Joseph Bode
Daniel Ty
Benjamin Stewart
12/11/2020

The group divided the work up in these ways.

Java File: Benjamin Stewart, Daniel Ty
Algorithms: Benjamin Stewart, Daniel Ty
Graphs: Benjamin Stewart, Joseph Bode
Complexity Analysis: Joseph Bode
Solutions: Daniel Ty, Benjamin Stewart

**<u>*DISCLAIMER 1*: We returned the *Values* that make up *t* instead of the *Indices.*</u>**

Solution: Brute Force

For this algorithm, if the target is 0, we return true and the empty set. If our given array, S, is 0 and our target is nonzero, we return false. We recursively check if isSumBF(S, n - 1, target, indices) or isSumBF(S, n - 1, target - S[n - 1], indices), return true and indices. Else, we return false.

Solution: Dynamic Programming

Create 2D Boolean Array *subset* of size [n+1][t+1]. We then fill out default true and false values. We then traverse through *subset* and check if subset[i-1][j] == j -> subset[i][j]. Else we check if arr[i-1] > j -> subset[i][j] = false else -> subset[i][j] = subset[i-1][j-arr[i-1]]. Finally we return *subset*[n][sum]

Solution: Clever

Split the Indices, Compute a table T of all subsets of L that don't exceed t. Return True if any I in T = t, else continue. Compute a table W of all subsets of H that yield a subset of S that don't exceed t. Return True if any J in W = t, else continue. Sort table W in ascending order. If S[I] + S[J] = t, return true, else if > t break to return to the next iteration of outer loop since W is sorted, else continue. If we get through the whole method with no return true, we return false.

Complexity Analysis: Brute Force

In this algorithm, we took every subset of S and computed their sums and then compared it against the target. There are $2^n$ subsets for every set and each of them hold up to $n$ elements. This is so that each sum evaluation costs $\Theta(n2^n)$ arithmetic operations. Every subset can be represented as a sequence of n bits. This indicates whether each element is or is not in the subset. The sum of each of these fits $lg\ t$ bits. This is the space that is required to show $t$ itself. We also know that any sum exceeding this would be disregarded and discarded. Since it is known that

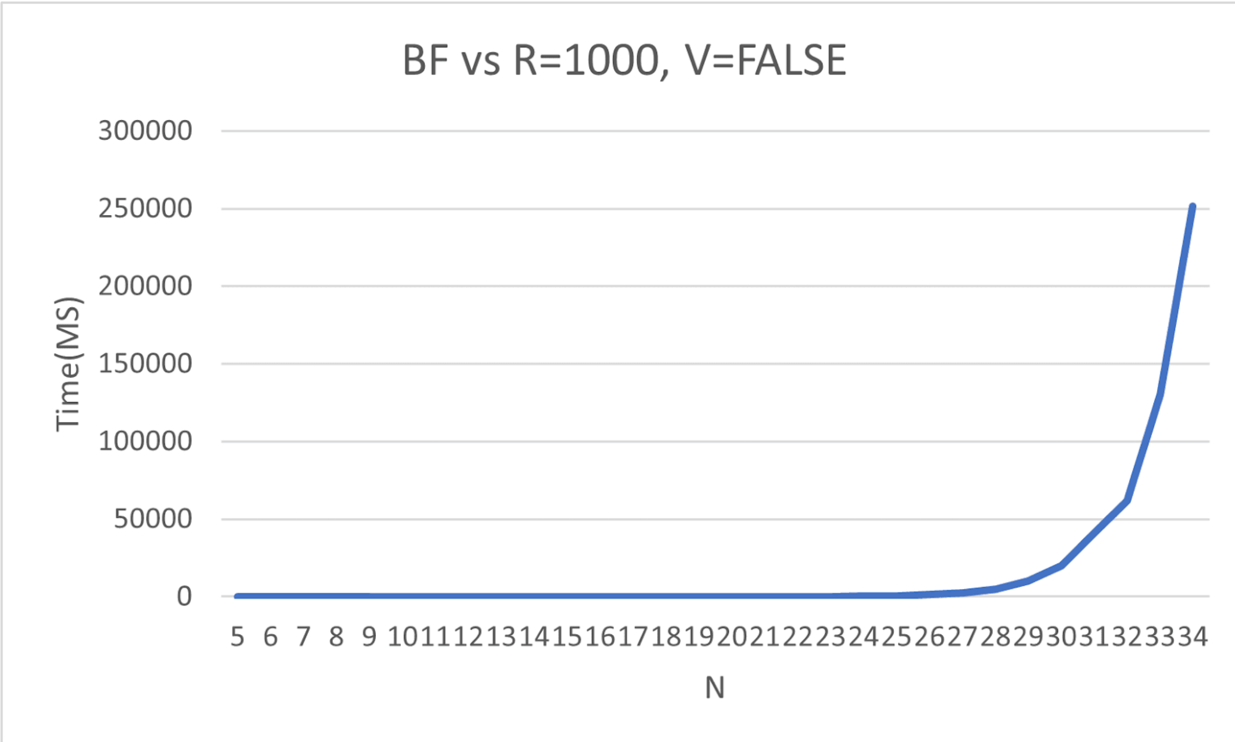
$lg\ t \in O(n)$ (the space needed to store $t$ is in the same general area as the space that is needed to store the input), the space complexity is $O(n)$ bits ( $O(1)$ elements).

