Operating Systems Lab

CPE 435-01

Lab 09: Performance Analysis

By: David Thornton

Lab Date: 8 March 2021

Lab Due: 15 March 2021

Demonstration Due: 15 March 2021

Introduction

The purpose of this lab is to give students an introduction to code testing and performance analysis as well as energy consumption. Specifically, the student will become familiar with Valgrind and a power consumption measurement tool depending on their operating system.

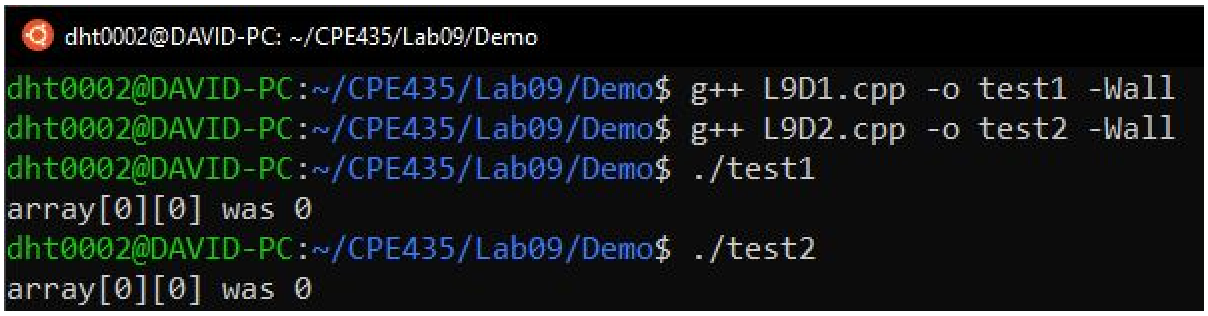
Observations

Part I: Valgrind

Part I.a: Cachegrind

- For machines with more than two levels of cache, cachegrind will simulate the machine as if it only has two levels of cache. It will simulate the first and third level caches in a machine with three levels, for example.

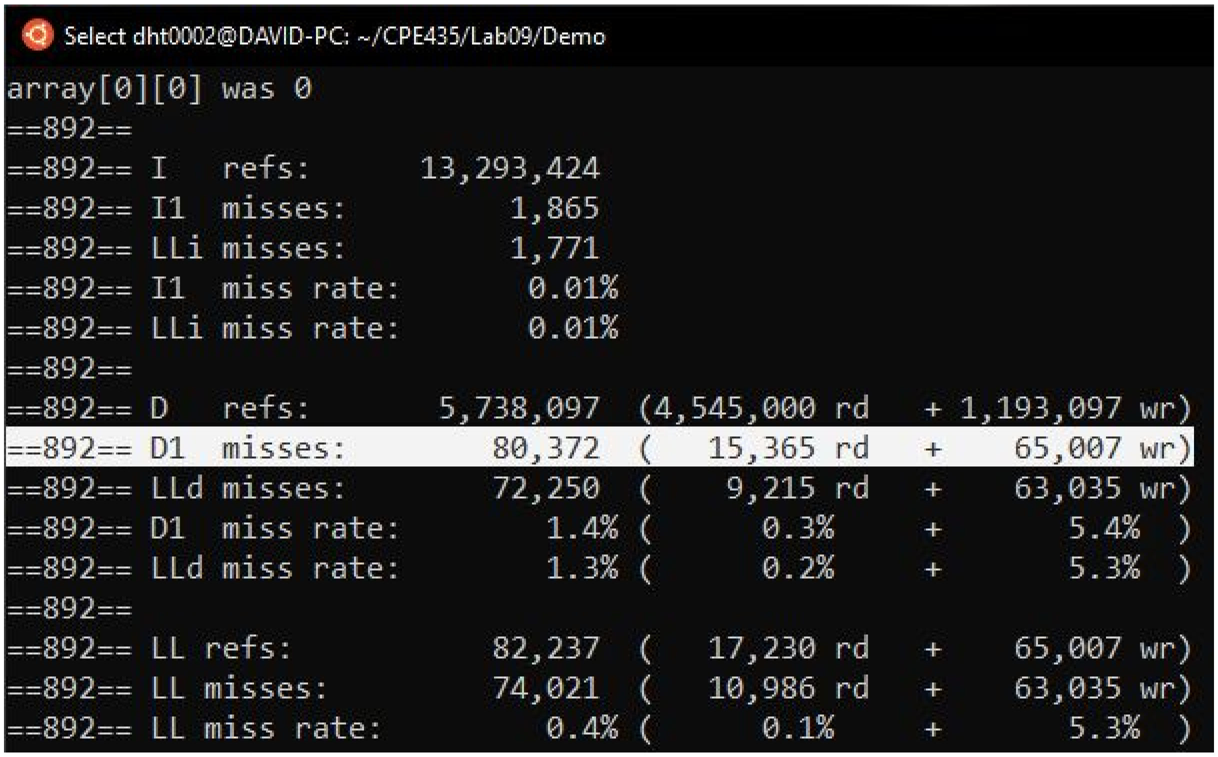
Assignment I



```
dht0002@DAVID-PC: ~/CPE435/Lab09/Demo
dht0002@DAVID-PC:~/CPE435/Lab09/Demo$ g++ L9D1.cpp -o test1 -Wall
dht0002@DAVID-PC:~/CPE435/Lab09/Demo$ g++ L9D2.cpp -o test2 -Wall
dht0002@DAVID-PC:~/CPE435/Lab09/Demo$ ./test1
array[0][0] was 0
dht0002@DAVID-PC:~/CPE435/Lab09/Demo$ ./test2
array[0][0] was 0
```

