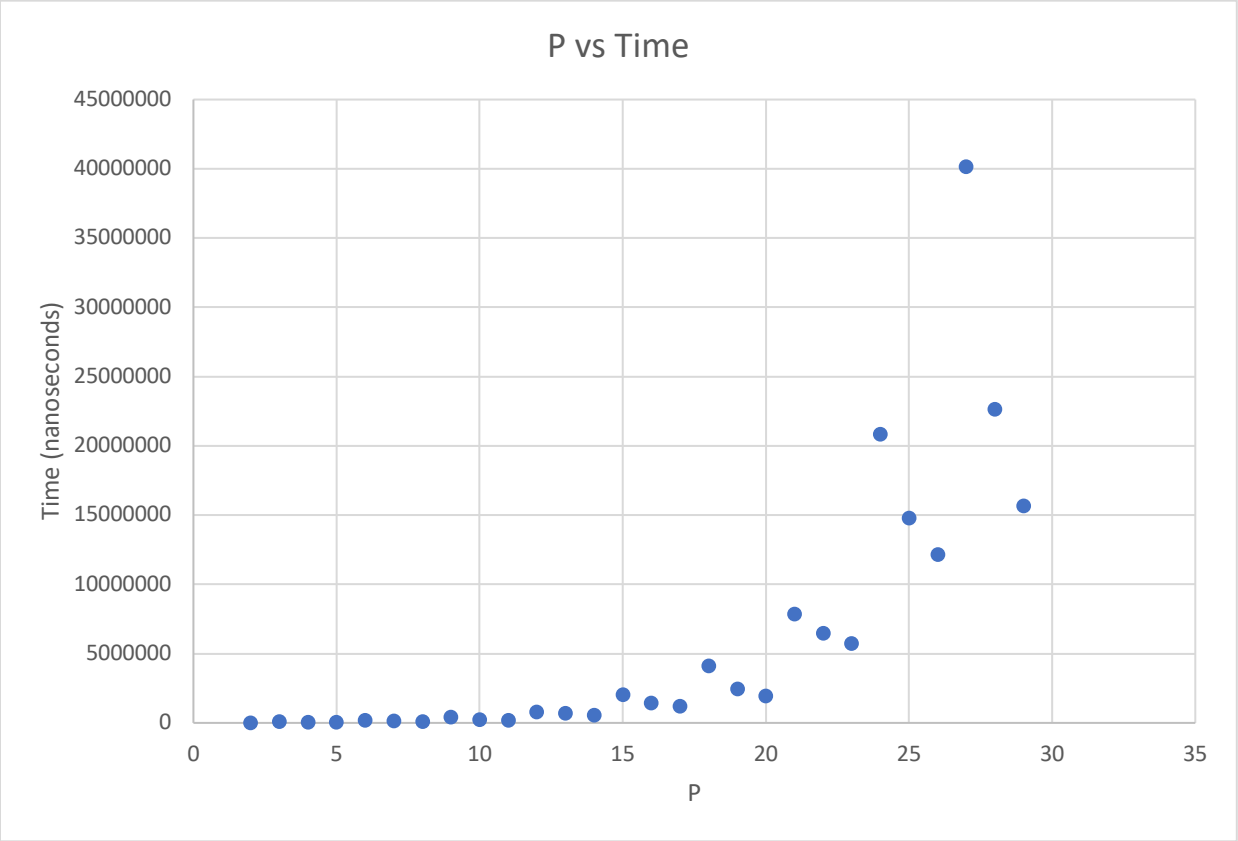
Eli Laird
CS 7350
Homework 2

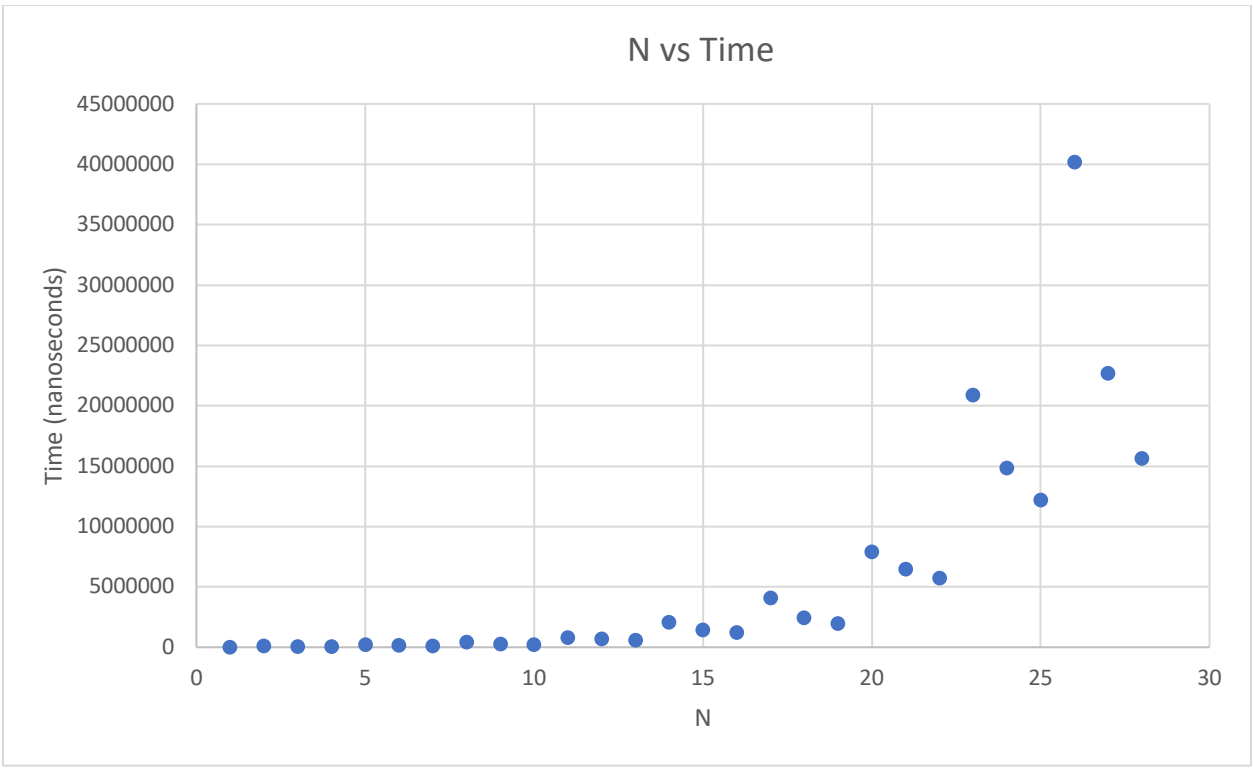
Problem 1:

