



FDS 2025 春夏  
project1  
Performance Measurement (A+B)

徐天杰

2025-03-21

# Contents

|          |                                  |           |
|----------|----------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>              | <b>1</b>  |
| <b>2</b> | <b>Algorithm Specification</b>   | <b>1</b>  |
| 2.1      | Brute-force Search . . . . .     | 1         |
| 2.2      | Divide and conquer . . . . .     | 1         |
| 2.3      | Array and sum generate . . . . . | 2         |
| <b>3</b> | <b>Testing Data</b>              | <b>3</b>  |
| 3.1      | When $V = 1000$ . . . . .        | 3         |
| 3.2      | When $V = 5000$ . . . . .        | 4         |
| 3.3      | When $V = 10000$ . . . . .       | 5         |
| 3.4      | When $V = 20000$ . . . . .       | 6         |
| 3.5      | When $V = 40000$ . . . . .       | 8         |
| 3.6      | When $V = 60000$ . . . . .       | 9         |
| 3.7      | When $V = 80000$ . . . . .       | 10        |
| 3.8      | When $V = 100000$ . . . . .      | 11        |
| <b>4</b> | <b>Analysis and Comments</b>     | <b>12</b> |
| 4.1      | Brute-force search . . . . .     | 12        |
| 4.2      | Divide and conquer . . . . .     | 12        |
| 4.3      | Comprehensive analysis . . . . . | 12        |
| <b>5</b> | <b>Appendix</b>                  | <b>13</b> |
| <b>6</b> | <b>Declaration</b>               | <b>17</b> |

# 1 Introduction

In this project, we need to implement at least two algorithms to find  $a + b = c$  in a given collection of  $N$  integers. And we need to further count the consumption of time of the two algorithms as the size of  $N$  grows , so that we can prove the time complexity of the two algorithms.

- (1). In Chapter 2, I will introduce the implementation of the two algorithms using pseudocode and explanations.
- (2). In Chapter 3, I present my test data , with table and graphs of test cases .
- (3). In Chapter 4, I conduct an analysis of my program, including time complexity and space complexity.
- (4). In Chapter 5, I include the complete code in the appendix.

# 2 Algorithm Specification

The following are the pseudocodes and explanations for these search algorithms.

## 2.1 Brute-force Search

```
1  brute_force_search(arr, sum, n):  
2      res = 0 //initialize result  
3      for i from 0 to n-1: //traversal the first element  
4          for j from i+1 to n-1: //traversal the second element  
5              if arr[i] + arr[j] == sum: //check the sum  
6                  res = 1  
7                  return res //target found  
8      return res //target not found
```

I deploy Brute-force search as the first function that it searches every integer group in the array to check if there's any group can let  $a+b=c$ .

## 2.2 Divide and conquer

```
1  div_con(arr, sum, left_p, right_p, count):  
2      res = 0 //initialize result  
3
```

```

4 //only begin search if the size is smaller than 20
5 if (right_p - left_p) > 20:
6     mid = (right_p + left_p) / 2 //calculate the mid num
7     res_left = div_con(arr, sum, left_p, mid, count) //recursiononly
8         check the left part
9     if res_left == 1: //if find in the left half
10        res = 1
11    return res //target found
12
13    res_right = div_con(arr, sum, mid, right_p, count) //recursiononly
14        check the right part
15    if res_right == 1: //if find in the left half
16        res = 1
17    return res //target found
18
19 //if the size if smaller than 20 or the smaller branch didn't find
20 //the result
21 for i from left_p to right_p - 1:
22     for j from i + 1 to right_p - 1:
23         if arr[i] + arr[j] == sum: //check the sum
24             res = 1
25             // printf("%d %d + %d = %d\n", count, arr[i], arr[j], sum)
26         return res //target found
27
28 return res //target not found

```

The second function uses divide and conquer,I recursively divide the array into small arrays.And deploy brute-force search in the small arrays.

Note that I choose to **check the left part first**,and if the target is found immediately return 1 to avoid scanning all the subsets and find more than 1 number group,which well cost about 10 or more times of cost time.

## 2.3 Array and sum generate

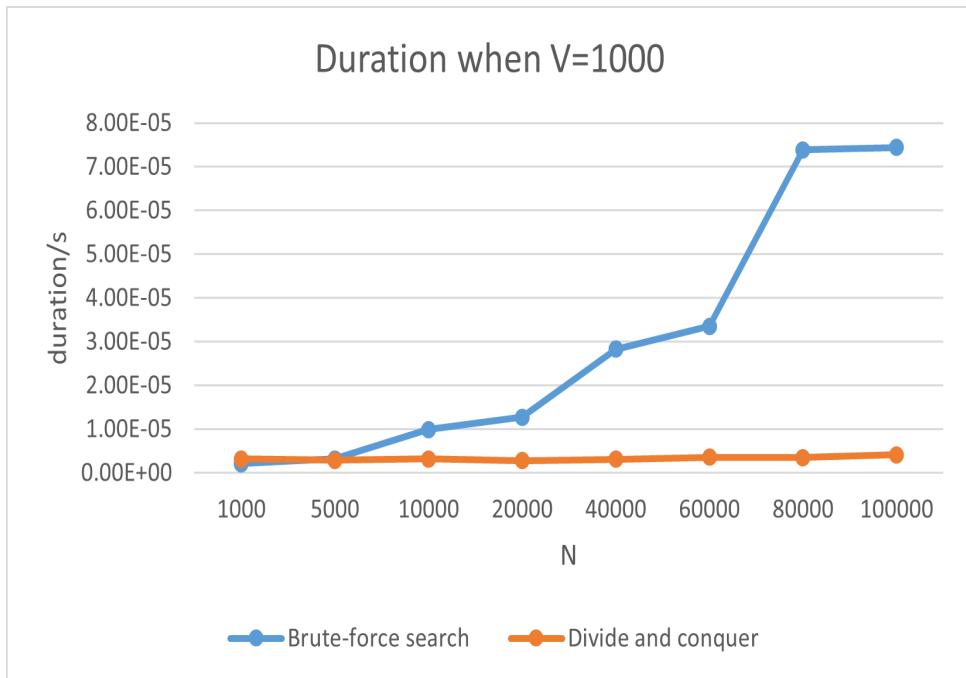
I'd like to briefly explain array and sum generate process here.This process is automatically generated in program and it is totally random.Also,to make sure program can find a answer I choose to randomly select 2 integers from generated array to calculate the target sum