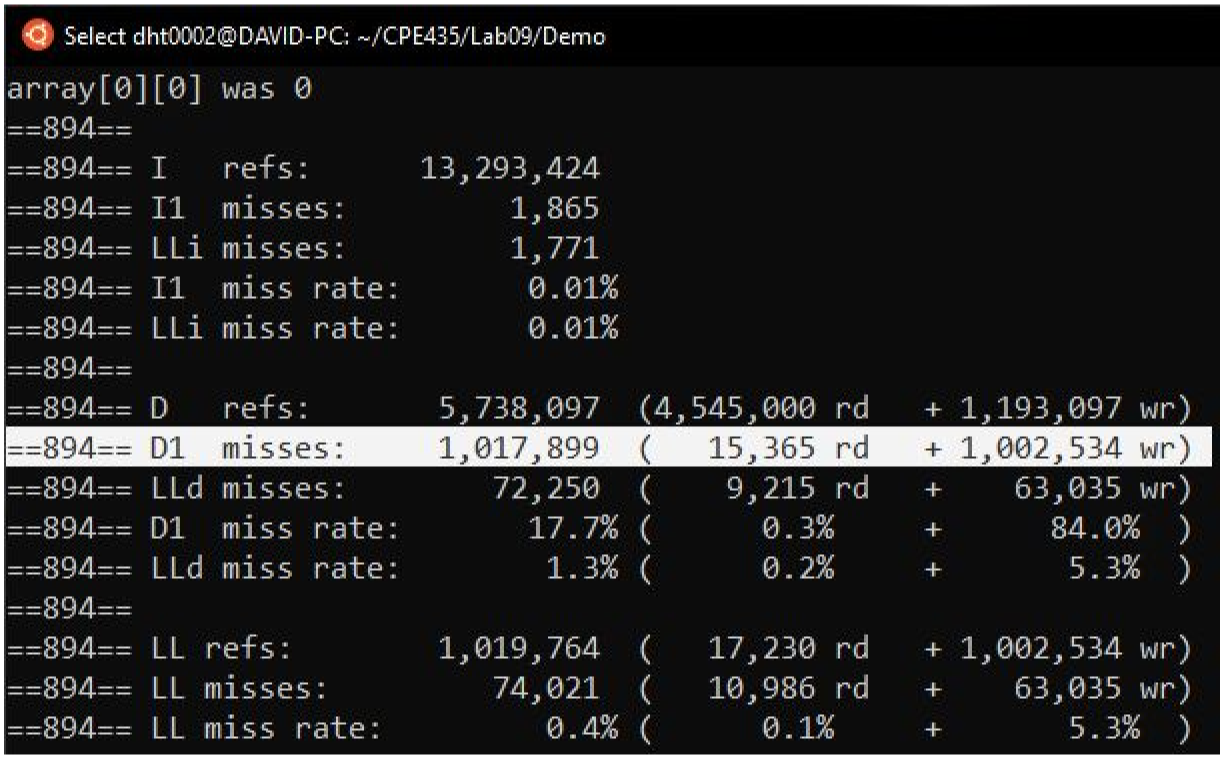
Figure 1: Compilation of test1 and test2

Assignment II



```
Select dht0002@DAVID-PC: ~/CPE435/Lab09/Demo
array[0][0] was 0
==892==
==892== I   refs:      13,293,424
==892== I1  misses:         1,865
==892== LLi misses:         1,771
==892== I1  miss rate:       0.01%
==892== LLi miss rate:       0.01%
==892==
==892== D   refs:       5,738,097  (4,545,000 rd   + 1,193,097 wr)
==892== D1  misses:        80,372  (   15,365 rd   +    65,007 wr)
==892== LLd misses:        72,250  (    9,215 rd   +    63,035 wr)
==892== D1  miss rate:        1.4% (      0.3%     +       5.4%  )
==892== LLd miss rate:        1.3% (      0.2%     +       5.3%  )
==892==
==892== LL refs:           82,237  (   17,230 rd   +    65,007 wr)
==892== LL misses:         74,021  (   10,986 rd   +    63,035 wr)
==892== LL miss rate:         0.4% (      0.1%     +       5.3%  )
```