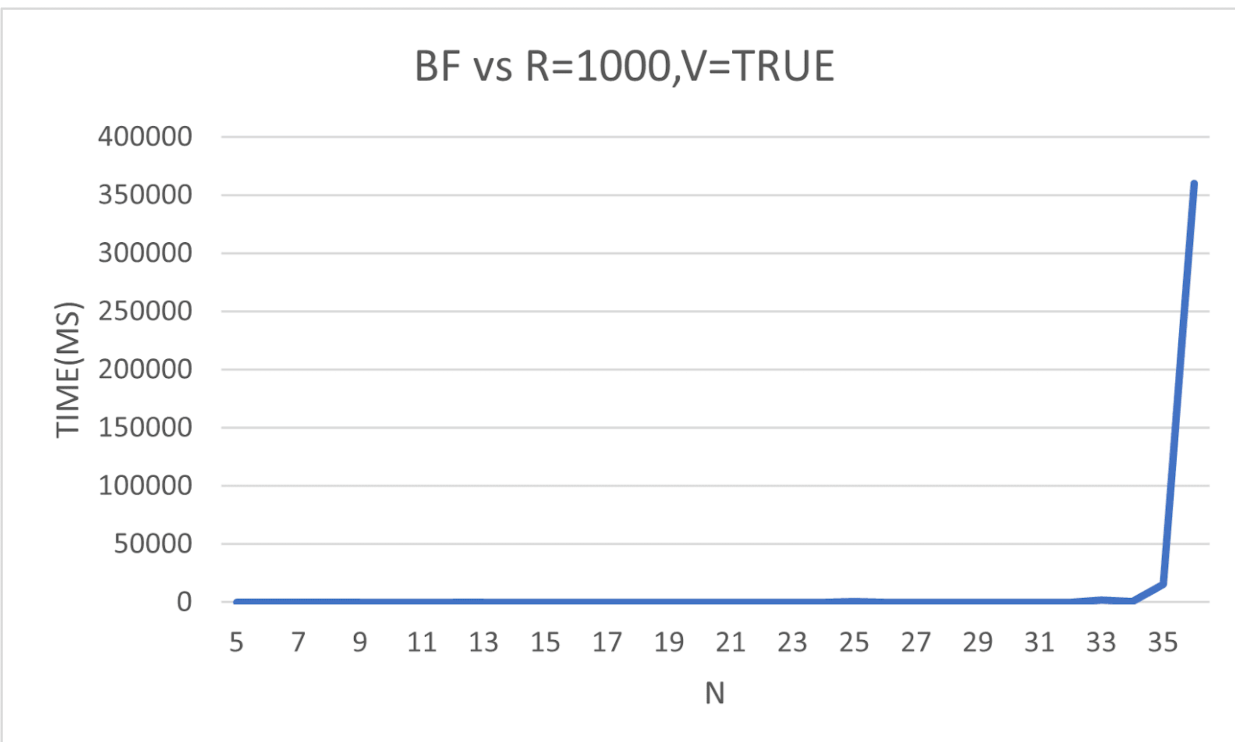
Complexity Analysis: Dynamic Programming

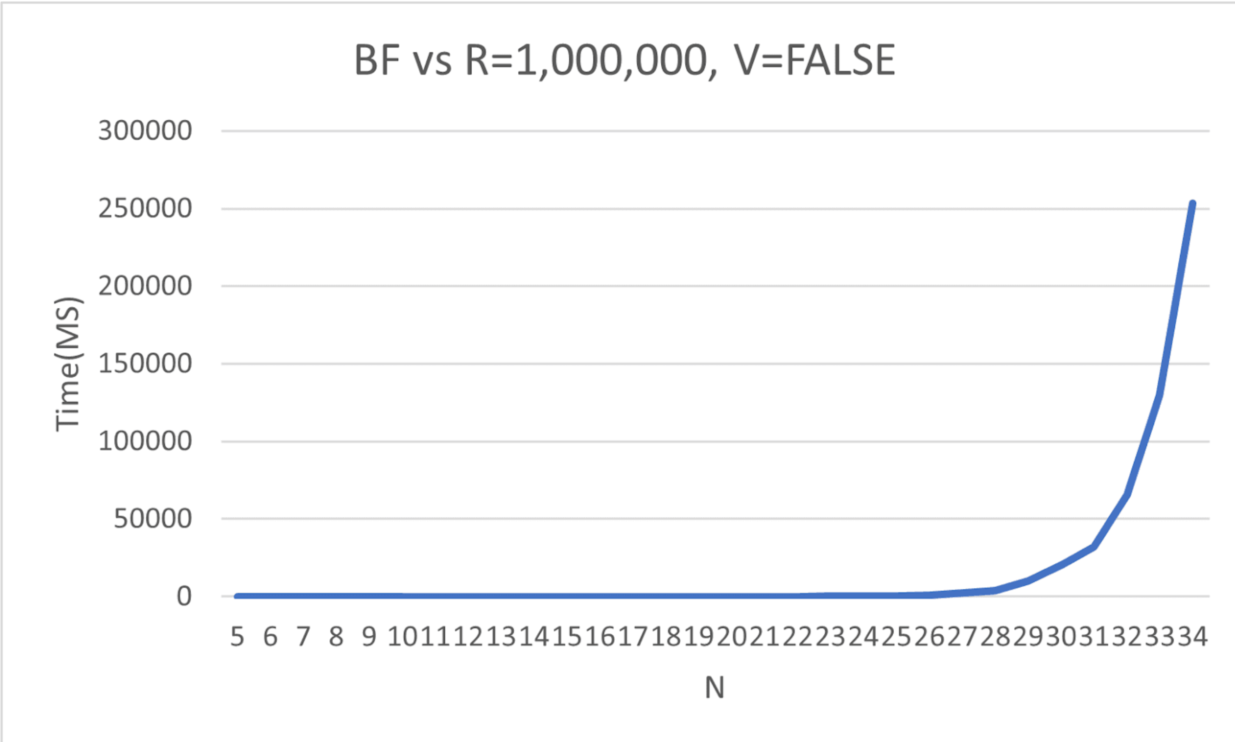
The dynamic programming algorithm is the same thing that was gone over in the course files and lectures. There are two nested loops. One of them being $t$ steps and the other being $n$ steps. The total number of steps (with both of them taking constant time) is $nt$ and the time complexity should be $\Theta(nt)$. We already know from the first analysis that $lg\ t \in O(n)$, we can simplify this to $\Theta(n2^n)$ operations. The DP algorithm needs an $n\ x\ t$ matrix of numbers that is large enough to store a value similar to $t$. That is $O(lg\ t) = O(n)$ bits. The final space complexity is $O(nt\ lg\ t) = O(n^2 2^n)$ bits, which is $O(n2^n)$ elements.

