta(n)$

L[n_1 + 1] = ∞
R[n_2 + 1] = ∞
i = 1
j = 1

$\Theta(1)$

for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1

$\Theta(n)$

Now why does Merge-Sort have that analysis?

- Lets represent this as a recurrence.
- $\begin{cases} \Theta(1) & n \leq c \text{ (base case)} \\ D(n) + aT(n/b) + C(n) & n > c \end{cases}$

  Divide Step Conquer Step Combine Step

- The Combine Step (Merge) has a running time $\Theta(n)$
- $\begin{cases} \Theta(1) & n = 1 \\ \Theta(1) + 2T(n/2) + \Theta(n) & n > c \end{cases}$

Master Theorem

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- $2T\left(\frac{n}{2}\right) + \Theta(n)$
- $f(n)\ vs\ n^{\log_2 2}$
- We can apply case 2 of Master Theorem
- Hence the running time analysis of Merge-Sort is $\Theta\left(n^{\log_2 2} lgn\right) = \Theta(nlgn)$

Merge-Sort Example

12	30	21	8	6	9	1	7

Remember Quick-Sort

```
Quick-Sort(A,p,r)
if p < r
    q = Partition (A, p, r)
   Quick-Sort(A, p, q – 1)
   Quick-Sort(A, q + 1, r)
```

```
Partition(A,p,q,r)
x = A[r]                          | Θ(1)
i = p – 1                         |
for j = p to r – 1                  |
   if A[j] ≤ x                      | Θ(n)
      i = i + 1                     |
      exchange A[i] and A[j]        |
exchange A[i + 1] and A[r]      | Θ(1)
return i + 1                    |
```

Worst Case for Quick-Sort

- Two subproblems are completely unbalanced:
  - One subproblem has (n – 1) elements
  - The other subproblem has 0 elements
- Ex. Input Array is already sorted in increasing order
- $T(n) = T(n-1) + T(0) + \Theta(n)$
- $T(n) = T(n-1) + cn$
- $T(n) = cn + c(n-1) + c(n-2) + \cdots + c2 + T(1)$
- $T(n) = c\big(n + (n-1) + (n-2)\big) + \cdots + c2 + T(1)$
- $T(n) = c\left(n * \frac{(n+1)}{2} - 1\right) + \Theta(1) = \Theta(n^2)$

Best Case for Quick-Sort

- The two subproblems are balanced
- This means each subproblem has n/2 elements
- $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$
- $T(n) = 2T\left(\frac{n}{2}\right) + cn$
- We can apply case 2 of Master Theorem
- $T(n) = \Theta(n \lg n)$

Now wait a minute! Aren’t we trying to design more efficient algorithms with various techniques? Why is quicksort running time in the worst case scenario quadratic? Doesn’t divide and conquer guarantee better efficiency in running time and memory?

Lets move from sorting to the fake coin problem!

The Fake Coin Problem

- The problem is described as follows:
- *You have n coins that are all supposed to be gold coins of the same weight, but you know that one coin is fake and weighs less than the others. You have a balance scale: you can put any number of coins on each side of the scale at one time, and it will tell you if the two sides weight the same, or which side is lighter if they don't weight the same.*
- How many weightings will you do?

Two Common Solutions

- Brute Force
  - Weigh each coin in sequential order until the fake one is found
- Divide and Conquer
  - Weigh two piles of coins and determine where the fake coin is
  - Discard heavier pile and divide pile into two sub piles
  - Repeat until fake coin found