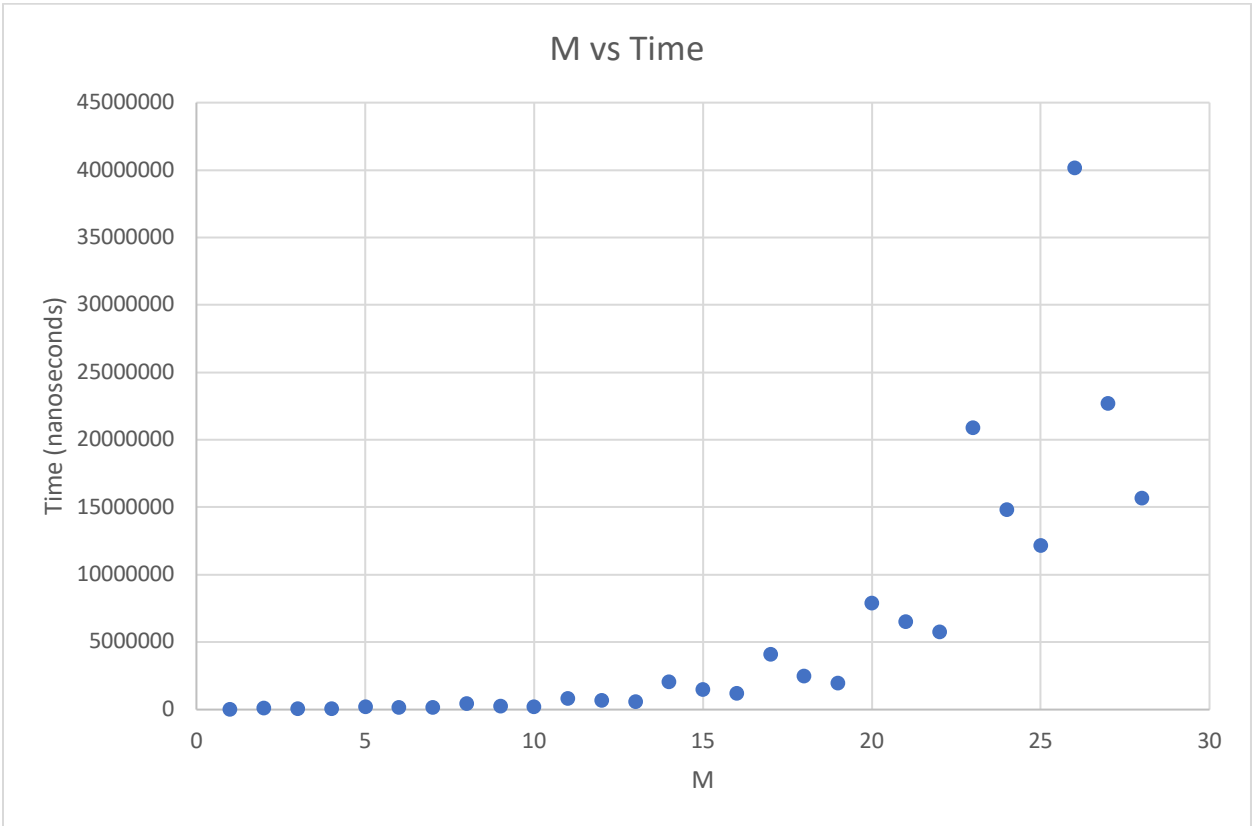
- **Asymptotic Bounds:** $\boldsymbol{\theta(pn - m)}$
- **Support:** From the data in the table, we can conclude that the algorithm is bounded by $\theta(pn)$. We see that the function is bounded by *p* when we increase *p* by a factor of 10 and keep *m* and *n* the same, *t* increase by a factor of 10. When we increase *n* by a factor of 2 and keep *p* and *m* the same, *t* increases by a factor of 2. When we increase *p* by a factor of 10 AND increase *n* by a factor of 2, *t* increases by a factor of 10 * 2, or in other words *pn*. Whenever *m* is increased, this reduces the amount of collisions when drawing from the distribution, thus decreasing the value of *t*. This relation of *m* to *t* is independent from *pm* so the final bounds are $\theta(pn - m)$.