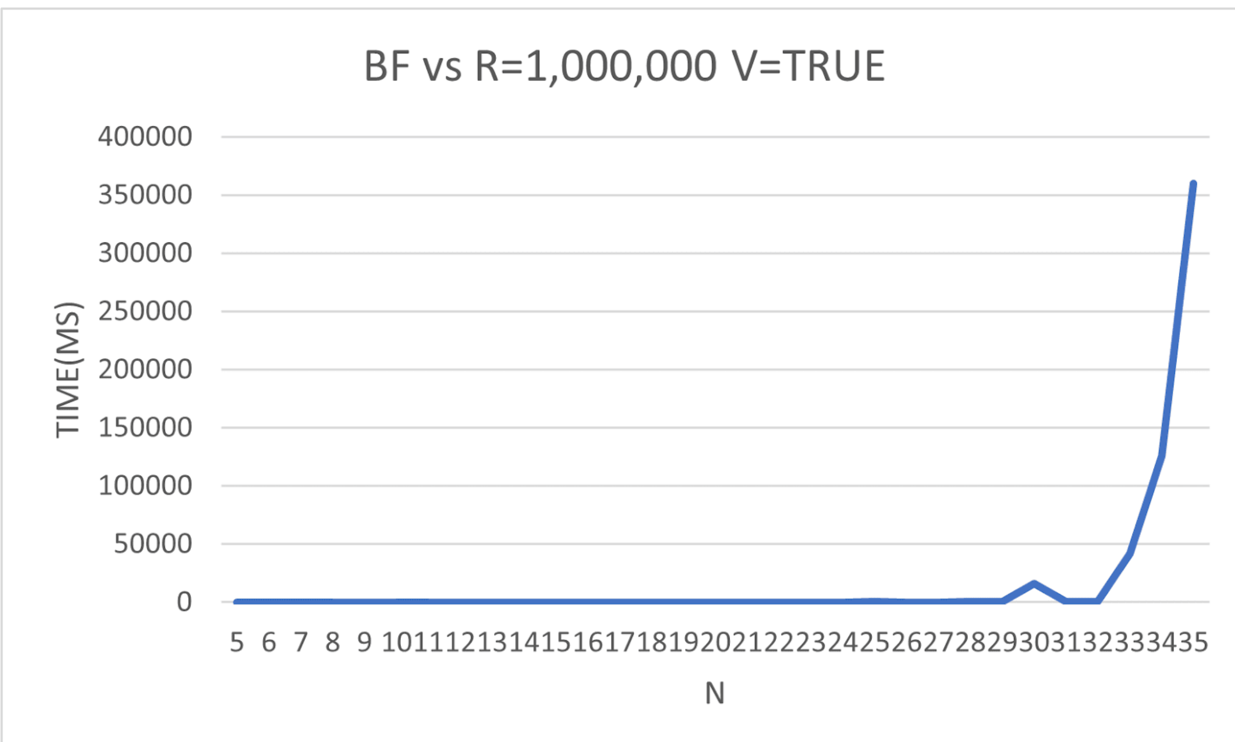
Complexity Analysis: Clever Algorithms

The first step listed in the algorithm from the homework pdf takes at worst $O(n)$ time as we are just splitting a list of size $n$. We can only note that Step 6 is even easier than this step, Step 6 is just $O(1)$. In Step 2, scanning is performed over all the subsets such that $I \subseteq L$. $L$ contains $n/2$ elements, so there are $2^{n/2}$ subsets. For each one the algorithm computes $\sum_{i \in I} S[i]$ consisting of up to $n/2$ elements. This step does perform $\Theta(n2^{n/2})$ additions. Similar to step 2, Step 3 does $\Theta(n2^{n/2})$ additions. The table size is $N = 2^{n/2}$, step 4 can be done with a $\Theta(N lg(n))$ sorting method. The cost is $\Theta(n2^{n/2})$ comparisons. Step 5 consists of finding the subset $J \subseteq H$ with maximum weight not passing t - weight(I) can be done with a binary search on a table W at the cost $\Theta(lg(n))$. Since the table