C:\Windows\system32\cmd.exe

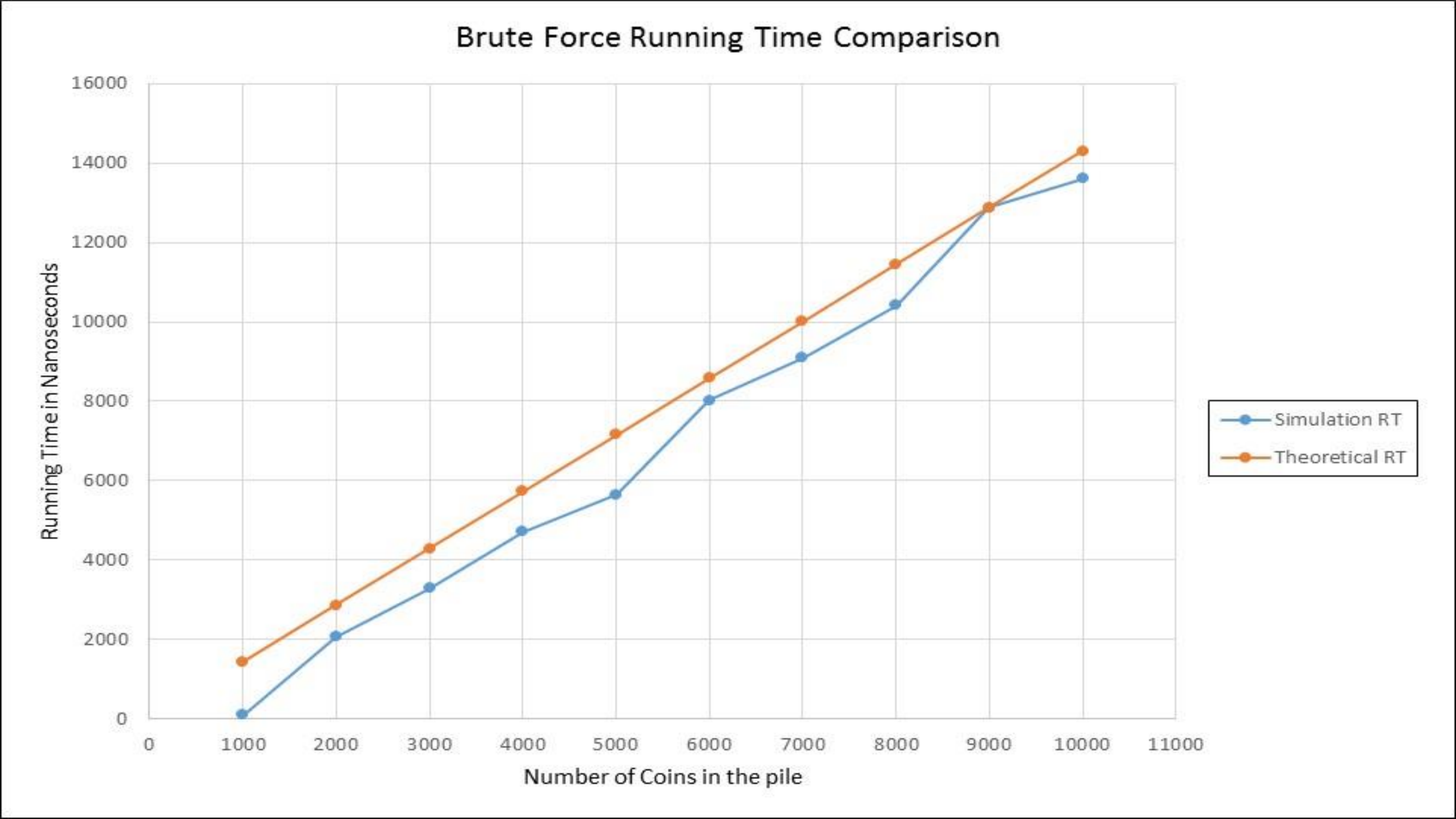
```
Welcome to the Fake Coin Program!
________________________________________________________________________________
Please select the number of coins you want to be in the problem.
NOTE: The number you select will increment by 1000 to help display the running
time.

SELECTION: 1000

Brute Force Now Executing...

Pile 1 with 1000 coins: 109 93 94 94 93
Pile 2 with 2000 coins: 2044 2059 2059 2059 2060
Pile 3 with 3000 coins: 3291 3292 3292 3276 3291
Pile 4 with 4000 coins: 4711 4727 4711 4712 4711
Pile 5 with 5000 coins: 5632 5694 5631 5616 5632
Pile 6 with 6000 coins: 8050 8034 8049 8003 8019
Pile 7 with 7000 coins: 9094 9080 9079 9095 9095
Pile 8 with 8000 coins: 10421 10436 10421 10421 10405
Pile 9 with 9000 coins: 12870 12886 12885 12870 12901
Pile 10 with 10000 coins: 13573 13728 13572 13587 13572
***************************************************************************
Divide and Conquer Now Executing...

Pile 1 with 1000 coins: 687 671 670 671 687
Pile 2 with 2000 coins: 624 639 624 640 624
Pile 3 with 3000 coins: 827 827 826 812 826
Pile 4 with 4000 coins: 780 780 780 780 780
Pile 5 with 5000 coins: 796 780 796 780 795
Pile 6 with 6000 coins: 733 718 733 718 733
Pile 7 with 7000 coins: 874 873 889 874 874
Pile 8 with 8000 coins: 733 749 733 733 749
