**\*\*HELPER FUNCTIONS\*\***

CONTENTS:

1. SAVE FUNCTION (SAVES AN ARRAY TO A TEXT FILE).
2. LOAD FUNCTION (LOADS AN ARRAY FROM A TEXT FILE).
3. SWAP FUNCTION (SWAP TWO ARRAY ELEMENTS).
4. DISPLAY FUNCTION (DISPLAYS AN ARRAY).
5. RANDOM PERMUTATION ARRAY FUNCTION (GENERATES RANDOM ARRAY).

class helperFunctions{

public:

void Save(string Txt,int n)

{

helperFunctions H;

ofstream MyFile;

MyFile.open(Txt);

int \*Arr=new int[n];

H.RPA(Arr,n);

for(int i=0;i<n;i++)

{

MyFile<<Arr[i]<<endl;

}

MyFile.close();

delete [] Arr;

}

void Load(int \*Arr,string Txt,int n)

{

ifstream MyFile;

MyFile.open(Txt);

for(int i=0;i<n;i++)

{

MyFile>>Arr[i];

}

MyFile.close();

}

void Swap(int &x,int &y)

{

int Temp=x;

x=y;

y=Temp;

}

void Print(int \*a,int n)

{

for(int i=0;i<n;i++)

{

cout<<a[i]<<endl;

}

}

void RPA(int \*a,int n)

{

for(int i=0;i<n;i++)

{

a[i]=i;

}

srand(time(0));

for(int i=2;i<n;i++)

{

int m= rand() % i + 1;

Swap(a[i],a[m]);

}

}

};

**\*\*1) SELECTION SORT\*\***

1. IMPLEMENTATION:

int SelectionSort(int arr[],int n)

{

int count=0;

helperFunctions H;

for(int i=1;i<n-1;i++)

{

int x=i;

for(int j=i+1;j<n;j++)

{

if(arr[j]<arr[x])

{

x=j;

}

count++;

}

H.Swap(arr[i],arr[x]);

}

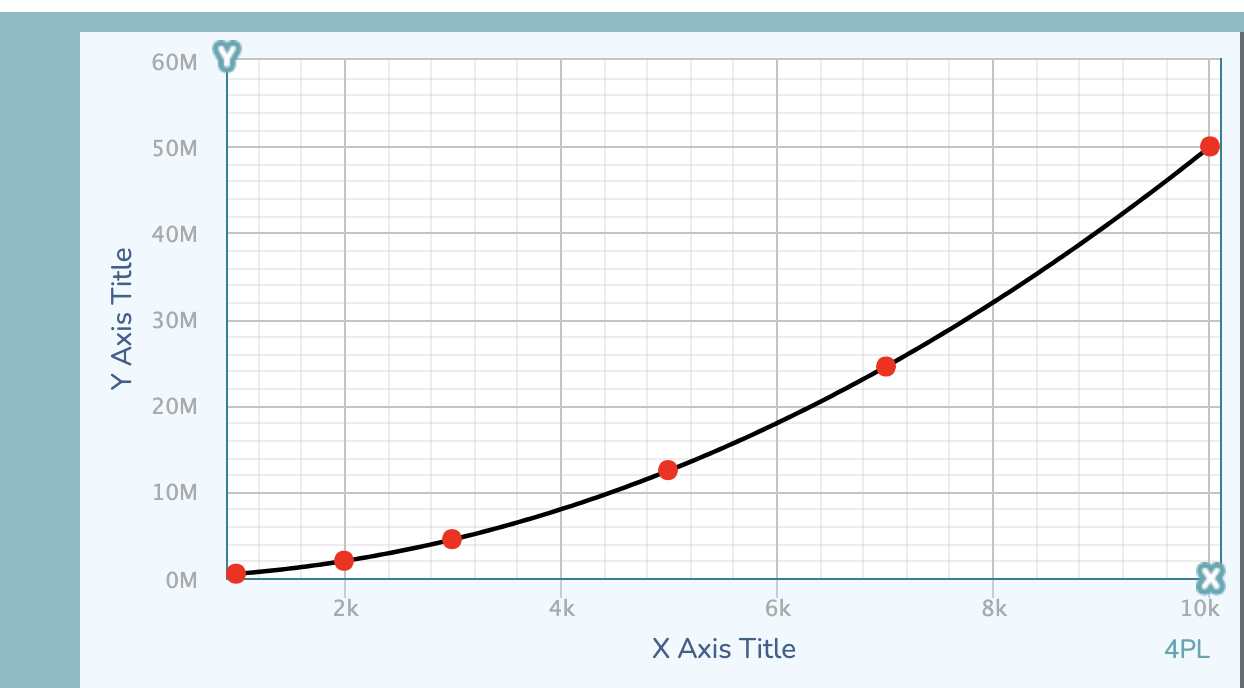
return count;