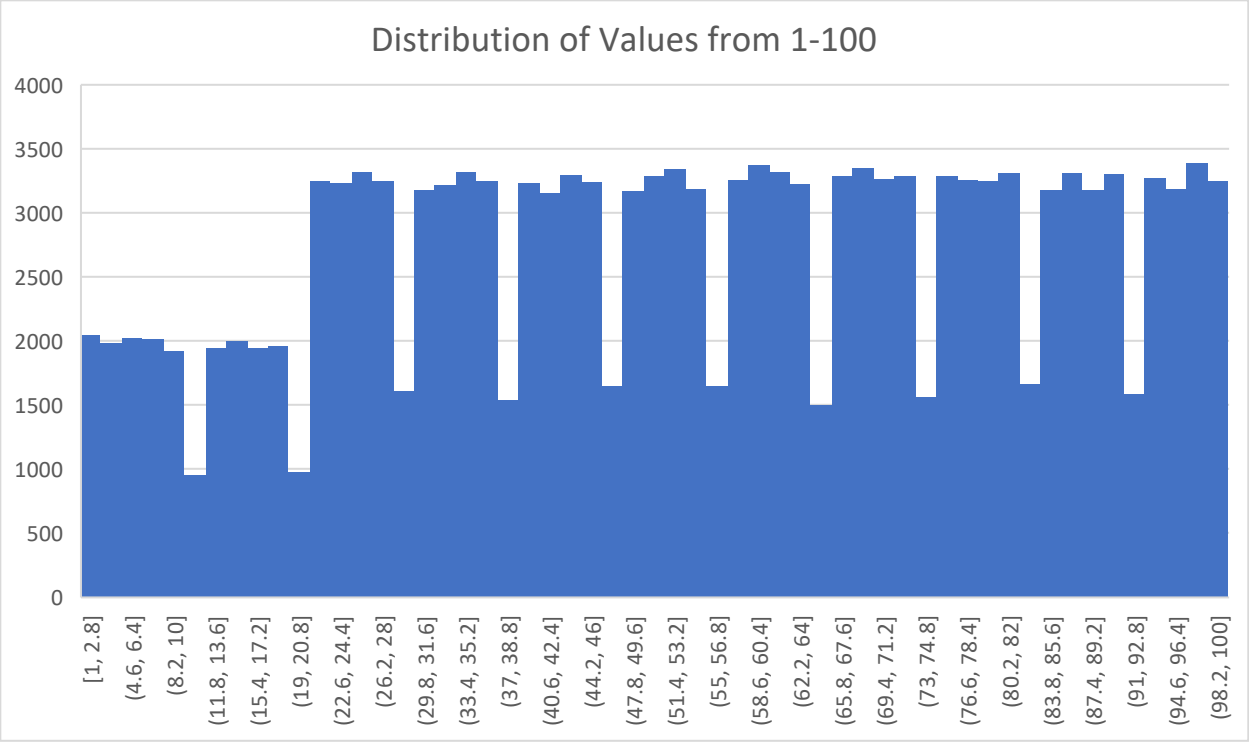
Unique Iterations with Uniform Distribution			
p	n	m	t (nanoseconds)
100	5	20	85017
100	5	100	67829
100	5	500	58515
100	10	20	208219
100	10	100	153057
100	10	500	125981
100	15	20	448288
100	15	100	257391
100	15	500	206341
1000	5	20	819032
1000	5	100	686410
1000	5	500	587768
1000	10	20	2053418
1000	10	100	1456296
1000	10	500	1214312
1000	15	20	4108572
1000	15	100	2463805
1000	15	500	1972694
10000	5	20	7876627
10000	5	100	6490328
10000	5	500	5724835
10000	10	20	20865959
10000	10	100	14819085
10000	10	500	12174799
10000	15	20	40157165
10000	15	100	22674070

10000	15	500	15654036




N vs Time
Time (nanoseconds)
45000000
40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
0
5
10
15
20
25
30
N


M vs Time
Time (nanoseconds)
45000000
40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
0
5
10
15
20
25
30
M


Distribution of Values from 1-100
4000
3500
3000
2500
2000
1500
1000
500
0
[1, 2.8]
(4.6, 6.4]
(8.2, 10]
(11.8, 13.6]
(15.4, 17.2]
(19, 20.8]
(22.6, 24.4]
(26.2, 28]
(29.8, 31.6]
(33.4, 35.2]
(37, 38.8]
(40.6, 42.4]
(44.2, 46]
(47.8, 49.6]
(51.4, 53.2]
(55, 56.8]
(58.6, 60.4]
(62.2, 64]
(65.8, 67.6]
(69.4, 71.2]
(73, 74.8]
(76.6, 78.4]
(80.2, 82]
(83.8, 85.6]
(87.4, 89.2]
(91, 92.8]
(94.6, 96.4]
(98.2, 100]