Pile 9 with 9000 coins: 842 843 842 842 843
Pile 10 with 10000 coins: 842 858 843 842 843
***************************************************************************
Average Running Time for the Brute Force Algorithm (in nanoseconds).
96.6  2056.2  3288.4  4714.4  5641  8031  9088.6  10420.8  12882.4  13606.4

Average Running Time for the Divde & Conquer Algorithm (in nanoseconds).
677.2  630.2  823.6  780  789.4  727  876.8  739.4  842.4  845.6

Number of weights it took to reach the fake coin.
Pile 1 of Coins: Brute Force- 21 weights  Divide and Conquer- 16 weights
Pile 2 of Coins: Brute Force- 734 weights  Divide and Conquer- 15 weights
Pile 3 of Coins: Brute Force- 1168 weights  Divide and Conquer- 19 weights
Pile 4 of Coins: Brute Force- 1751 weights  Divide and Conquer- 18 weights
Pile 5 of Coins: Brute Force- 2085 weights  Divide and Conquer- 18 weights
Pile 6 of Coins: Brute Force- 2863 weights  Divide and Conquer- 17 weights
Pile 7 of Coins: Brute Force- 3240 weights  Divide and Conquer- 20 weights
Pile 8 of Coins: Brute Force- 3680 weights  Divide and Conquer- 17 weights
Pile 9 of Coins: Brute Force- 4482 weights  Divide and Conquer- 19 weights
Pile 10 of Coins: Brute Force- 4733 weights  Divide and Conquer- 19 weights
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Would you like to run the problem again?
0: No
1: Yes
Selection:
```