### 3 Testing Data

To analyze the performance of the two functions, we conducted tests using three different values of  $V$ , which represents the maximum number in the integer array. For each value of  $V$ , we tested **eight different array sizes** to compare the running times of the Brute-force search and Divide-and-Conquer algorithms. As expected, the **Brute-force search performs better when the array size  $n$  is small**, regardless of the value of  $V$ . However, as  $n$  increases, **the Divide-and-Conquer algorithm becomes more efficient**. Below are the results collected from our program.

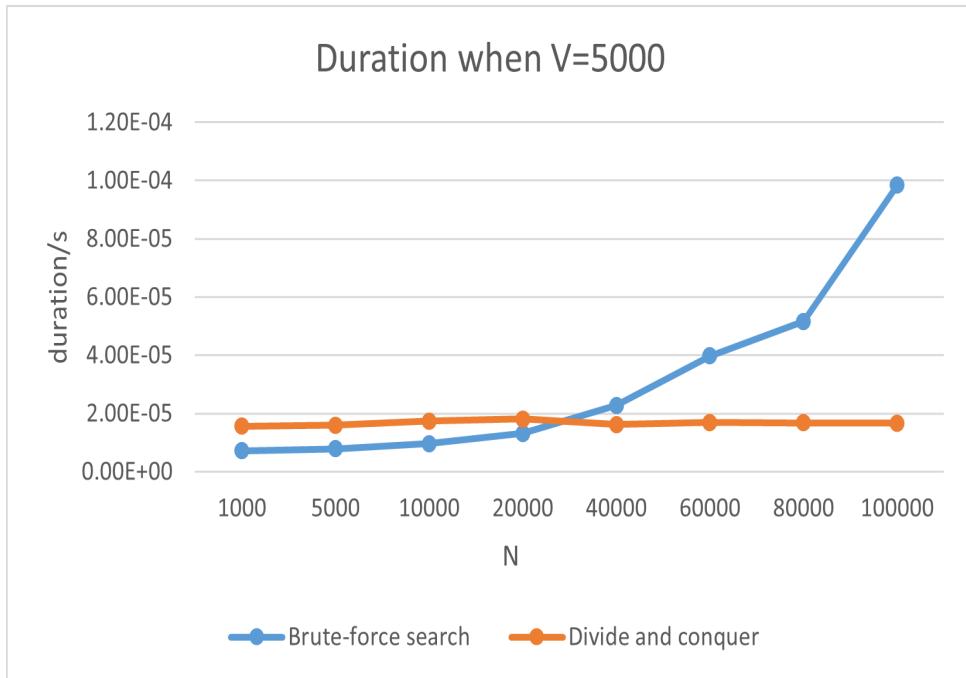
#### 3.1 When $V = 1000$

| $V=1000$           | $N$             | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 21       | 32       | 99       | 127      | 283      | 335      | 739      | 744      |
|                    | Total Time(sec) | 0.021    | 0.032    | 0.099    | 0.127    | 0.283    | 0.335    | 0.739    | 0.744    |
|                    | Duration(sec)   | 2.10E-06 | 3.20E-06 | 9.90E-06 | 1.27E-05 | 2.83E-05 | 3.35E-05 | 7.39E-05 | 7.44E-05 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 32       | 29       | 32       | 28       | 31       | 36       | 35       | 41       |
|                    | Total Time(sec) | 0.032    | 0.029    | 0.032    | 0.028    | 0.031    | 0.036    | 0.035    | 0.041    |
|                    | Duration(sec)   | 3.20E-06 | 2.90E-06 | 3.20E-06 | 2.80E-06 | 3.10E-06 | 3.60E-06 | 3.50E-06 | 4.10E-06 |

Figure 1: Running times for  $V = 1000$ Figure 2: Plotted running times for  $V = 1000$