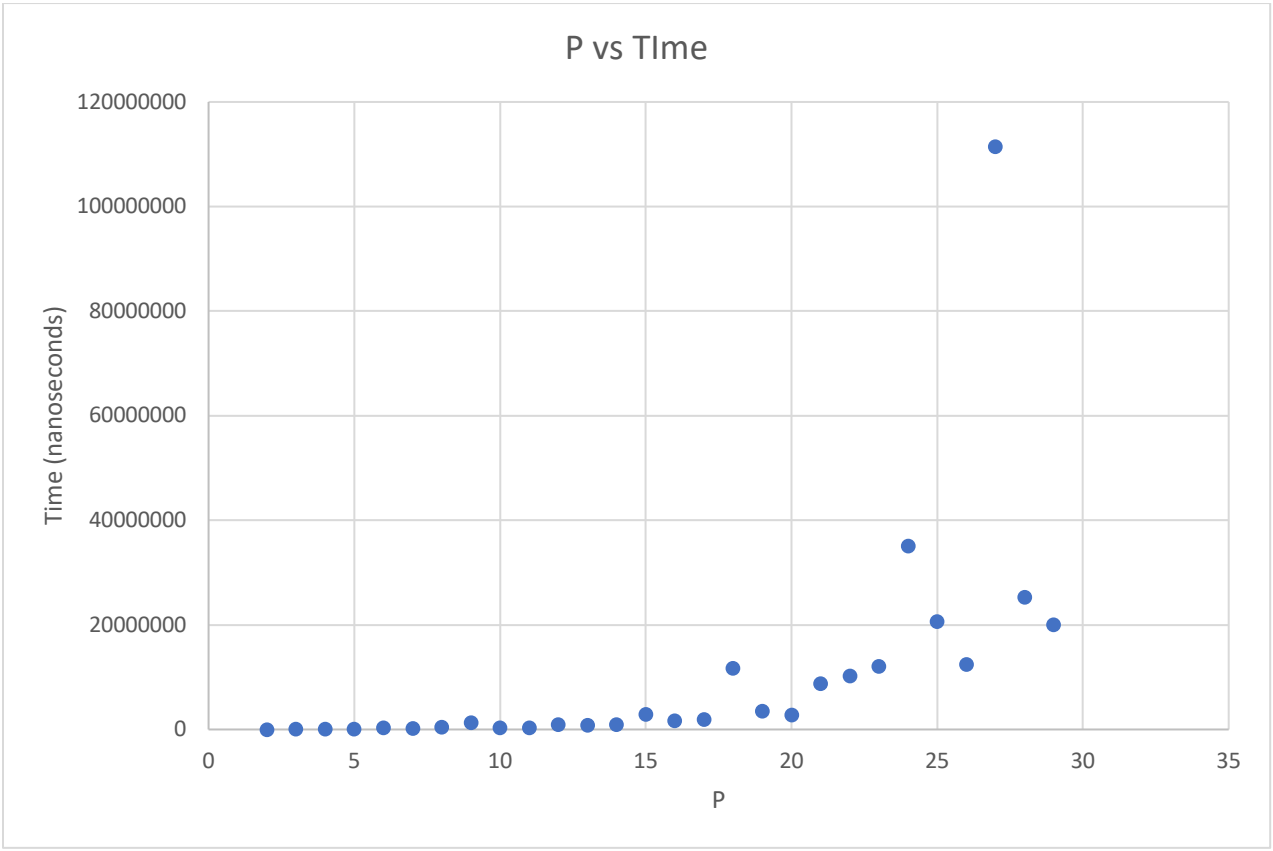
Code:

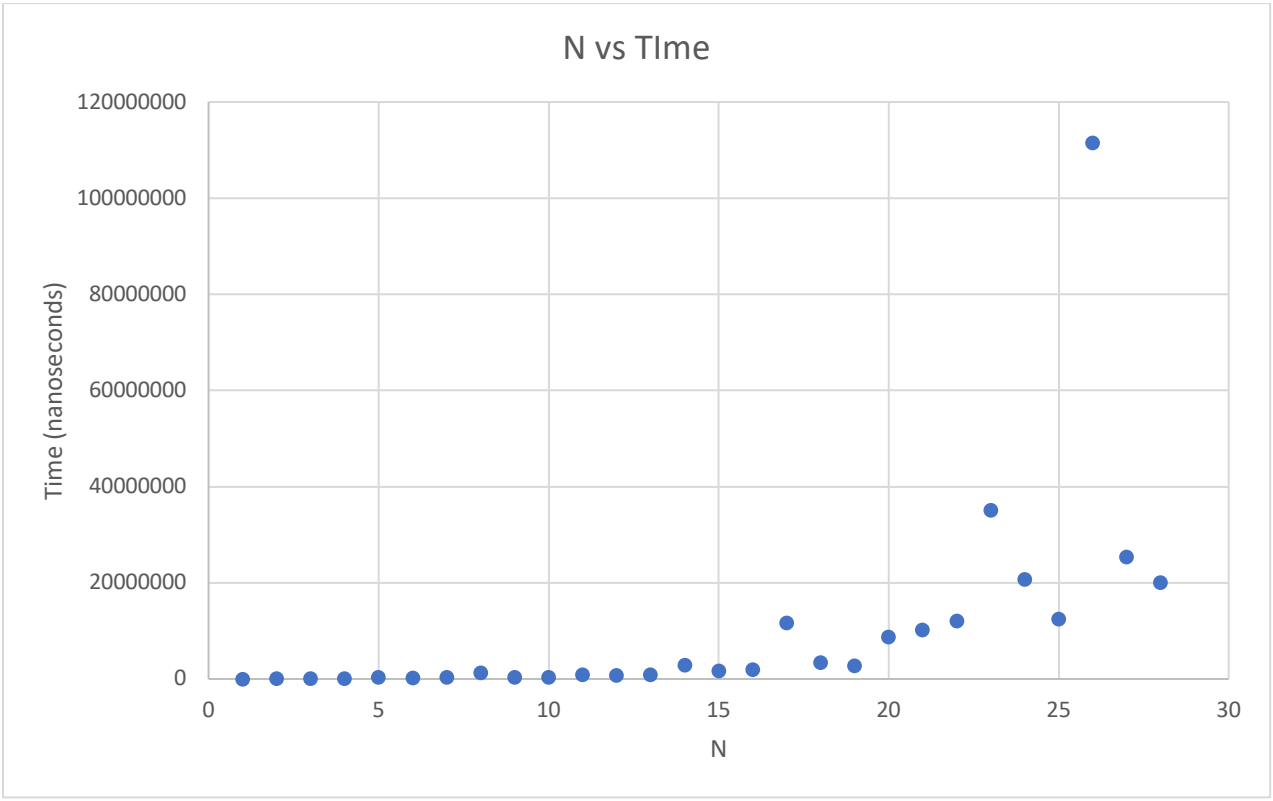
```
bool isUnique(int j, int arr[], int start, int end){
    for(int i = start; i<=end; i++)
        if(arr[i] == j)
            return false;
    return true;
}

long long problem1(long n, long m, int p, std::default_random_engine generator,
bool printDist = false){
    using namespace std::chrono;

    int arr[p * n];
    int iter = 0;
    std::uniform_int_distribution<long int> dist(1,m);

    //measure starting time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    //loop p iterations
    for(int i = 0; i < p; i++){
        int uniqueNums = 0;
        int setIter = iter;

        //loop until n unique numbers are generated
        while(uniqueNums < n){
            int x = dist(generator);

            if(isUnique(x, arr, setIter, setIter+n-1)){
                arr[iter++] = x;
                uniqueNums++;
            }
        }
    }

    //measure total time
    high_resolution_clock::time_point end = high_resolution_clock::now();
    auto total_time = duration_cast<nanoseconds>(end - start).count();
    std::cout << "p: " << p << " n: " << n << " m: " << m << " time: " <<
total_time << std::endl;

    if(printDist){
        //store distribution of numbers to file
        std::ofstream fout("problem1_dist.csv");
        for(int i = 0; i < n*p; i++)
            fout << arr[i] << ",\n";
        fout.close();
    }


    return total_time;

}
```