Brute Force

n	Simulation RT	Theoretical RT	Hidden Constant
1000	96.6	1000	0.09660
2000	2056.2	2000	1.02810
3000	3288.4	3000	1.09613
4000	4714.4	4000	1.17860
5000	5641	5000	1.12820
6000	8031	6000	1.33850
7000	9088.6	7000	1.29837
8000	10420.8	8000	1.30260
9000	12882.4	9000	1.43138
10000	13606.4	10000	1.36064
c_{max}		1.43138	

Brute Force

n	Simulation RT	Theoretical RT	% Difference
1000	96.6	1431.4	93.25
2000	2056.2	2862.8	28.17
3000	3288.4	4294.1	23.42
4000	4714.4	5725.5	17.66
5000	5641	7156.9	21.18
6000	8031	8588.3	6.49
7000	9088.6	10019.6	9.29
8000	10420.8	11451.0	9.00
9000	12882.4	12882.4	0.00
10000	13606.4	14313.8	4.94
c_{max}		1.43138	
Average % Difference		21.34	



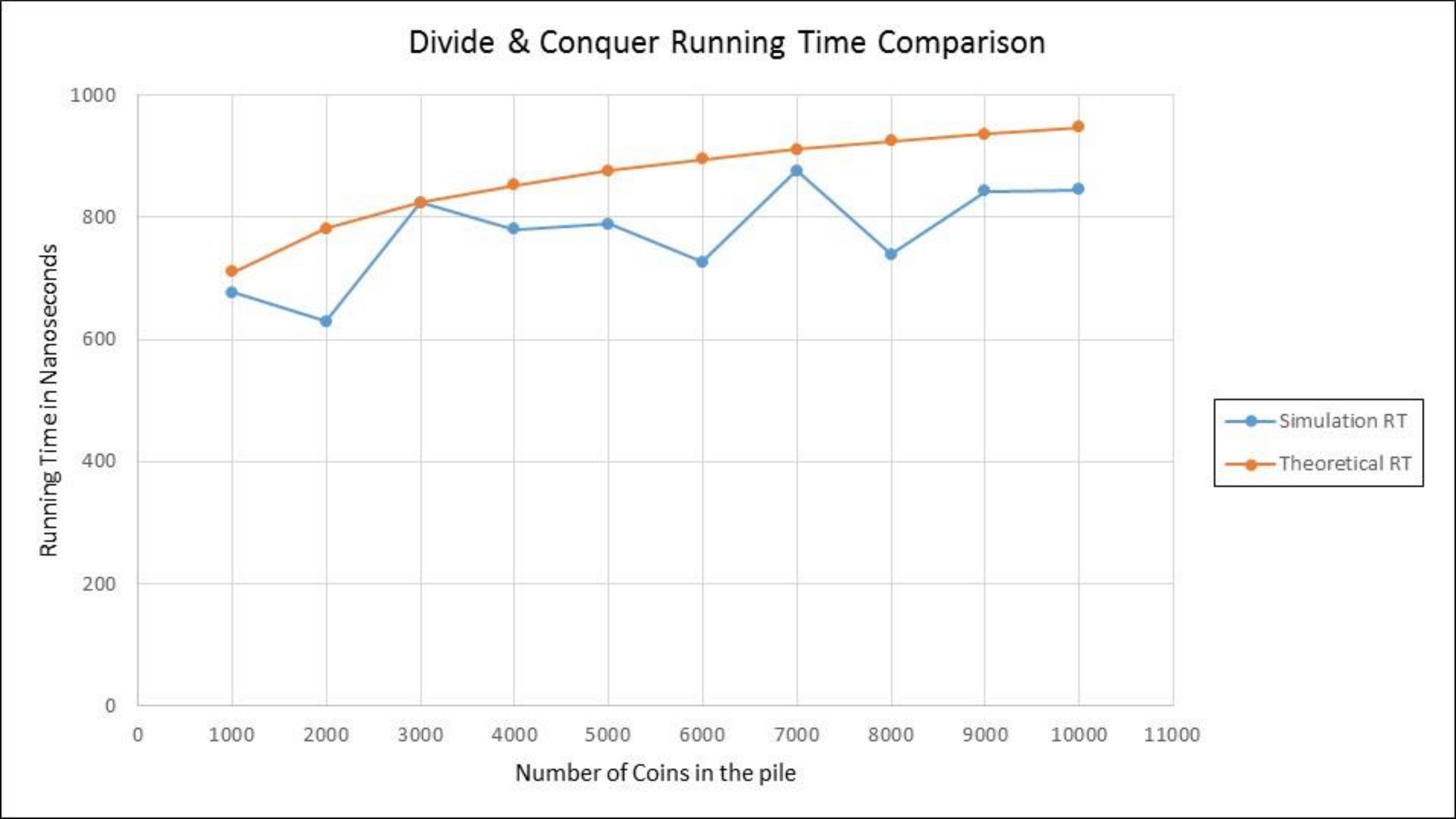
Brute Force Running Time Comparison
Running Time in Nanoseconds
16000
14000
12000
10000
8000
6000
4000
2000
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
Number of Coins in the pile
Simulation RT
Theoretical RT