}

1. RESULTS:

| N | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 |
| --- | --- | --- | --- | --- | --- | --- |
| T(N) Experiment | 499500 | 1999000 | 4498500 | 12497500 | 24496500 | 49995000 |
| T(N) Model | 499500 | 1999000 | 4498500 | 12497500 | 24496500 | 49995000 |

1. T(N) MODEL EQUATION: MODEL = N(N-1)/2 (GIVEN EQUATION)
2. GRAPH REPRESENTATION:

****

**\*\*2) INSERTION SORT\*\***

1. INSERTION SORT FUNCTION:

void insertion(int \*a, int size, int& count){

int i, j, v;

for (i = 1; i < size; i++)

{

v = a[i];

j = i ;

while (j > 0 && (count++,a[j-1] > v))

{

a[j] = a[j-1];

j--;

}

a[j] = v;

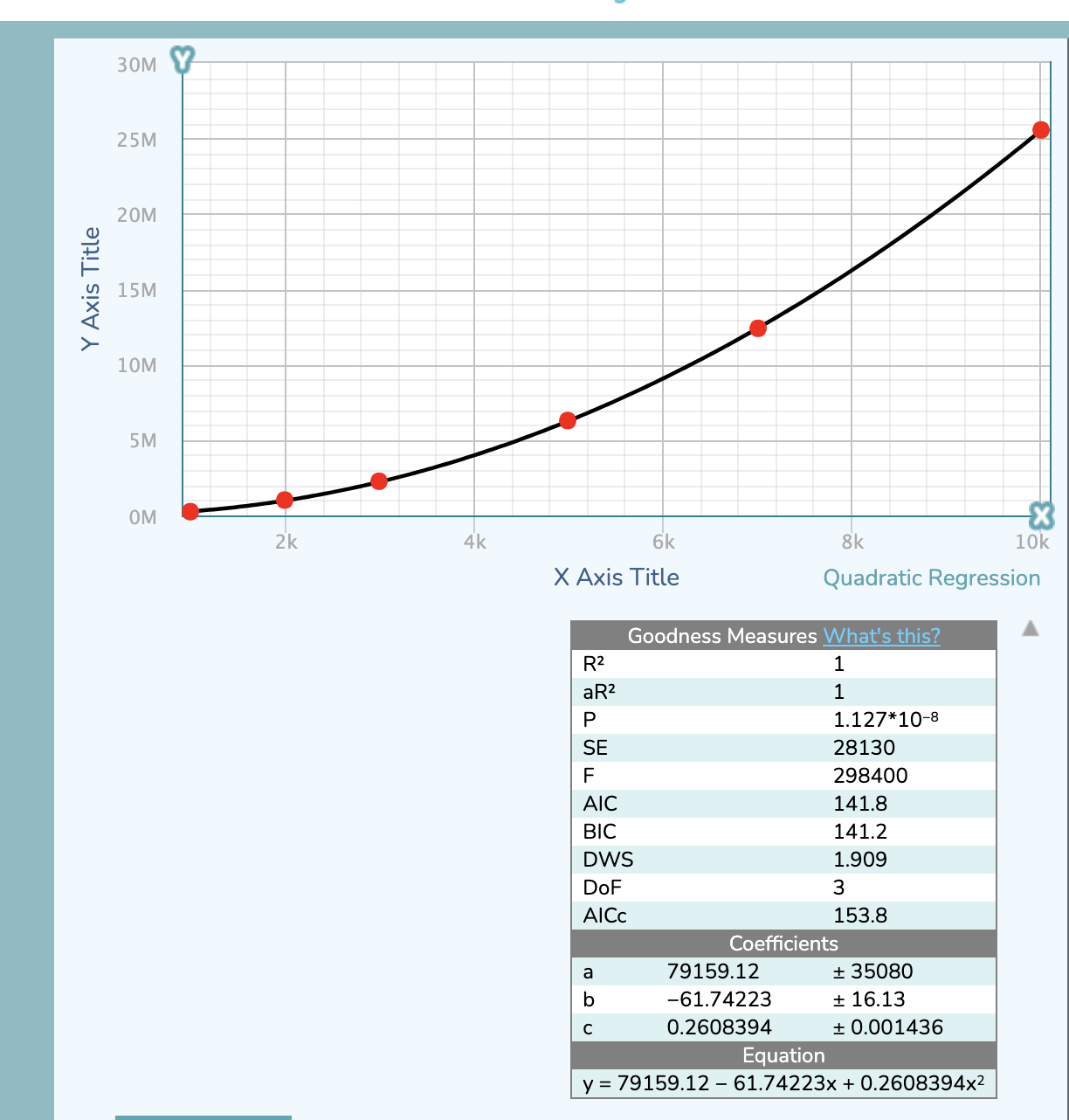
}

}

1. RESULTS:

| N | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 |
| --- | --- | --- | --- | --- | --- | --- |
| T(N) Experiment | 251473 | 1019425 | 2265121 | 6288313 | 12404111 | 25555527 |
| T(N) Model | 278256 | 999032 | 2241487 | 6291432 | 12428094 | 25545676 |

1. T(N) MODEL EQUATION: MODEL = 79159.12 - 61.74223\*N + 0.2608394\*N\*N
2. GRAPH REPRESENTATION:

****

**\*\*3) MERGE SORT\*\***

1. IMPLEMENTATION:

class MergeSort{

public:

void Merge(int \*Arr,int left, int mid, int right,int &Counter)

{

int i, j, k;int n1 = mid - left + 1;int n2 = right - mid;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = Arr[left + i];

for (j = 0; j < n2; j++)

R[j] = Arr[mid + 1 + j];

i = 0;j = 0;k = left;

while (i < n1 && j < n2) {

Counter++;

if (L[i] <= R[j]) {

Arr[k] = L[i];

i++;

}

else {

Arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

Arr[k] = L[i];i++;k++;

}

while (j < n2) {

Arr[k] = R[j];j++;k++;

}

}

void mergeSort(int \*Arr,int left,int right,int &Counter)

{

if (left < right)

{

int mid = left + (right - left) / 2;

mergeSort(Arr,left, mid,Counter);

mergeSort(Arr,mid + 1, right,Counter);

Merge(Arr,left, mid, right,Counter);