size is $N = 2^{n/2}$, the cost per a search is $\Theta(n)$ comparisons. Since a search will be performed for each of the $2^{n/2}$ elements of $I \subseteq L$, the final cost of this step is $\Theta(n2^{n/2})$ comparisons. The total time complexity factoring all of this is $\Theta(n2^{n/2})$ arithmetic operations. This clever algorithm does require two tables of subsets of lists of length $n/2$ elements. There exists $2^{n/2}$ such subsets for each. This totals to $2 * 2^{n/2}$ list entries. Every subset can be shown by a sequence of $n/2$ bits. The overall space used is $(n/2) * 2 * 2^{n/2}$, or $O(n2^{n/2})$ or $O(n2^{n/2})$ elements.


BF vs R=1000, V=FALSE
300000
250000
200000
150000
100000
50000
0
Time(MS)
5 6 7 8 9 10111213141516171819202122232425262728293031323334
N

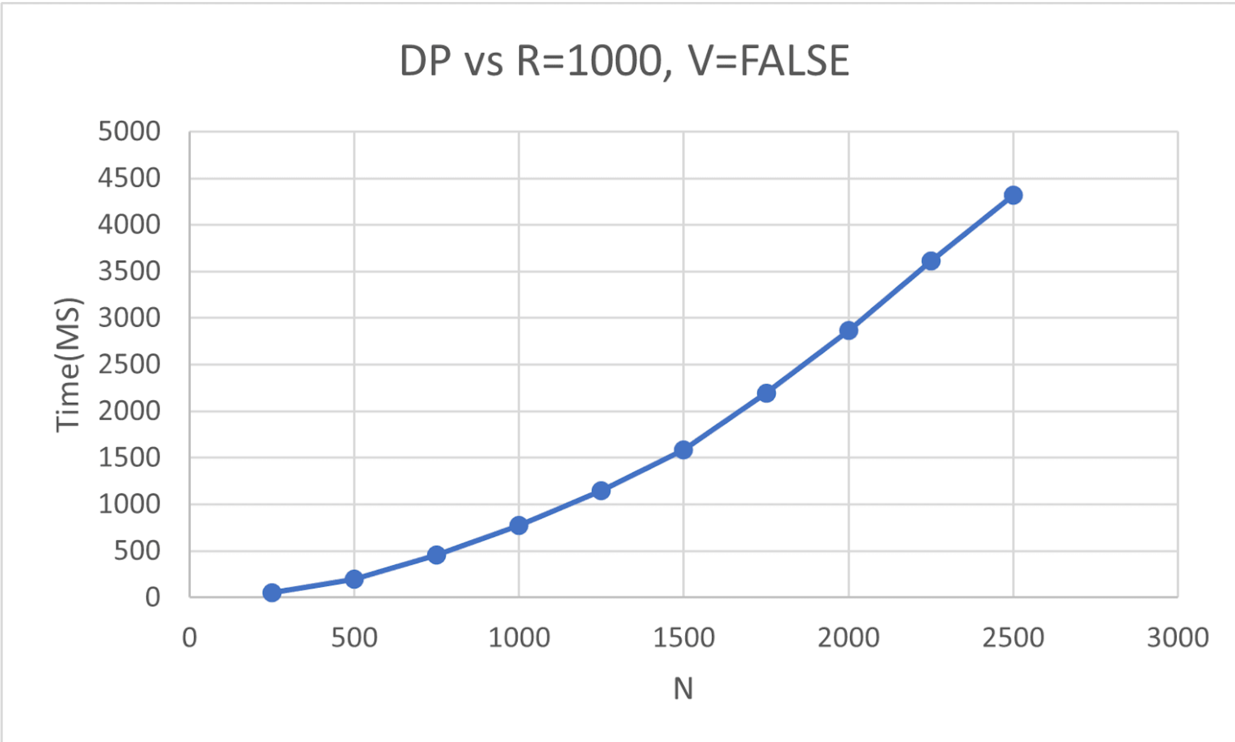

BF vs R=1000,V=TRUE
400000
350000
300000
250000
200000
150000
100000
50000
0
TIME(MS)
5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35
N

