During the implementation and execution of the linear and quadratic algorithms of this lab, we yielded results in line with our theoretical expectations. Given that the number of actions taken by the quadratic algorithm scales quadratically with a greater number of elements, and that the actions by the linear algorithm would scale linearly, the gap in between the execution time widening by an increasing factor (fig. 3) is to be expected, with the gap beginning with a factor of 676 (i.e. linear time times 676 == quadratic time) and widening to a factor of 7309. The variability of this factor exemplifies the difference in running times between Big-Oh linear and quadratic notations. In our experiment, we increased the array sizes by 10,000 each execution and as is shown by our trendline, the linear algorithm had an execution time 676x faster than our quadratic algorithm, which goes to show the importance of a more efficient algorithm when scaling.

C:\Users\zacal\OneDrive\Desktop>a.exe

Linear:

Value of 0: 228 Average: 228

Value of 1: 1000 Average: 614

Value of 2: 70 Average: 432

Value of 3: 361 Average: 414

Value of 4: 523 Average: 436

Execution for array of size 10000: 207

Value of 0: 115 Average: 115

Value of 1: 24 Average: 69

Value of 2: 649 Average: 262

Value of 3: 74 Average: 215

Value of 4: 371 Average: 246

Execution for array of size 20000: 413

Value of 0: 3 Average: 3

Value of 1: 280 Average: 141

Value of 2: 995 Average: 426

Value of 3: 19 Average: 324

Value of 4: 451 Average: 349

Execution for array of size 30000: 556

Value of 0: 891 Average: 891

Value of 1: 304 Average: 597

Value of 2: 574 Average: 589

Value of 3: 733 Average: 625

Value of 4: 531 Average: 606

Execution for array of size 40000: 728

Value of 0: 779 Average: 779

Value of 1: 328 Average: 553

Value of 2: 153 Average: 420

Value of 3: 678 Average: 484

Value of 4: 612 Average: 510

Execution for array of size 50000: 940

Value of 0: 667 Average: 667

Value of 1: 351 Average: 509

Value of 2: 731 Average: 583

Value of 3: 392 Average: 535

Value of 4: 460 Average: 520

Execution for array of size 60000: 1062

Value of 0: 554 Average: 554

Value of 1: 375 Average: 464

Value of 2: 78 Average: 335

Value of 3: 337 Average: 336

Value of 4: 540 Average: 376

Execution for array of size 70000: 1285

Value of 0: 442 Average: 442

Value of 1: 399 Average: 420

Value of 2: 657 Average: 499

Value of 3: 282 Average: 445

Value of 4: 620 Average: 480

Execution for array of size 80000: 1651

Value of 0: 330 Average: 330

Value of 1: 423 Average: 376

Value of 2: 235 Average: 329

Value of 3: 996 Average: 496

Value of 4: 700 Average: 536

Execution for array of size 90000: 1621

Value of 0: 218 Average: 218

Value of 1: 679 Average: 448

Value of 2: 582 Average: 493

Value of 3: 941 Average: 605

Value of 4: 548 Average: 593

Execution for array of size 100000: 1914

Quadratic:

Value of 0: 228 Average: 663

Value of 1: 1000 Average: 445.5

Value of 2: 70 Average: 706.333

Value of 3: 361 Average: 854.25

Value of 4: 523 Average: 1015.2

Execution for array of size 10000: 139979

Value of 0: 115 Average: 890

Value of 1: 24 Average: 502.5

Value of 2: 649 Average: 381.333

Value of 3: 74 Average: 483

Value of 4: 371 Average: 558.8

Execution for array of size 20000: 563089

Value of 0: 3 Average: 352

Value of 1: 280 Average: 177.5

Value of 2: 995 Average: 212.667

Value of 3: 19 Average: 479

Value of 4: 451 Average: 642.6

Execution for array of size 30000: 1259032

Value of 0: 897 Average: 89

Value of 1: 801 Average: 493

Value of 2: 534 Average: 894.667

Value of 3: 323 Average: 1229

Value of 4: 160 Average: 1494.2

Execution for array of size 40000: 2243449

Value of 0: 792 Average: 498

Value of 1: 553 Average: 645

Value of 2: 73 Average: 878.333

Value of 3: 627 Average: 1013.25

Value of 4: 102 Average: 1219.6

Execution for array of size 50000: 3494404

Value of 0: 689 Average: 499

Value of 1: 54 Average: 594

Value of 2: 477 Average: 643.667

Value of 3: 227 Average: 787.75

Value of 4: 358 Average: 919.6

Execution for array of size 60000: 5033551

Value of 0: 594 Average: 498

Value of 1: 284 Average: 546

Value of 2: 72 Average: 656.667

Value of 3: 185 Average: 730

Value of 4: 243 Average: 811

Execution for array of size 70000: 6873723

Value of 0: 504 Average: 499

Value of 1: 243 Average: 501.5

Value of 2: 396 Average: 583.333

Value of 3: 501 Average: 723.25

Value of 4: 989 Average: 907.4

Execution for array of size 80000: 8968267

Value of 0: 421 Average: 498

Value of 1: 698 Average: 459.5

Value of 2: 912 Average: 679.333

Value of 3: 640 Average: 1017.25

Value of 4: 364 Average: 1348

Execution for array of size 90000: 11340877

Value of 0: 348 Average: 0

Value of 1: 631 Average: 174

Value of 2: 253 Average: 442.333

Value of 3: 202 Average: 639.75

Value of 4: 147 Average: 798.6

Execution for array of size 100000: 13990528

#include <iostream>

#include <stdlib.h>

#include <chrono>

#include <ctime>

typedef std::chrono::high\_resolution\_clock Clock;

void quadratic\_prefix\_average(float\* dbl\_arr,int array\_size,float\* ret\_arr);

void linear\_prefix\_average(float\* dbl\_arr,int array\_size,float\* ret\_arr);

int main(){

int array\_size = 10000;

std::cout << "\nLinear:\n";

do{

float arr[array\_size];

float arr2[array\_size];

auto start\_time = Clock::now();

srand(time(NULL)+array\_size);

for (int i=0;i<array\_size;i++){

arr[i] = (rand() % 1000)+1;

}

linear\_prefix\_average(arr,array\_size,arr2);

auto end\_time = Clock::now();

for (int i=0;i<5;i++){

std::cout << "\nValue of " << i << ": " << arr[i] << " Average: " << arr2[i] << std::endl;

}

std::cout << "\nExecution for array of size " << array\_size << ": " <<

std::chrono::duration\_cast<std::chrono::microseconds>(end\_time-start\_time).count() <<

std::endl;

array\_size += 10000;

}while (array\_size <= 100000);

std::cout << "\nQuadratic:\n";

array\_size = 10000;

do{

float arr[array\_size];

float arr2[array\_size];

auto start\_time = Clock::now();

srand(time(NULL)+array\_size);

for (int i=0;i<array\_size;i++){

arr[i] = (rand() % 1000)+1;

}

quadratic\_prefix\_average(arr,array\_size,arr2);

auto end\_time = Clock::now();

for (int i=0;i<5;i++){

std::cout << "\nValue of " << i << ": " << arr[i] << " Average: " << arr2[i] << std::endl;

}

std::cout << "\nExecution for array of size " << array\_size << ": " <<

std::chrono::duration\_cast<std::chrono::microseconds>(end\_time-start\_time).count() <<

std::endl;

array\_size += 10000;

}while (array\_size <= 100000);

return 0;

}

void linear\_prefix\_average(float\* dbl\_arr, int array\_size, float\* ret\_arr){

int a = 0;

for (int i=0;i<array\_size;i++){

a += dbl\_arr[i];

ret\_arr[i] = a/(i+1);

}

}

void quadratic\_prefix\_average(float\* dbl\_arr, int array\_size, float\* ret\_arr){

float a;

for (int i = 0; i <array\_size ; i++)

{

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

{

a += dbl\_arr[j];

}

ret\_arr[i] = a/(i+1);

}

}