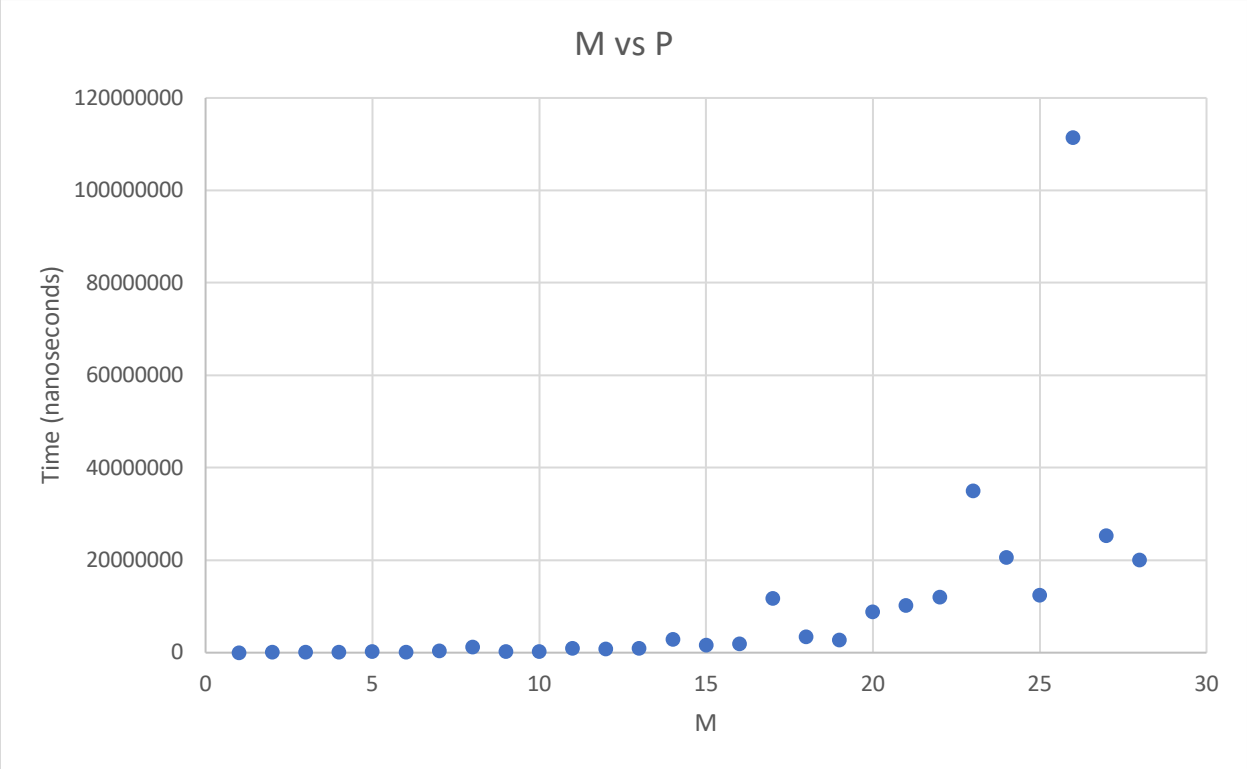
Problem 2:

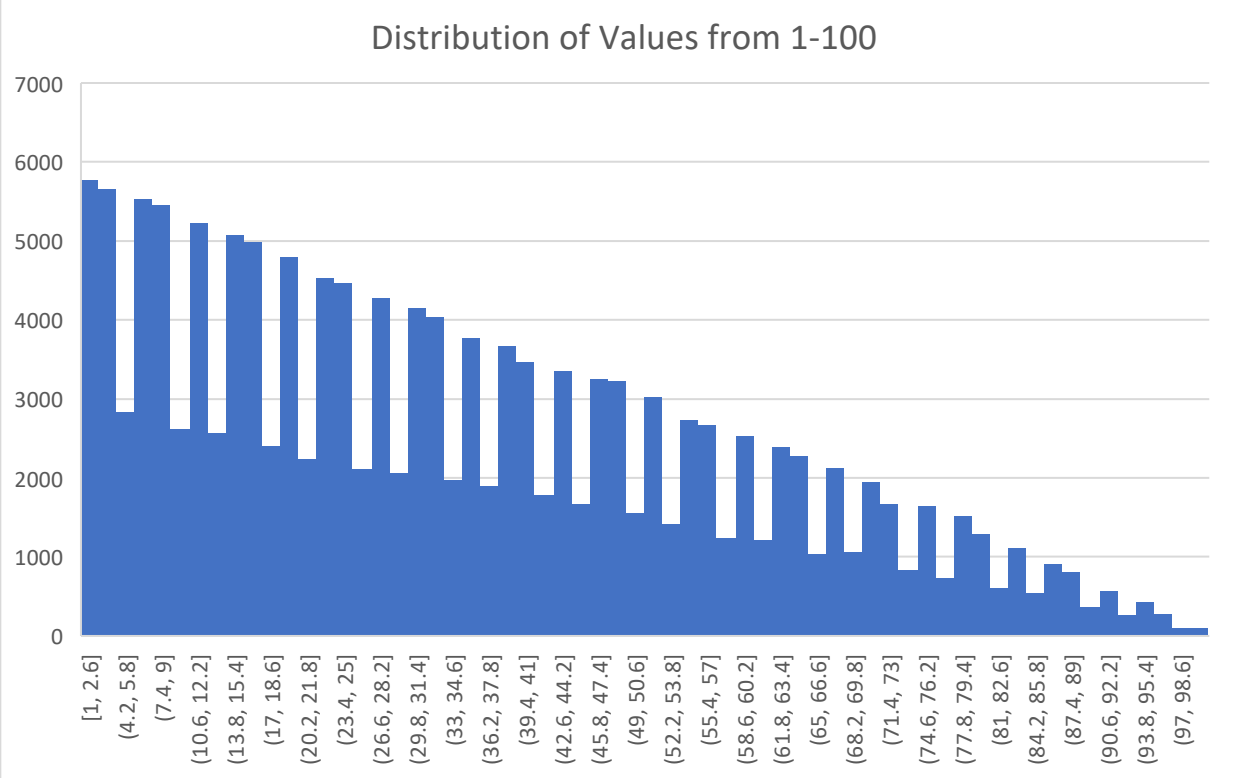
- **Asymptotic Bounds:** $\boldsymbol{\theta(pn)}$
- **Support:** From the data in the table, we can conclude that the algorithm is bounded by $\theta(pn)$. We see that the function is bounded by *p* when we increase *p* by a factor of 10 and keep *m* and *n* the same, *t* increase by a factor of 10. When we increase *n* by a factor of 2 and keep *p* and *m* the same, *t* increases by a factor of 2. When we increase *p* by a factor of 10 and increase *n* by a factor of 2, *t* increases by a factor of 10 * 2, or in other words *pn*. Whenever *m* is increased, this reduces the amount of collisions when drawing from the distribution, thus decreasing the value of *t*. This decrease is less than in Problem 1 due to the *skewed* distribution. This relation of *m* to *t* is independent from *pm* so the final bounds are $\theta(pn - m)$.

Unique Iterations for Skewed Distribution			
p	n	m	t (nanoseconds)
100	5	20	88378
100	5	100	80031
100	5	500	81402
100	10	20	309971
100	10	100	168864
100	10	500	384123
100	15	20	1264845
100	15	100	299151
100	15	500	291882
1000	5	20	886520
1000	5	100	765789
1000	5	500	878860
1000	10	20	2928734
1000	10	100	1616659
1000	10	500	1895595
1000	15	20	11686244
1000	15	100	3463592
1000	15	500	2709388
10000	5	20	8775396
10000	5	100	10248381
10000	5	500	12085335
10000	10	20	35060850
10000	10	100	20666251
10000	10	500	12446172
10000	15	20	111452014
10000	15	100	25310469
10000	15	500	20004239


P vs TIme
120000000
100000000
80000000
60000000
40000000
20000000
0
Time (nanoseconds)
0
5
10
15
20
25
30
35
P


N vs TIme
120000000
100000000
80000000
60000000
40000000
20000000
0
Time (nanoseconds)
0
5
10
15
20
25
30
N


M vs P
120000000
100000000
80000000
60000000
40000000
20000000
0
Time (nanoseconds)
0
5
10
15
20
25
30
M


Distribution of Values from 1-100
7000
6000
5000
4000
3000
2000
1000
0
[1, 2.6]
(4.2, 5.8]
(7.4, 9]
(10.6, 12.2]
(13.8, 15.4]
(17, 18.6]
(20.2, 21.8]
(23.4, 25]
(26.6, 28.2]
(29.8, 31.4]
(33, 34.6]
(36.2, 37.8]
(39.4, 41]
(42.6, 44.2]
(45.8, 47.4]
(49, 50.6]
(52.2, 53.8]
(55.4, 57]
(58.6, 60.2]
(61.8, 63.4]
(65, 66.6]
(68.2, 69.8]
(71.4, 73]
(74.6, 76.2]
(77.8, 79.4]
(81, 82.6]
(84.2, 85.8]
(87.4, 89]
(90.6, 92.2]
(93.8, 95.4]
(97, 98.6]