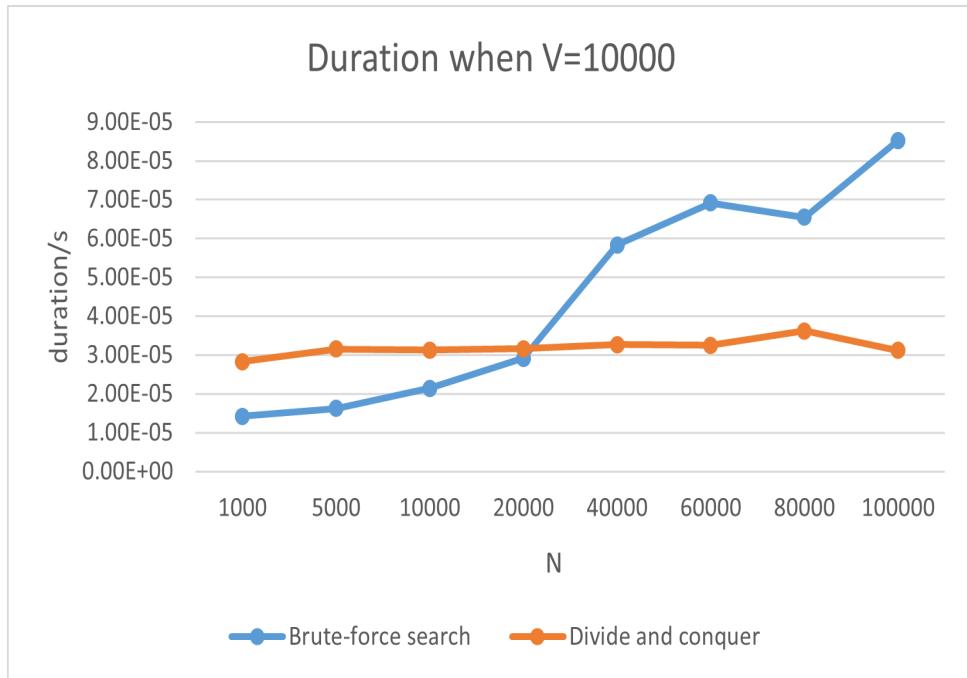
### 3.2 When $V = 5000$

| $V=5000$           | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 72       | 79       | 96       | 131      | 228      | 398      | 516      | 985      |
|                    | Total Time(sec) | 0.072    | 0.079    | 0.096    | 0.131    | 0.228    | 0.398    | 0.516    | 0.985    |
|                    | Duration(sec)   | 7.20E-06 | 7.90E-06 | 9.60E-06 | 1.31E-05 | 2.28E-05 | 3.98E-05 | 5.16E-05 | 9.85E-05 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 157      | 160      | 174      | 181      | 162      | 170      | 168      | 167      |
|                    | Total Time(sec) | 0.157    | 0.16     | 0.174    | 0.181    | 0.162    | 0.17     | 0.168    | 0.167    |
|                    | Duration(sec)   | 1.57E-05 | 1.60E-05 | 1.74E-05 | 1.81E-05 | 1.62E-05 | 1.70E-05 | 1.68E-05 | 1.67E-05 |

Figure 3: Running times for  $V = 5000$ Figure 4: Plotted running times for  $V = 5000$

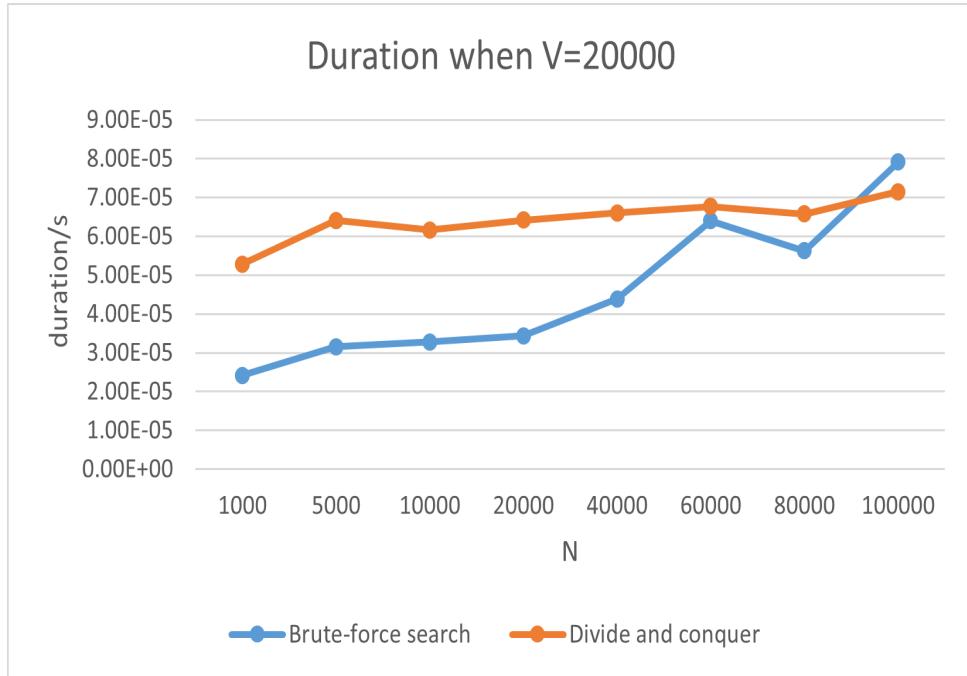
### 3.3 When $V = 10000$

| V=10000            | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 142      | 162      | 214      | 291      | 584      | 692      | 655      | 853      |
|                    | Total Time(sec) | 0.142    | 0.162    | 0.214    | 0.291    | 0.584    | 0.692    | 0.655    | 0.853    |
|                    | Duration(sec)   | 1.42E-05 | 1.62E-05 | 2.14E-05 | 2.91E-05 | 5.84E-05 | 6.92E-05 | 6.55E-05 | 8.53E-05 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 283      | 315      | 313      | 316      | 327      | 325      | 362      | 312      |
|                    | Total Time(sec) | 0.283    | 0.315    | 0.313    | 0.316    | 0.327    | 0.325    | 0.362    | 0.312    |
|                    | Duration(sec)   | 2.83E-05 | 3.15E-05 | 3.13E-05 | 3.16E-05 | 3.27E-05 | 3.25E-05 | 3.62E-05 | 3.12E-05 |