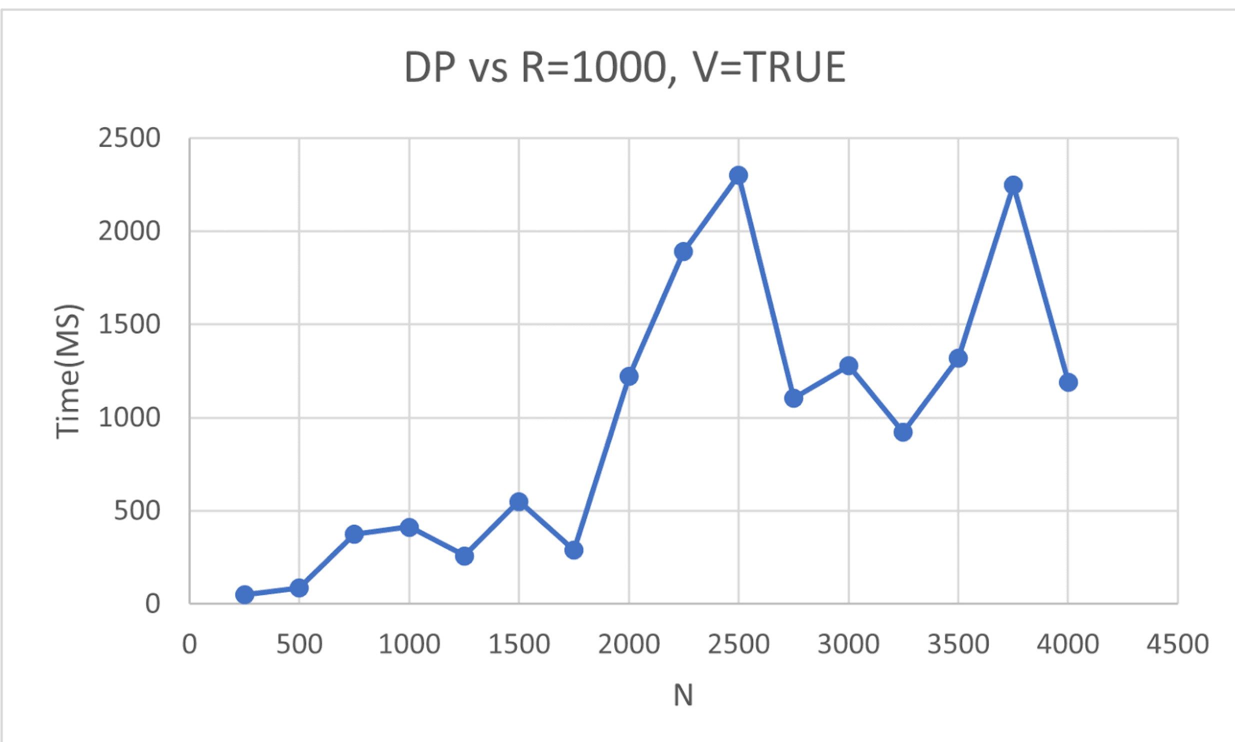

BF vs R=1,000,000, V=FALSE
300000
250000
200000
150000
100000
50000
0
Time(MS)
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
N

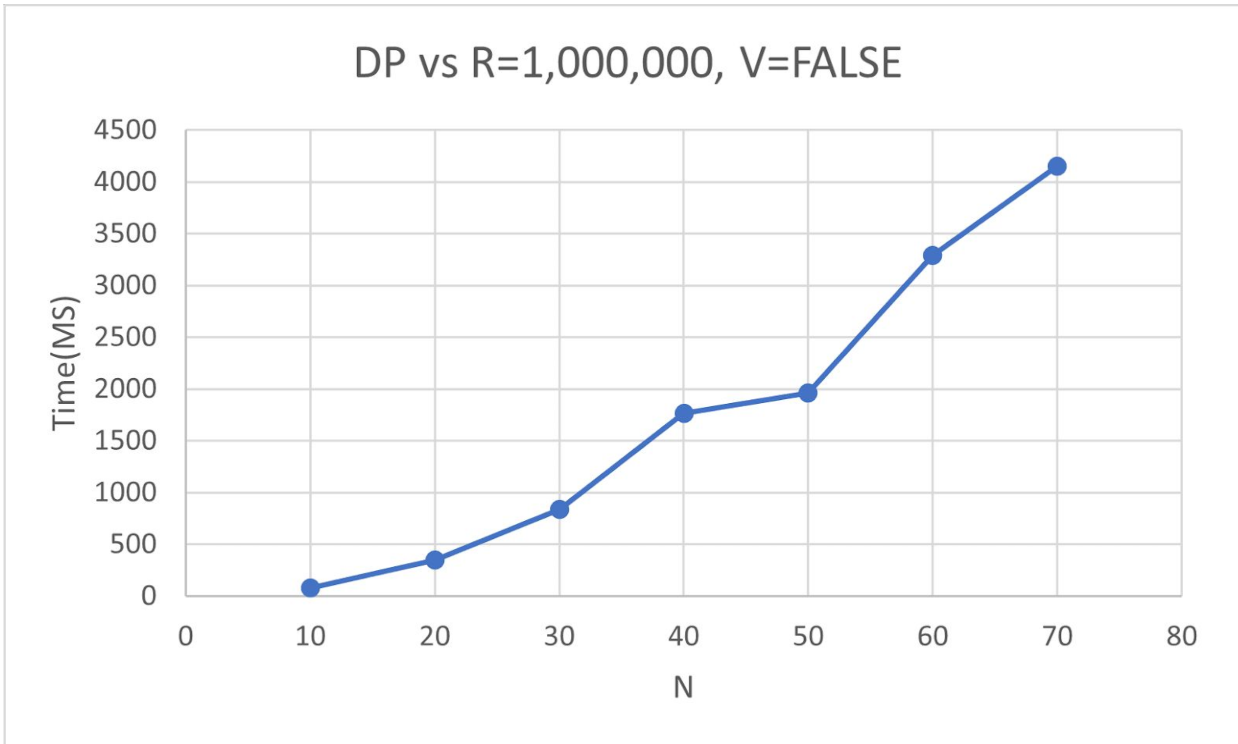

BF vs R=1,000,000 V=TRUE
400000
350000
300000
250000
200000
150000
100000
50000
0
TIME(MS)
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
N