Divide & Conquer

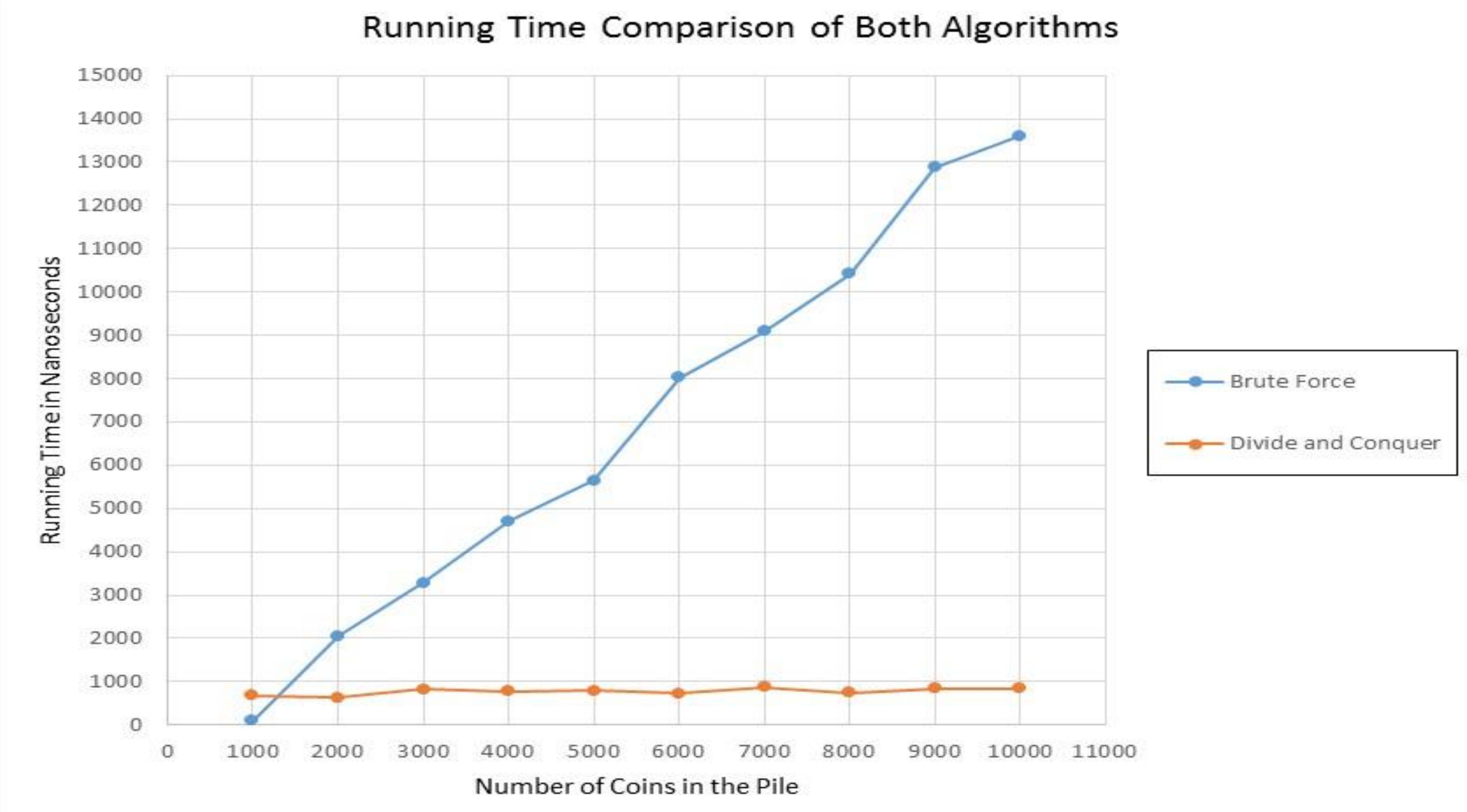
n	Simulation RT	Theoretical RT	Hidden Constant
1000	677.2	996.6	0.67953
2000	630.2	1096.6	0.57470
3000	823.6	1155.1	0.71303
4000	780	1196.6	0.65186
5000	789.4	1228.8	0.64243
6000	727	1255.1	0.57925
7000	876.8	1277.3	0.68644
8000	739.4	1296.6	0.57027
9000	842.4	1313.6	0.64131
10000	845.6	1328.8	0.63638
c_{max}		0.71303	