Figure 5: Running times for  $V = 10000$ Figure 6: Plotted running times for  $V = 10000$

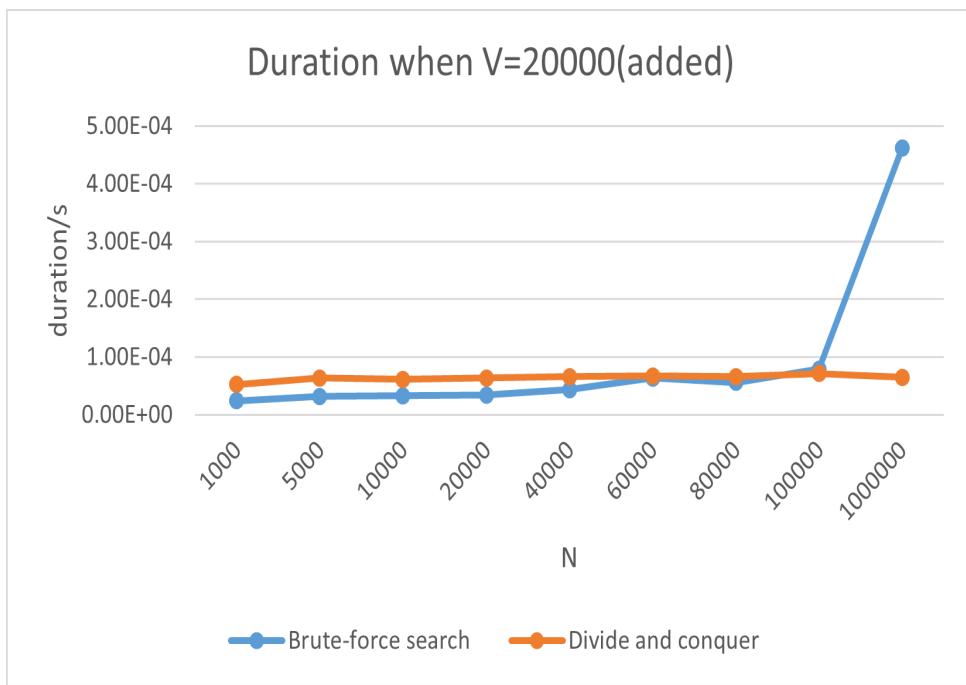
### 3.4 When $V = 20000$

| $V=20000$          | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 242      | 315      | 328      | 344      | 439      | 640      | 563      | 792      |
|                    | Total Time(sec) | 0.242    | 0.315    | 0.328    | 0.344    | 0.439    | 0.64     | 0.563    | 0.792    |
|                    | Duration(sec)   | 2.42E-05 | 3.15E-05 | 3.28E-05 | 3.44E-05 | 4.39E-05 | 6.40E-05 | 5.63E-05 | 7.92E-05 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 529      | 641      | 617      | 642      | 660      | 677      | 658      | 715      |
|                    | Total Time(sec) | 0.529    | 0.641    | 0.617    | 0.642    | 0.66     | 0.677    | 0.658    | 0.715    |
|                    | Duration(sec)   | 5.29E-05 | 6.41E-05 | 6.17E-05 | 6.42E-05 | 6.60E-05 | 6.77E-05 | 6.58E-05 | 7.15E-05 |

Figure 7: Running times for  $V = 20000$ Figure 8: Plotted running times for  $V = 20000$ 

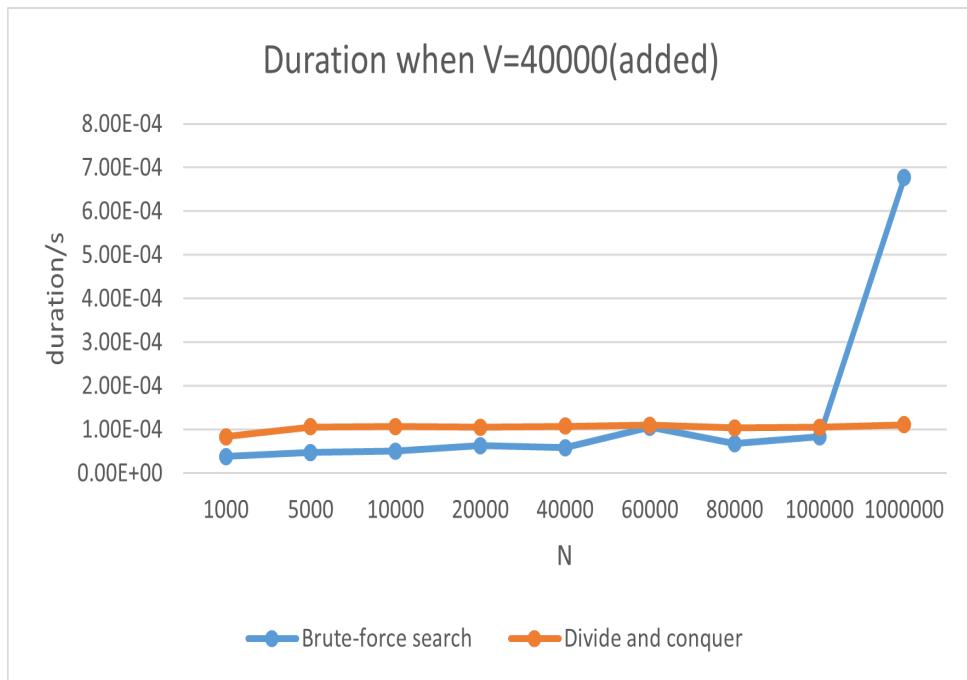
As the three figures shows that no matter what the  $V$  is we can find Brute-force search performs better at the beginning and Divide and conquer performs better as  $n$  increases . But I also notice when  $V = 20000$  even if  $n = 20000$  , Divide and conquer uses more time . So I add one test group in the last table as below:

| $V=20000$          | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 242      | 315      | 328      | 344      | 439      | 640      | 563      | 792      |
|                    | Total Time(sec) | 0.242    | 0.315    | 0.328    | 0.344    | 0.439    | 0.64     | 0.563    | 0.792    |
|                    | Duration(sec)   | 2.42E-05 | 3.15E-05 | 3.28E-05 | 3.44E-05 | 4.39E-05 | 6.40E-05 | 5.63E-05 | 7.92E-05 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 529      | 641      | 617      | 642      | 660      | 677      | 658      | 715      |
|                    | Total Time(sec) | 0.529    | 0.641    | 0.617    | 0.642    | 0.66     | 0.677    | 0.658    | 0.715    |
|                    | Duration(sec)   | 5.29E-05 | 6.41E-05 | 6.17E-05 | 6.42E-05 | 6.60E-05 | 6.77E-05 | 6.58E-05 | 7.15E-05 |

Figure 9: Running times for  $V = 20000$ (data added)Figure 10: Plotted running times for  $V = 20000$

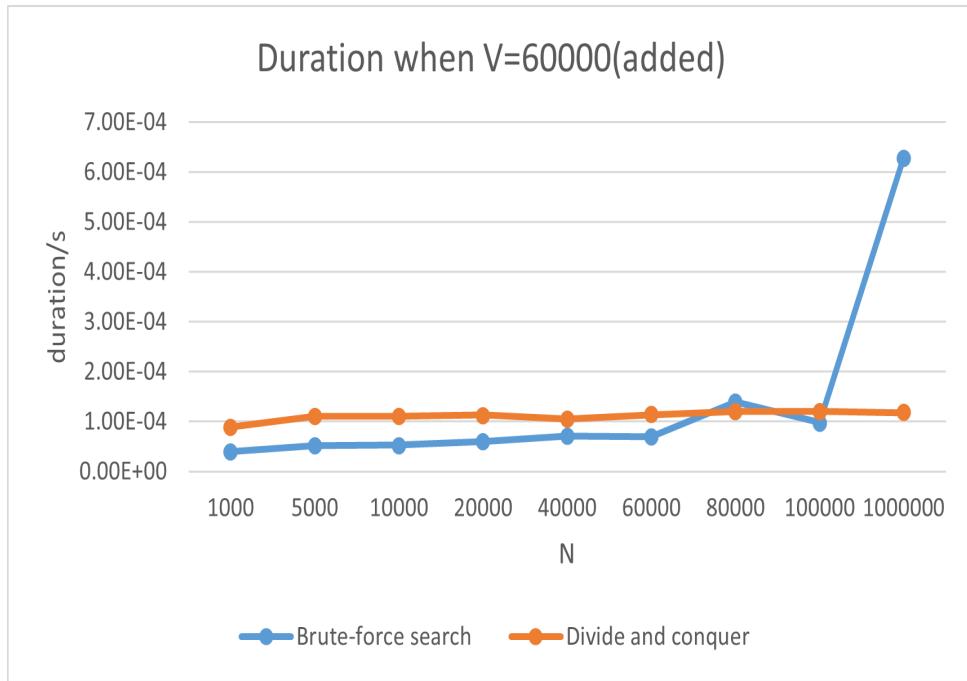
### 3.5 When $V = 40000$

| $V=40000$          | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   | 1000000  |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 383      | 480      | 507      | 636      | 590      | 1057     | 684      | 831      | 6775     |
|                    | Total Time(sec) | 0.383    | 0.48     | 0.507    | 0.636    | 0.59     | 1.057    | 0.684    | 0.831    | 6.775    |
|                    | Duration(sec)   | 3.83E-05 | 4.80E-05 | 5.07E-05 | 6.36E-05 | 5.90E-05 | 1.06E-04 | 6.84E-05 | 8.31E-05 | 6.78E-04 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 841      | 1064     | 1068     | 1059     | 1078     | 1102     | 1042     | 1060     | 1109     |
|                    | Total Time(sec) | 0.841    | 1.064    | 1.068    | 1.059    | 1.078    | 1.102    | 1.042    | 1.06     | 1.109    |
|                    | Duration(sec)   | 8.41E-05 | 1.06E-04 | 1.07E-04 | 1.06E-04 | 1.08E-04 | 1.10E-04 | 1.04E-04 | 1.06E-04 | 1.11E-04 |

Figure 11: Running times for  $V = 40000$ Figure 12: Plotted running times for  $V = 40000$