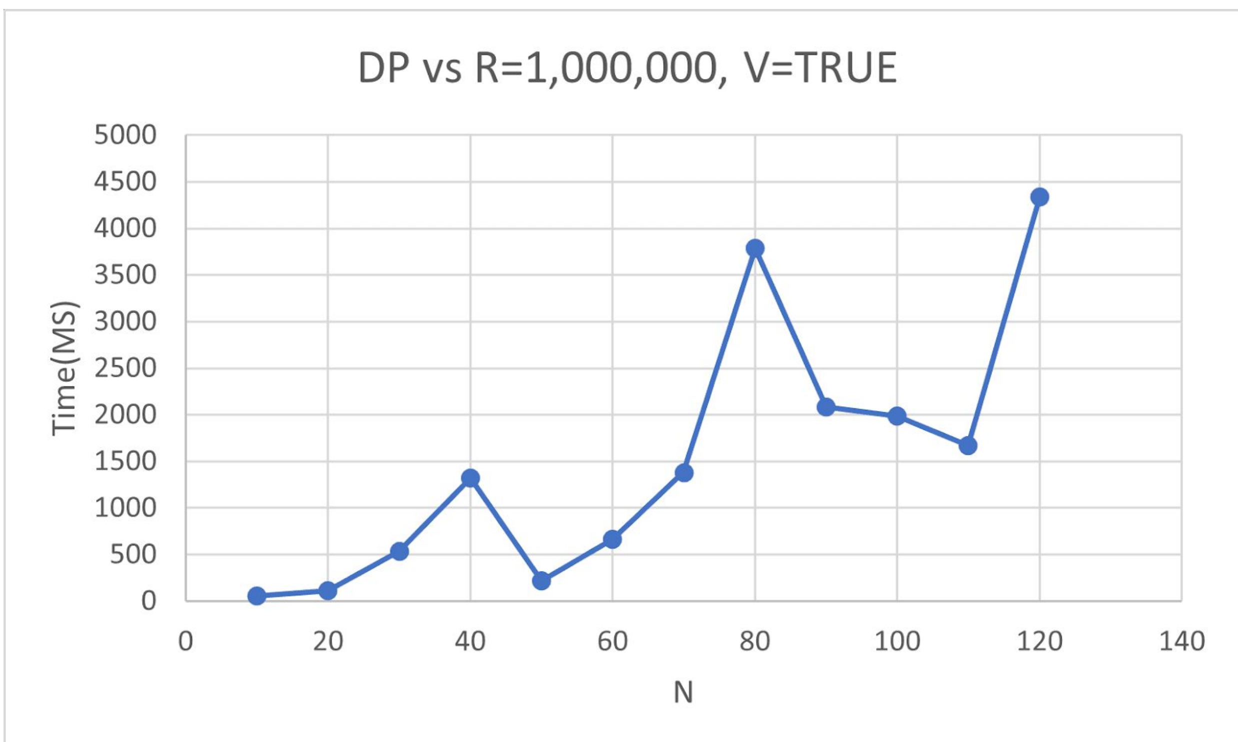
***DISCLAIMER 2*: We stopped at certain values not due to time but due to space.**

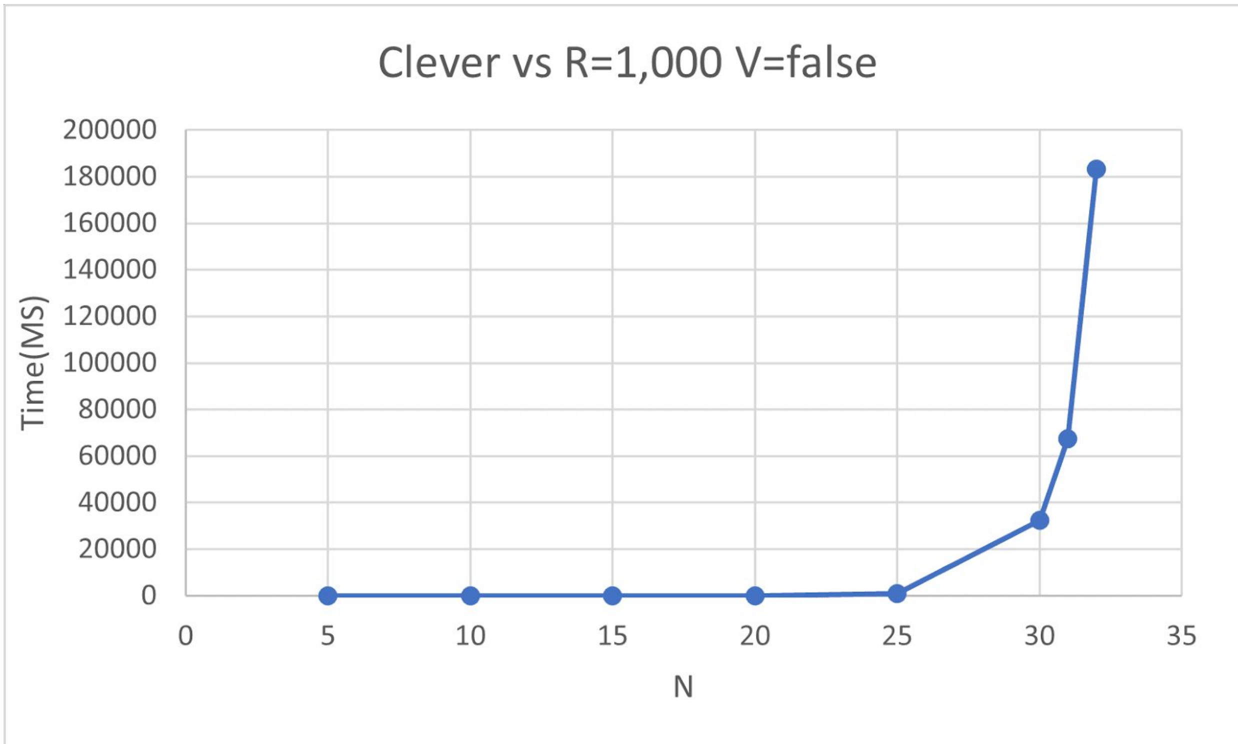
**We ran out of memory (Space constraint).**