}

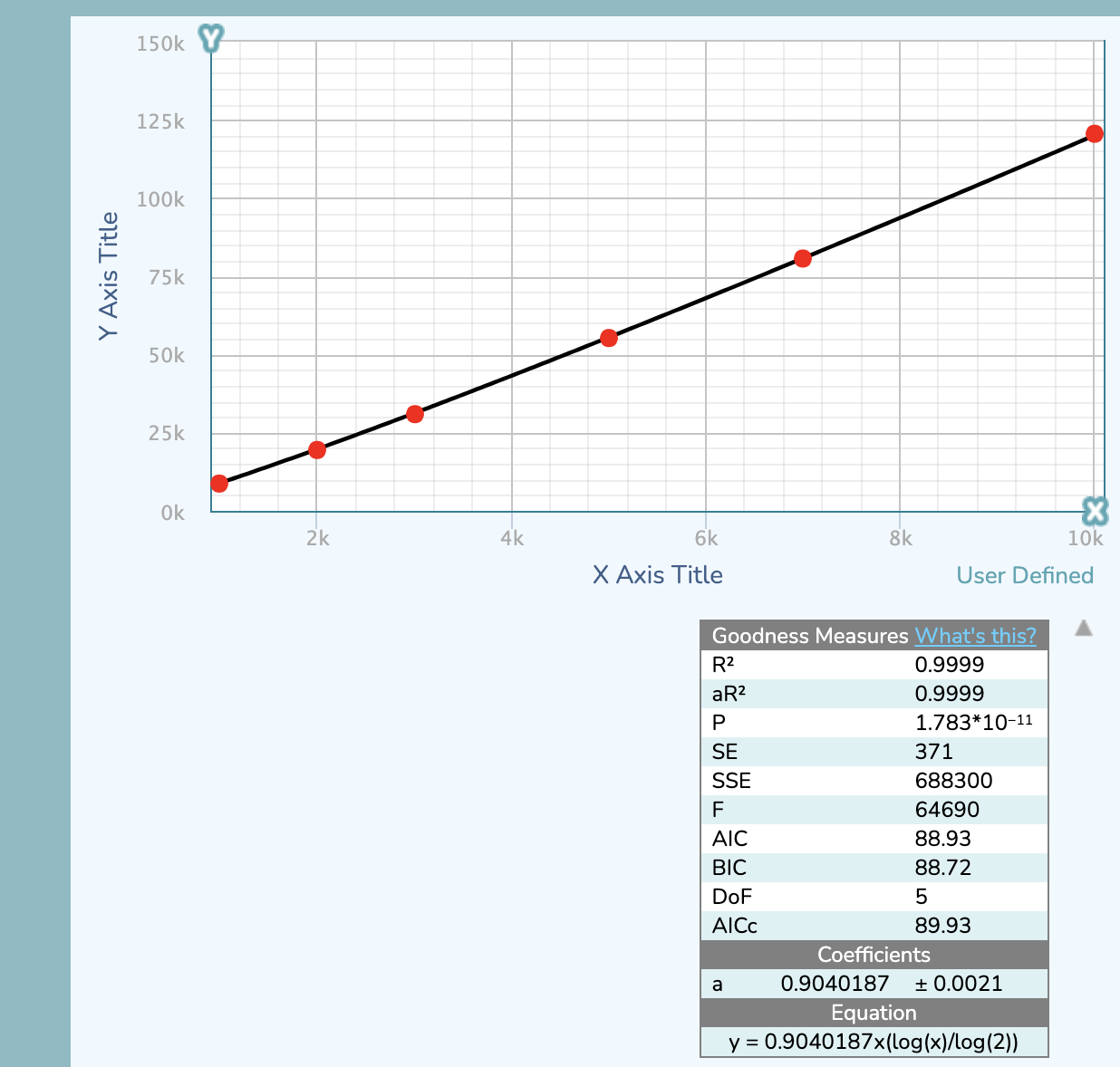
}

};

1. RESULTS:

| N | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 |
| --- | --- | --- | --- | --- | --- | --- |
| T(N) Experiment | 8720 | 19461 | 30913 | 55251 | 80666 | 120558 |
| T(N) Model | 9009 | 19826 | 31326 | 55541 | 80830 | 120123 |

1. T(N) MODEL EQUATION: MODEL **=** 0.9040187\*N \* log2(N)
2. GRAPH REPRESENTATION:

****

**\*\*4) QUICK SORT\*\***

1. IMPLEMENTATION:

class QuickSort{

private:

helperFunctions H;

int partition(int arr[], int low, int high,int &Counter)

{

int pivot = arr[low];

int k = high;

for (int i = high; i > low; i--) {

if (arr[i] > pivot)

{

H.Swap(arr[i], arr[k--]);

}

Counter++;

}

H.Swap(arr[low], arr[k]);

return k;

}

public:

void quickSort(int arr[], int low, int high,int &Counter)

{

if (low < high) {

int i = partition(arr, low, high,Counter);

quickSort(arr, low, i - 1,Counter);

quickSort(arr, i + 1, high,Counter);

}

}

void rQuickSort(int \*arr,int low,int high,int &Counter)

{

if(low<high)

{

if((high-low)>5)

{

int k=rand()%(high-low+1)+low;

H.Swap(arr[low],arr[k]);

}

int i = partition(arr, low, high,Counter);

rQuickSort(arr, low, i-1,Counter);

rQuickSort(arr, i+1, high,Counter);

}

}

};