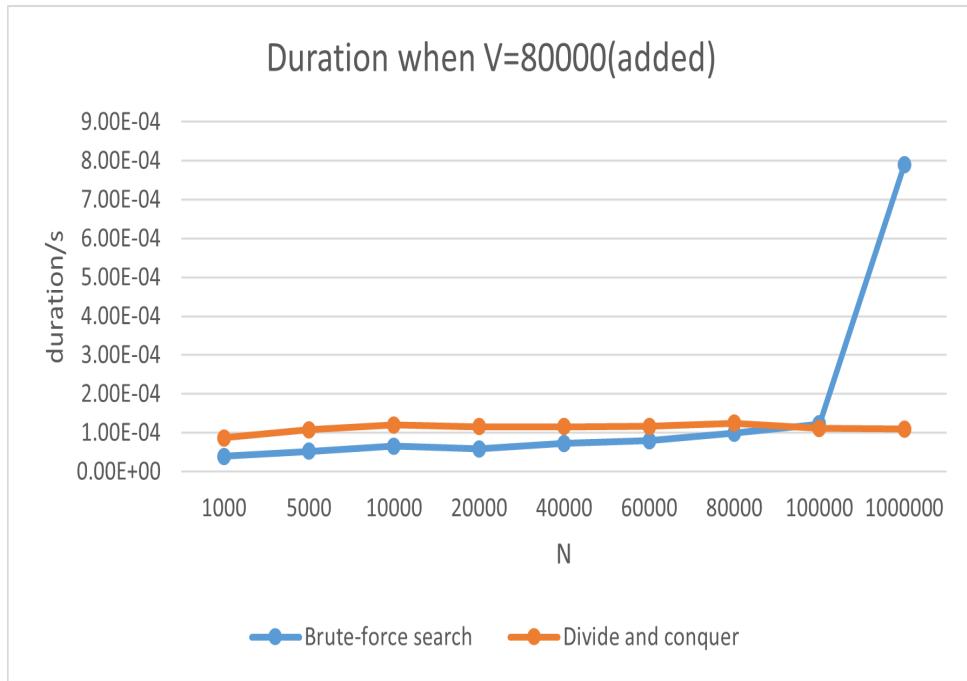
### 3.6 When $V = 60000$

| $V=60000$          | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   | 1000000  |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 392      | 525      | 533      | 596      | 712      | 693      | 1388     | 982      | 6276     |
|                    | Total Time(sec) | 0.392    | 0.525    | 0.533    | 0.596    | 0.712    | 0.693    | 1.388    | 0.982    | 6.276    |
|                    | Duration(sec)   | 3.92E-05 | 5.25E-05 | 5.33E-05 | 5.96E-05 | 7.12E-05 | 6.93E-05 | 1.39E-04 | 9.82E-05 | 6.28E-04 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 889      | 1110     | 1109     | 1130     | 1052     | 1140     | 1199     | 1200     | 1179     |
|                    | Total Time(sec) | 0.889    | 1.11     | 1.109    | 1.13     | 1.052    | 1.14     | 1.199    | 1.2      | 1.179    |
|                    | Duration(sec)   | 8.89E-05 | 1.11E-04 | 1.11E-04 | 1.13E-04 | 1.05E-04 | 1.14E-04 | 1.20E-04 | 1.20E-04 | 1.18E-04 |

Figure 13: Running times for  $V = 60000$ Figure 14: Plotted running times for  $V = 60000$