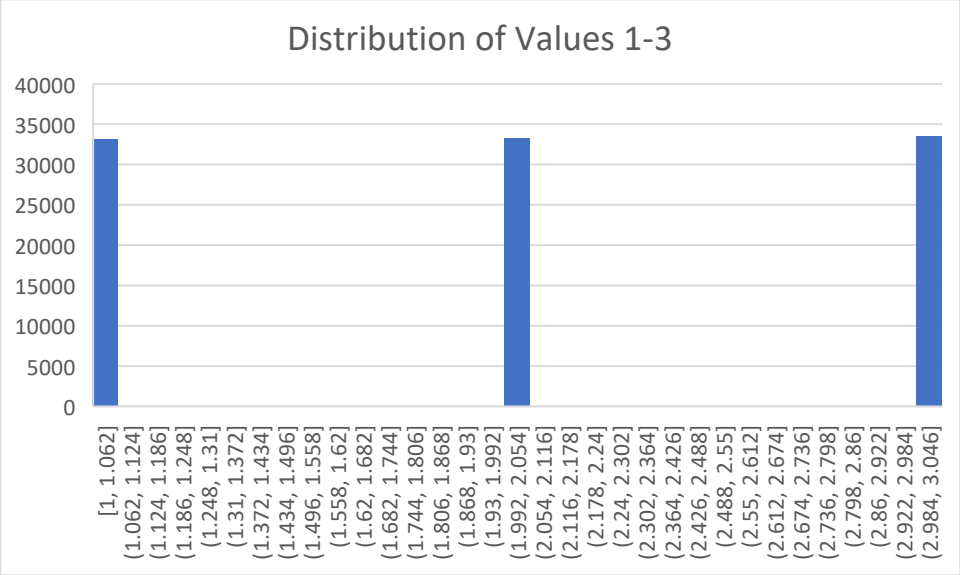
Code:

```
std::vector<int>* skewed_distribution(int m, std::default_random_engine generator){
    std::vector<int> *dist = new std::vector<int>();
    int val = 1;
    //create skewed distribution by adding numbers with skewed num of occurences
    for(int i = m; i > 0; i--){
        for(int j = 0; j < i; j++)
            dist->push_back(val);
        val++;
    }

    //shuffle distribution
    std::shuffle(dist->begin(),dist->end(), generator);
    return dist;
}

long long problem2(long n, long m, int p, std::default_random_engine generator, bool
printDist = false){
    using namespace std::chrono;

    int arr[p * n];
    int iter = 0;
    std::vector<int>* dist_skewed = skewed_distribution(m, generator); //create
skewed distribution
    std::uniform_int_distribution<long int> rand(0,dist_skewed->size()-1); //rand #
generator for skewed dist selection

    //measure starting time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    //loop p iterations
    for(int i = 0; i < p; i++){
        int uniqueNums = 0;
        int setIter = iter;

        //loop until n unique numbers are generated
        while(uniqueNums < n){
            int x = dist_skewed->at(rand(generator)); //select a random # from
skewed distribution

            if(isUnique(x, arr, setIter, setIter+n-1)){
                arr[iter++] = x;
                uniqueNums++;
            }
        }
    }

    //measure total time
    high_resolution_clock::time_point end = high_resolution_clock::now();
    auto total_time = duration_cast<nanoseconds>(end - start).count();
    std::cout << "p: " << p << " n: " << n << " m: " << m << " time: " << total_time
<< std::endl;

    if(printDist){
        //store distribution of numbers to file
        std::ofstream fout("problem2_dist.csv");
        for(int i = 0; i < n*p; i++)
            fout << arr[i] << ",\n";
        fout.close();
    }

    return total_time;
}
```

Problem 3:

- **Asymptotic Bounds:** $\boldsymbol{\theta(n^2)}$
- **Support:** From the table, we can conclude that this function is bounded by $\theta(n^2)$. We see this when we increase *n* by a factor of 10, *t* increases by a factor of 100, or n^2.

Sorting N Numbers with Bubble Sort	
n	t (nanoseconds)
10	1715
100	29587
1000	1977750
10000	322396507
100000	2.4608E+10