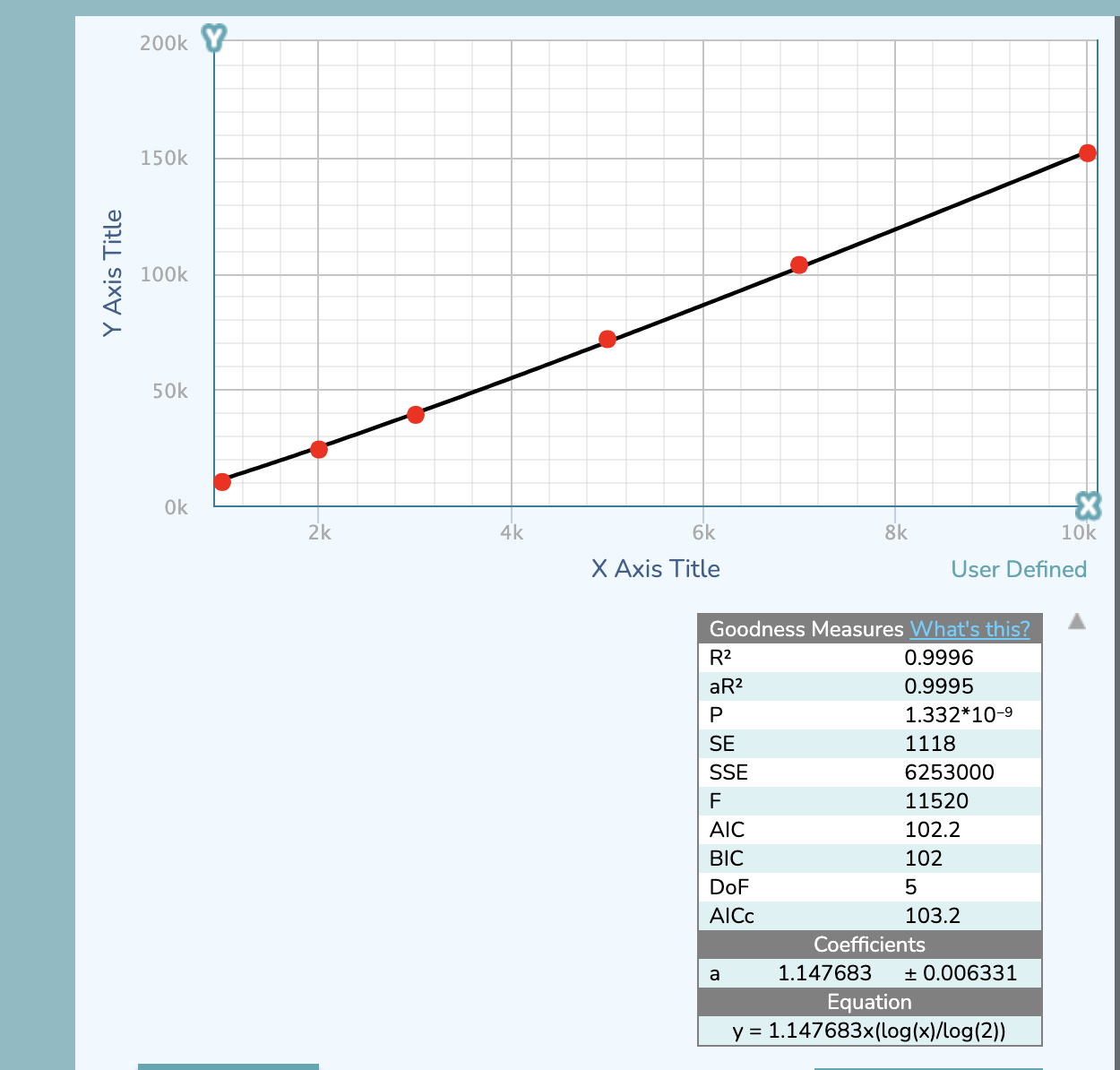
1. RESULTS:

CASE : D : Pivot is the first element in the array :

* In this case we will be using the member function quickSort().

| N | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 |
| --- | --- | --- | --- | --- | --- | --- |
| T(N) Experiment | 10094 | 24091 | 39034 | 71637 | 103601 | 151789 |
| T(N) Model | 11437 | 25170 | 39769 | 70511 | 102616 | 152500 |

* T(N) MODEL EQUATION: MODEL = 1.147683\*N \* log2(N)
* GRAPH REPRESENTATION:

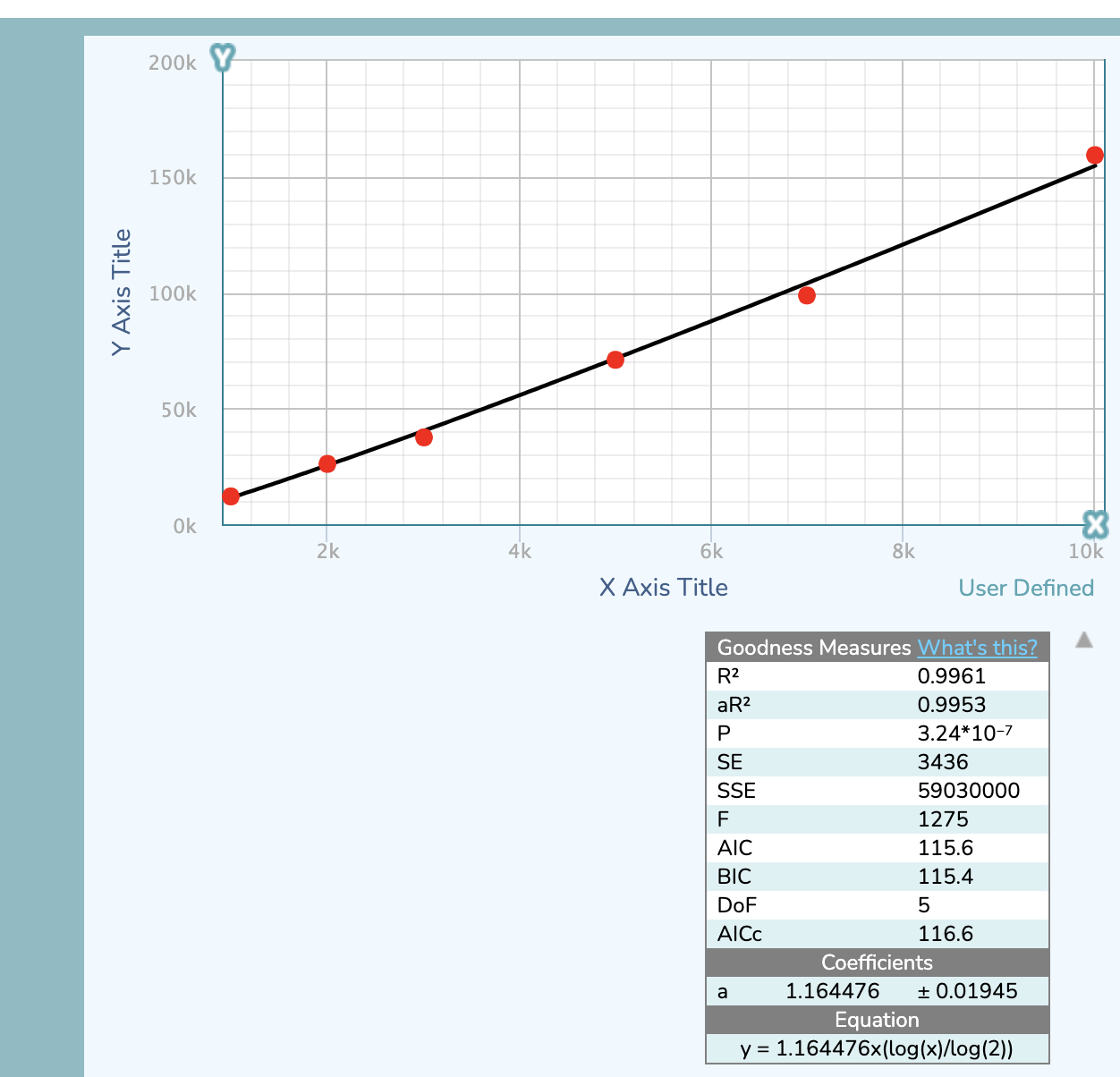


CASE : E : Pivot is chosen randomly :

* In this case we will be using the member function rQuickSort().

| N | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 |
| --- | --- | --- | --- | --- | --- | --- |
| T(N) Experiment | 11983 | 26015 | 37468 | 70951 | 98712 | 159289 |
| T(N) Model | 11604 | 25538 | 40351 | 71543 | 104118 | 154732 |

* T(N) MODEL EQUATION: MODEL = 1.164476\*N \* log2(N)
* GRAPH REPRESENTATION:

****