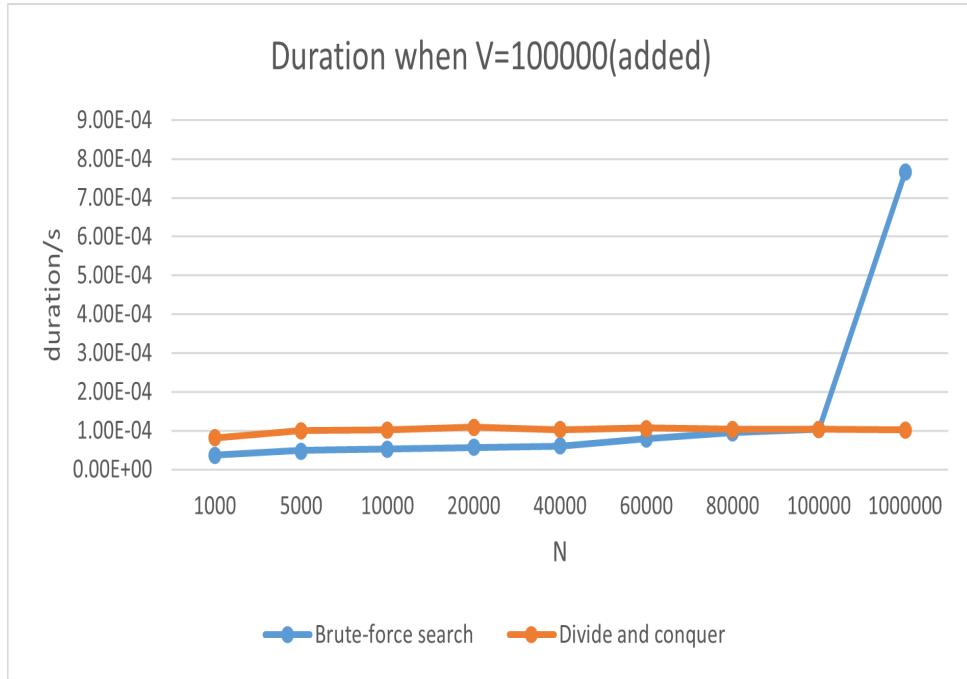
### 3.7 When $V = 80000$

| $V=80000$          | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   | 1000000  |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 397      | 522      | 660      | 586      | 729      | 796      | 994      | 1225     | 7893     |
|                    | Total Time(sec) | 0.397    | 0.522    | 0.66     | 0.586    | 0.729    | 0.796    | 0.994    | 1.225    | 7.893    |
|                    | Duration(sec)   | 3.97E-05 | 5.22E-05 | 6.60E-05 | 5.86E-05 | 7.29E-05 | 7.96E-05 | 9.94E-05 | 1.23E-04 | 7.89E-04 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 865      | 1074     | 1196     | 1159     | 1157     | 1172     | 1250     | 1116     | 1101     |
|                    | Total Time(sec) | 0.865    | 1.074    | 1.196    | 1.159    | 1.157    | 1.172    | 1.25     | 1.116    | 1.101    |
|                    | Duration(sec)   | 8.65E-05 | 1.07E-04 | 1.20E-04 | 1.16E-04 | 1.16E-04 | 1.17E-04 | 1.25E-04 | 1.12E-04 | 1.10E-04 |

Figure 15: Running times for  $V = 80000$ Figure 16: Plotted running times for  $V = 80000$

### 3.8 When $V = 100000$

| $V=100000$         | N               | 1000     | 5000     | 10000    | 20000    | 40000    | 60000    | 80000    | 100000   | 1000000  |
|--------------------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Brute-force search | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 368      | 490      | 529      | 568      | 603      | 786      | 952      | 1032     | 7658     |
|                    | Total Time(sec) | 0.368    | 0.49     | 0.529    | 0.568    | 0.603    | 0.786    | 0.952    | 1.032    | 7.658    |
|                    | Duration(sec)   | 3.68E-05 | 4.90E-05 | 5.29E-05 | 5.68E-05 | 6.03E-05 | 7.86E-05 | 9.52E-05 | 1.03E-04 | 7.66E-04 |
| Divide and conquer | Iterations(K)   | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    | 10000    |
|                    | Ticks           | 818      | 1004     | 1020     | 1092     | 1028     | 1072     | 1040     | 1045     | 1013     |
|                    | Total Time(sec) | 0.818    | 1.004    | 1.02     | 1.092    | 1.028    | 1.072    | 1.04     | 1.045    | 1.013    |
|                    | Duration(sec)   | 8.18E-05 | 1.00E-04 | 1.02E-04 | 1.09E-04 | 1.03E-04 | 1.07E-04 | 1.04E-04 | 1.05E-04 | 1.01E-04 |

Figure 17: Running times for  $V = 100000$ Figure 18: Plotted running times for  $V = 100000$

## 4 Analysis and Comments

### 4.1 Brute-force search

#### (1). Time complexity:

- Worst case: Every time the program tries to find the targeted group,it searches every possible group,so the worst case is to search until the last number group,so the time complexity is  $O(n^2)$ .

#### (2). Space complexity:

- Brute-force search is none-recursive and only uses constant amount of local variables.It only needs few variables: res,i,j.So the space complexity is  $O(1)$ ,only uses constant amount of space for the variables.

### 4.2 Divide and conquer

#### (1). Time complexity:

- Divide and conquer uses recursion structure to divide the array into small arrays smaller than a constant size.This recursive step needs  $O(\log n)$  time.And after division,the brute search process takes at most  $O(20^2)$  ,which is constant time  $O(1)$ .So the total time complexity is  $O(\log n)$ .

#### (2). Space complexity:

- Divide and conquer divide the array into 2 parts in every recursion,so the recursion step needs  $O(\log n)$  space.And in every recursion takes constant space for a few variables:res,mid,res\_left,res\_right.So the total space complexity is  $O(\log n)$ .

### 4.3 Comprehensive analysis

| Algorithm          | Time Complexity | Space Complexity |
|--------------------|-----------------|------------------|
| Brute-force search | $O(n^2)$        | $O(1)$           |
| Divide and search  | $O(\log n)$     | $O(\log n)$      |

Table 1: Complexity of the two algorithm

- The test data mostly proves the time complexity of the two algorithm that when the size of array ( $n$ ) is big enough,divide and conquer will take obviously less time than brute-force search.

- But we find that the brute-force search takes  $O(n^2)$  time complexity, but the actual **time cost doesn't grow as quadratic time**. After analyze I think this is the result of the problem itself. In this problem we need to find two integers that  $a+b=c$  and the worst case is the last two integer in the array. However if we calculate the situation we can find the array can offer  $n(n-1)$  combinations, which is much bigger than  $2V$  (the max sum), so when given a target sum, **there's usually many group can meet the problem's demand** in most cases. Therefore the function do not need quadratic time to solve the problem. And for this reason we imply 20 as the size threshold of Divide and conquer and it did quite well.

## 5 Appendix

```

1 # include<stdio.h>
2 # include<stdlib.h>           //For the random function
3 # include<time.h>
4 clock_t start1,stop1,start2,stop2;    //To record the start and end time
5                                     //of functions
6 double duration1,duration2;          //duration1 to count the time of
7                                     //function1,duration2 to count the time of function2
8 int ticks1,ticks2;
9
10
11 int* generate_array_of_n(int n,int max_num); //A function to generate
12                                     //random data for functions
13 void print_list(int* arr,int n);           //A function to print the
14                                     //list member out to help find a proper sum input
15
16 int brute_force_search(int* arr,int sum,int n); //Definition of
17                                     //function1
18 int div_con(int* arr,int sum,int left_p,int right_p,int count); // //
19                                     //Definition of function2
20
21
22 int main(void){
23     int n,max_num;                  //n represents the size of data,
24                                     //max_num represents the integers won't be larger than it
25     printf("Please input the size and max of the number(split them with
26           a space):");
27     scanf("%d %d",&n,&max_num);      //Input manual to contrl different
28                                     //situation