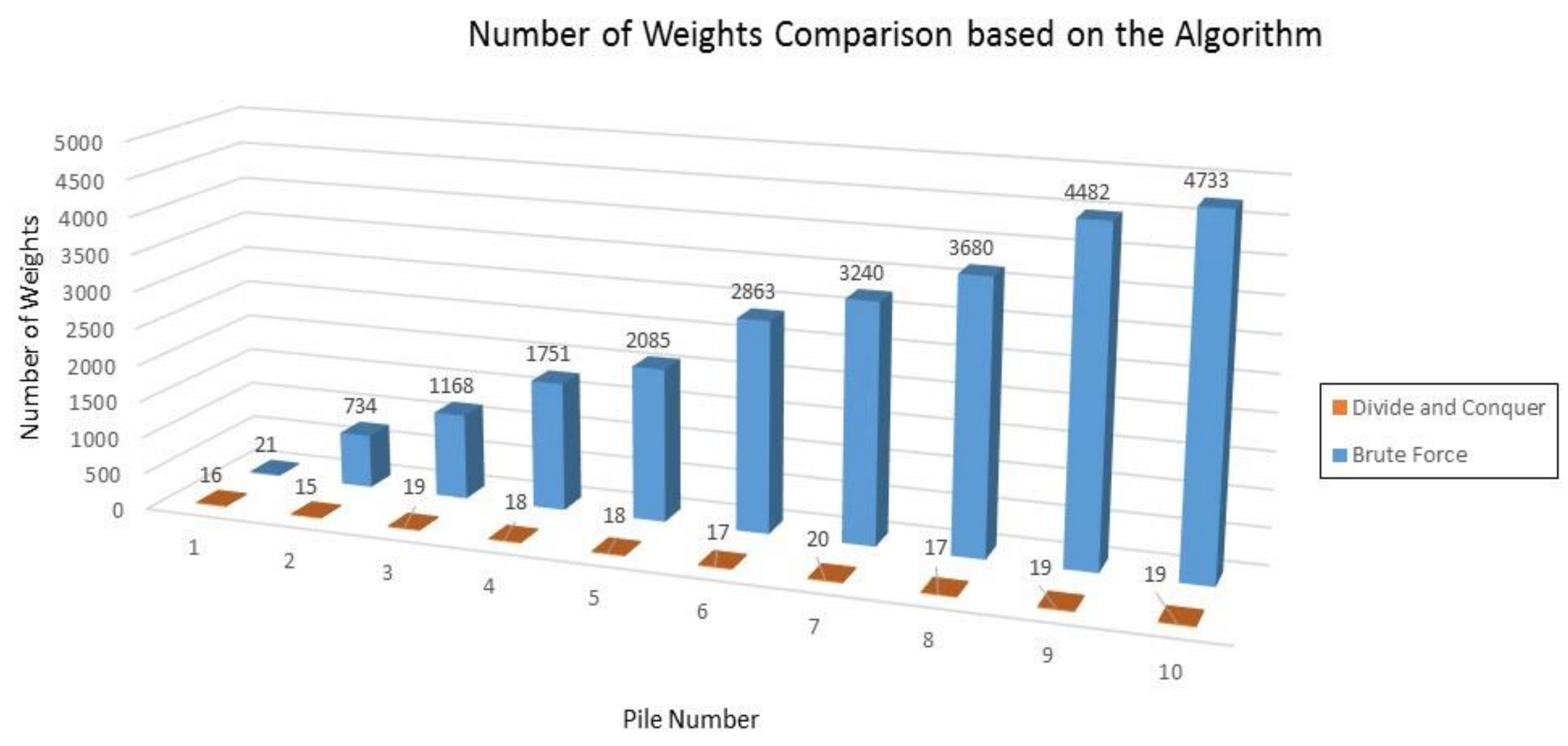
Divide & Conquer

n	Simulation RT	Theoretical RT	% Difference
1000	677.2	710.6	4.70
2000	630.2	781.9	19.40
3000	823.6	823.6	0.00
4000	780	853.2	8.58
5000	789.4	876.1	9.90
6000	727	894.9	18.76
7000	876.8	910.8	3.73
8000	739.4	924.5	20.02
9000	842.4	936.6	10.06
10000	845.6	947.5	10.75
c_{max}		0.71303	
Average % Difference		10.59	



Divide & Conquer Running Time Comparison
Running Time in Nanoseconds
1000
800
600
400
200
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
Number of Coins in the pile
Simulation RT
Theoretical RT


Running Time Comparison of Both Algorithms
Running Time in Nanoseconds
15000
14000
13000
12000
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
Number of Coins in the Pile
Brute Force
Divide and Conquer


Number of Weights Comparison based on the Algorithm
Number of Weights
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
16
21
15
734
19
1168
18
1751
18
2085
17
2863
20
3240
17
3680
19
4482
19
4733
1
2
3
4
5
6
7
8
9
10
Pile Number
Divide and Conquer
Brute Force

Binary Search

- Given a sorted array A of n numbers, determine whether a given number x belongs to the array.
- We want to search an array A[p…r] and check to see if a certain element exists.
- Divide the array into two halves.
  - $q = \left\lfloor \frac{p+r}{2} \right\rfloor$
- Compare the middle element with the value x. If x smaller than middle element, then we discard the right half of the array and look at the left half. If not, we look at the right half. Repeat the same step until the size of array is 1 element.
  - If x < A[q], then search x into A[p…q-1]
  - If x > A[q], then search x into A[q+1…r]

Binary Search Algorithm

<u>BinarySearch(A,p,r,x)</u>

```
if p == r
    if x==A[p] then return p
    else return -1 //not found
```

$q = \left\lfloor \frac{p+r}{2} \right\rfloor$

```
if x == A[q] then return q
else if x < A[q]
    BinarySearch(A, p, q – 1, x)
else
    BinarySerach(A, q+1, r, x)
```

Example of Applying Binary Search for element 13

2	7	13	32	71	77	80	84	99	101

Binary Search Running Time Analysis

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- Combine Step: Constant $\Theta(1)$
- 1 Problem (a = 1)
- Subproblems created (b = 2)
- $T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$
- We can apply Case 2 of Master Theorem
- This results in the running time analysis to be $\Theta(log\, n)$
- Significant Improvement compared to a linear search!

Growth of Functions for Search