5.b

int main()

{

std::srand(time(NULL));

int arr[] = { 8, 4, 5, 1, 6, 7, 3, 2, 9, 20, 11 };

int n = sizeof(arr) / sizeof(arr[0]);

double twoThree = 2.0 / 3.0;

double threeFour = 3.0 / 4.0;

int count = 7;

int arrSize = 10;

std::cout << "alpha value: " << twoThree << std::endl;

while (count >= 1) {

int\* arrSort = new int[arrSize];

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

{

arrSort[i] = rand() % arrSize;

}

auto start = high\_resolution\_clock::now();

stoogesort(arrSort, 0, arrSize - 1, twoThree);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Array size: " << arrSize << " / time taken by function: " << duration.count() << " microseconds" << std::endl;

arrSize = arrSize \* 2;

count--;

}

count = 7;

arrSize = 10;

std::cout << "alpha value: " << threeFour << std::endl;

while (count >= 1) {

int\* arrSort = new int[arrSize];

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

{

arrSort[i] = rand() % arrSize;

}

auto start = high\_resolution\_clock::now();

stoogesort(arrSort, 0, arrSize - 1, threeFour);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

std::cout << "Array size: " << arrSize << " / time taken by function: " << duration.count() << " microseconds" << std::endl;

arrSize = arrSize \* 2;

count--;

}

**5.c**



For the alpha value of 2/3, it fit better with polynomial curve at degree of 3. I guess the theoretical t(n) = n^2.71 is close to n^3 rather than n^2. That is why it has better r square value of 0.9995 with polynomial curve at degree of 3

**-----------------**



For the alpha value of 3/4, it fit better with polynomial curve at degree of 4. I guess the theoretical t(n) = n^4.18. That is why it has better r square value of 1 with polynomial curve at degree of 4

**5.d**

I plotted the chart with array size from 10, and it grows by 2times. As I expected, badsort algorithm with alpha value of 2/3 had shown better performance than alpha value of 3/4.