```

```

18     int* arr;           //The target arr
19     int res1,res2;      //res1,res2 to note whether the
20         function find the a and b
21     arr = generate_array_of_n(n,max_num); //Generate the random array
22     printf("The random array generate successfully!\n");
23                     //Show the generate step process
24                     well
25     printf("If you want to see the detail numbers , please enter number
26         1(or will skip it)!!!only use it when N is quite small:");
27     int flag;           //Use a flag to check if the
28         tester want to see the array
29     scanf("%d",&flag);
30     if(flag == 1){      //If you want to see the randomly
31         generated array input 1 or input any number else
32         print_list(arr,n); //Print the whole list
33     }
34
35     int sum;            //Take the targeted sum of
36         algorithm
37
38     srand((unsigned int)time(NULL)); //Initialize rand seed
39     start1 = clock();           //Mark the starting time of
40         function1
41     for (int i = 0; i < 10000; i++){ //Please change loop number if
42         needed
43         int k = rand() % n;        //To ensure the sum is working,I
44             choose to randomly select 2 element to calculate the sum
45         int j = rand() % n;
46         sum = arr[k] + arr[j];
47         res1 = brute_force_search(arr, sum, n); //Get the return result
48     } //Operate the function1 for K times(I'll set it without input
49         )
50     stop1 = clock();           //Mark the starting time of
51         function2
52     ticks1 = (int)(stop1-start1); //Calculate the running ticks
53     duration1 = ((double)(stop1 -start1))/CLK_TCK;//Calculate the running
54         time
55

```

```

44     start2 = clock();           //Mark the starting time of
        function2(actually all the same below)
45     for (int i = 0; i < 10000; i++){ //Please change loop number if
        needed
        int k = rand() % n;          //Same in function1
46     int j = rand() % n;
47     sum = arr[k] + arr[j];
48     res2 = div_con(arr, sum, 0, n-1,i); //Get the return result
49 }
50
51     stop2 = clock();
52     ticks2 = (int)(stop2-start2);
53     duration2 = ((double)(stop2 -start2))/CLK_TCK;//Calculate the running
        time
54
55     if (res1 == 1 && res2 == 1){ //Check if both find the a and b
        //Print the ticks and duration of the two functions
56     printf("The operation of function1(Brute-force) is: %e (s)and %d
        ticks\n",duration1,ticks1);
57     printf("The operation of function2(Divide and conquer) is: %e (s)
        and %d ticks\n",duration2,ticks2);
58                                         //Print out the result
59 }
60     }else if(res1 != 1){
61         printf("Function1 failed"); //If function1 failed
62     }else if(res2 != 1){
63         printf("Function2 failed"); //If function2 failed
64     }else{
65         printf("Both failed");      //If both failed
66     }
67     return 0;
68 }
69
70 int* generate_array_of_n(int n,int max_num){
71     int* arr = (int*)malloc(n * sizeof(int)); //Create a new array to
        contain numbers
72     if(arr == NULL){                      //Check the location
        printf("Memory allocation failed\n");
73     exit(1);                            //If malloc failed exit the
        program

```

```

75     }
76     srand((unsigned int)time(NULL)); //Initialize random seeds
77     for (int i = 0;i < n;i++){
78         arr[i] = rand() % (max_num+1); //Use % to contrl the numbers are
79             no more than max_num
80     }
81     return arr;
82 }
83
84 void print_list(int* arr,int n){ //Print the array randomly generated
85     for (int i = 0;i < n;i++){ //Go through all integers
86         printf("%d ",arr[i]);
87         if (i % 30 == 29) //30 integers each line
88             printf("\n");
89     }
90 }
91
92 int brute_force_search(int* arr,int sum,int n){
93     int res = 0; //Mark if the function find the
94         answer
95     for (int i = 0;i < n;i++){ //Go through every possible group
96         for (int j = i+1;j < n;j++){
97             if (arr[i] + arr[j] == sum){//Check the sum
98                 res = 1;
99                 return res; //Target found
100            }
101        }
102    }
103    return res; //Target not found
104 }
105
106 int div_con(int* arr,int sum,int left_p,int right_p,int count){
107     int res = 0; //Mark if the function find the
108         answer
109     if((right_p - left_p) > 20){ //Only start brute-force searching
110         when subset is smaller than 20
111         int mid = (right_p + left_p)/2; //Find the mid index
112         int res_left,res_right; //Mark result of left and right part
113             search

```

```

108     res_left = div_con(arr, sum, left_p, mid, count);
109     if(res_left == 1){           //Check if there's answer in the left
110         half first
111         res = 1;               //If answer found in the left half,
112         skip right half and return til the main function
113         return res;
114     }
115     res_right = div_con(arr, sum, mid, right_p, count);
116     if(res_right == 1){           //If there's no answer in the left
117         half, then check the right half
118         res = 1;               //Same above
119         return res;
120     }
121     for(int i = left_p;i < right_p;i++){//When the size of subset is
122         small enough, deploy brute search
123         for(int j = i+1;j < right_p;j++){
124             if(arr[i] + arr[j] == sum){
125                 res = 1;
126                 return res;           //Target found
127             }
128         }
129     }

```

## 6 Declaration

I hereby declare that all the work done in this project titled “Project 1 : Performance Measurement (A+B) ” is of my independent effort.