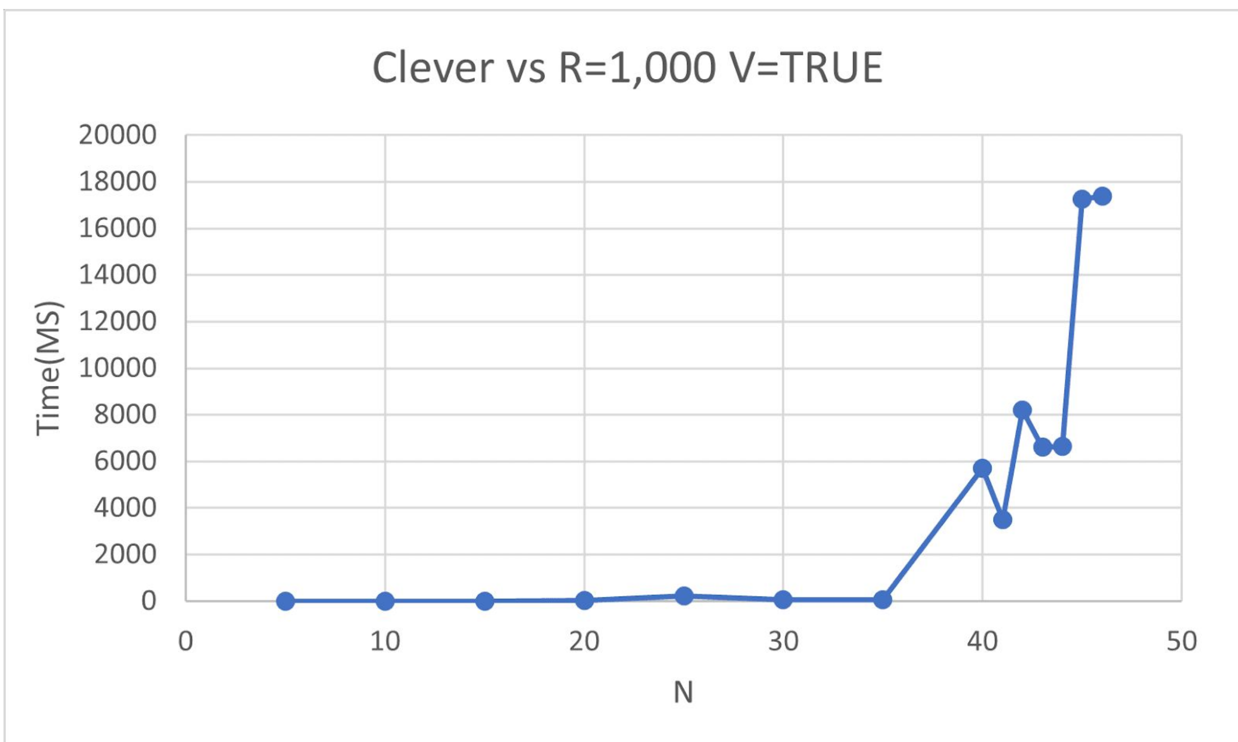


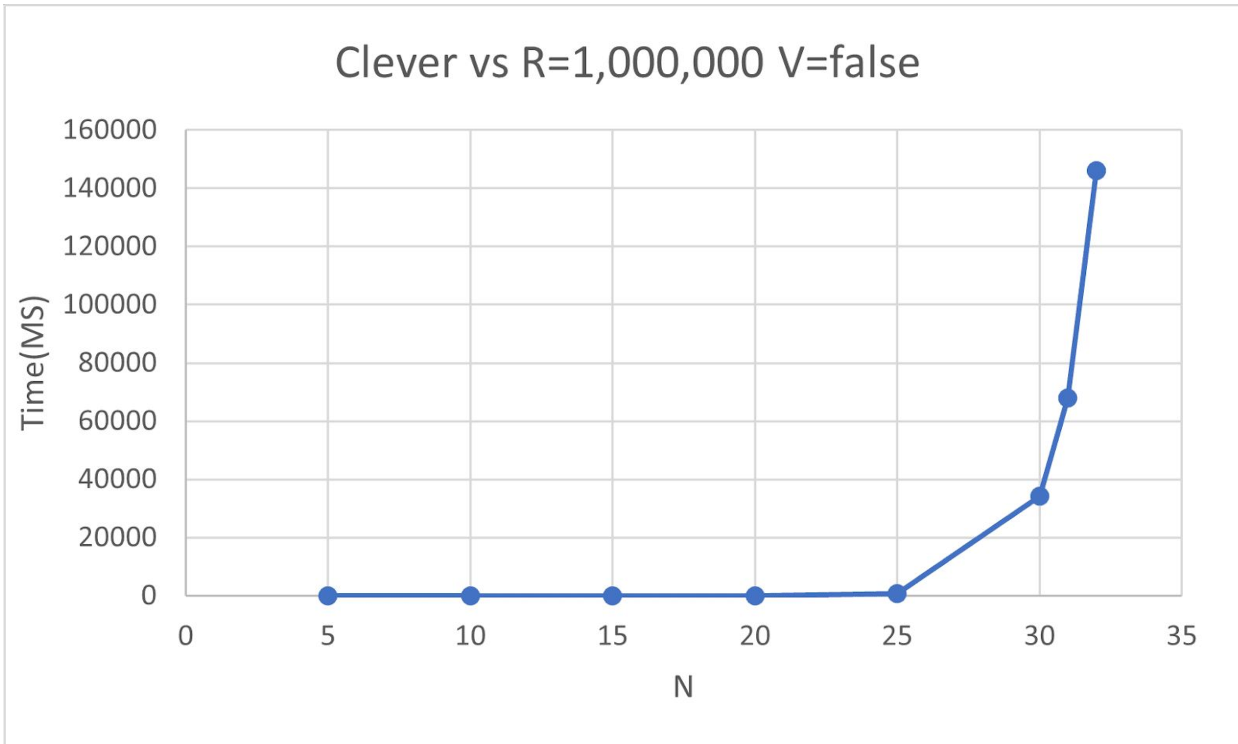



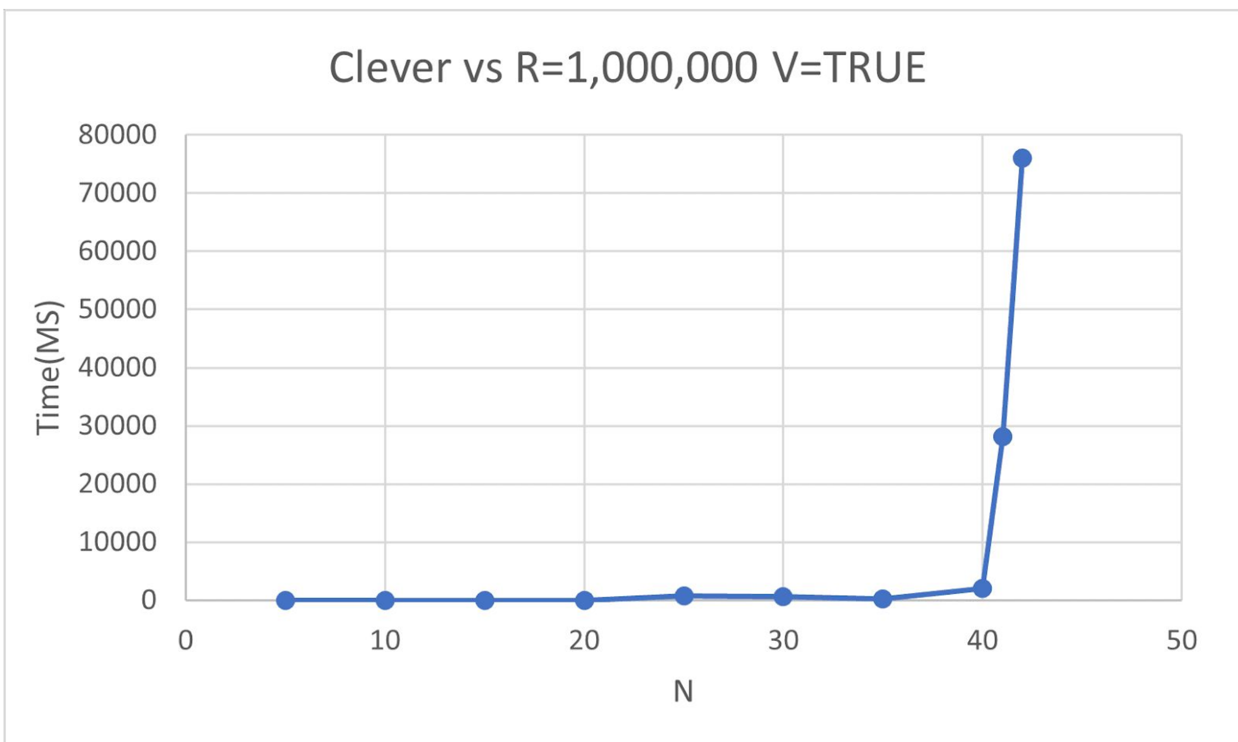

DP vs R=1,000,000, V=FALSE
Time(MS)
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
N


DP vs R=1,000,000, V=TRUE
Time(MS)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100
120
140
N


Clever vs R=1,000 V=false
Time(MS)
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30
35
N


Clever vs R=1,000 V=TRUE
Time(MS)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
10
20
30
40
50
N


Clever vs R=1,000,000 V=false
Time(MS)
N
160000
140000
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25
30
35


Clever vs R=1,000,000 V=TRUE
Time(MS)
N
80000
70000
60000
50000
40000
30000
20000
10000
0
0
10
20
30
40
50