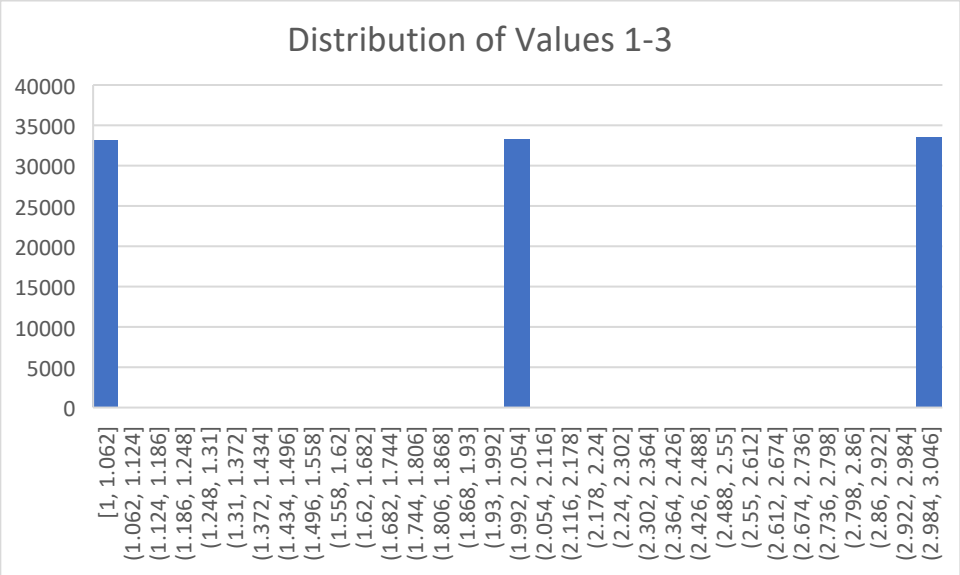
Code:

```
//Adapted from https://www.geeksforgeeks.org/bubble-sort/
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

//Adapted from https://www.geeksforgeeks.org/bubble-sort/
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)

        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}

long long problem3(long n, std::default_random_engine generator, bool printDist =
false){
    using namespace std::chrono;

    int arr[n];
    std::uniform_int_distribution<long int> dist(1,3);

    //measure starting time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    //insert n random numbers between 1 & 3
    for(int i = 0; i < n; i ++)
        arr[i] = dist(generator);

    //sort array using bubble sort
    bubbleSort(arr, n);

    //measure total time
    high_resolution_clock::time_point end = high_resolution_clock::now();
    auto total_time = duration_cast<nanoseconds>(end - start).count();
    std::cout << "n: " << n << " time: " << total_time << std::endl;

    if(printDist){
        //store distribution of numbers to file
        std::ofstream fout("problem3_dist.csv");
        for(int i = 0; i < n; i++)
            fout << arr[i] << ",\n";
        fout.close();
    }


    return total_time;

}
```

Problem 4:

- **Asymptotic Bounds:** $\boldsymbol{\theta(nlgn)}$
- **Support:** We can see from the table that the function is bounded by *n* whenever we increase *n* by a factor of 10, the value of *t* will also increase by a factor of 10. The *logn* comes from the functionality of merge sort. Merge sort continuously divides the array into two until the split array is of size 1. Once completely divided, the divided arrays are combined in sorted order. This division explains the *logn* portion of the asymptotic bounds, thus concluding with $\theta(nlgn)$.

Sorting N Numbers with Merge Sort	
n	t
10	2027
100	18369
1000	179661
10000	2709500
100000	18397437



Code:

```
//Adapted from https://www.geeksforgeeks.org/merge-sort/
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
       are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

```cpp
//Adapted from https://www.geeksforgeeks.org/merge-sort/
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}

long long problem4(int n, std::default_random_engine generator, bool printDist){
    using namespace std::chrono;

    int arr[n];
    std::uniform_int_distribution<long int> dist(1,3);

    //measure starting time
    high_resolution_clock::time_point start = high_resolution_clock::now();

    //insert n random numbers between 1 & 3
    for(int i = 0; i < n; i ++)
        arr[i] = dist(generator);

    //sort array using bubble sort
    mergeSort(arr, 0, n-1);

    //measure total time
    high_resolution_clock::time_point end = high_resolution_clock::now();
    auto total_time = duration_cast<nanoseconds>(end - start).count();
    std::cout << "n: " << n << " time: " << total_time << std::endl;

    if(printDist){
        //store distribution of numbers to file
        std::ofstream fout("problem3_dist.csv");
        for(int i = 0; i < n; i++)
            fout << arr[i] << ",\n";
        fout.close();
    }


    return total_time;
}
```

Main.cpp

```
int main() {

    //For each value, total numbers generated is p x n

    int p[] = {100, 1000, 10000};
    int n[] = {5, 10, 15};
    int m[] = {20, 100, 500};

    std::random_device rd;
    std::default_random_engine generator(rd());

    long time_1;
    long time_2;
    long time_3;
    long time_4;

    std::ofstream fout("problem1_data.csv");
    std::ofstream fout2("problem2_data.csv");

    //Run values for problems 1-2
    fout << "p,n,m,t\n";
    fout2 << "p,n,m,t\n";
    bool printDist = false;
    int iter = 0;
    for(int p_i : p){
        for(int n_i : n){
            for(int m_i : m){
                if(m_i == 100 && n_i == 15 && p_i == 10000)
                    printDist = true;
                time_1 = problem1(n_i,m_i,p_i, generator, printDist);
                time_2 = problem2(n_i,m_i,p_i, generator, printDist);

                fout << p_i << "," << n_i << "," << m_i << "," << time_1 << "\n";
                fout2 << p_i << "," << n_i << "," << m_i << "," << time_2 << "\n";

                printDist = false;
            }
        }
    }
```

Continued on next page

```
    fout.close();
    fout2.close();

    std::ofstream fout3("problem3_data.csv");
    std::ofstream fout4("problem4_data.csv");

    fout3 << "n,t\n";
    fout4 << "n,t\n";

    //run values for problems 3-4
    int n2[] = {10, 100, 1000, 10000, 100000};

    printDist = false;
    for(int i = 0; i < 5; i++){
        if(i == 4)
            printDist = true;

        time_3 = problem3(n2[i], generator, printDist);
        time_4 = problem4(n2[i], generator, printDist);

        fout3 << n2[i] <<  "," << time_3 << "\n";
        fout4 << n2[i] <<  "," << time_4 << "\n";
    }

    fout3.close();
    fout4.close();



    return 0;
```