Figure 2: valgrind - v -- tool = cachegrind ./test1



```
Select dht0002@DAVID-PC: ~/CPE435/Lab09/Demo
array[0][0] was 0
==894==
==894== I   refs:      13,293,424
==894== I1  misses:         1,865
==894== LLi misses:         1,771
==894== I1  miss rate:       0.01%
==894== LLi miss rate:       0.01%
==894==
==894== D   refs:       5,738,097  (4,545,000 rd   + 1,193,097 wr)
==894== D1  misses:     1,017,899  (   15,365 rd   + 1,002,534 wr)
==894== LLd misses:        72,250  (    9,215 rd   +    63,035 wr)
==894== D1  miss rate:       17.7% (      0.3%     +      84.0%  )
==894== LLd miss rate:        1.3% (      0.2%     +       5.3%  )
==894==
==894== LL refs:        1,019,764  (   17,230 rd   + 1,002,534 wr)
==894== LL misses:         74,021  (   10,986 rd   +    63,035 wr)
==894== LL miss rate:         0.4% (      0.1%     +       5.3%  )
```

Figure 3: valgrind - v -- tool = cachegrind ./test2

1. Which program is better in terms of performance?
    a. Test 1.
2. Why do you think one program is better?
    a. Test 1 is clearly the better program. With a D1 miss rate of 1.4% compared to 17.7%, it is a no brainer that test 1 will produce better, more reliable results.

Part I.b Memcheck

Assignment III

Test 3

- The array elements are not are not assigned a value.

Test 4

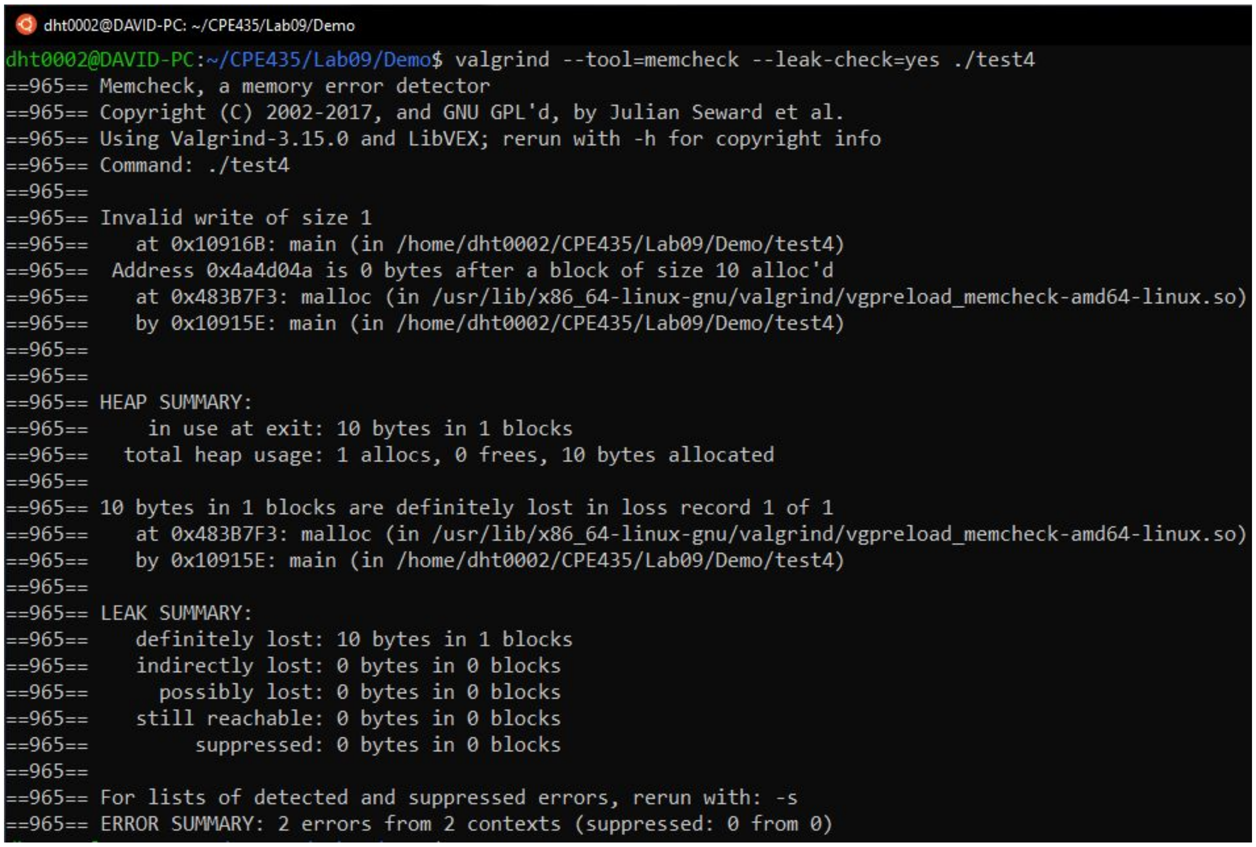
- The array is only allocated for 10 elements, indexes 0-9. Index 10 is out of bounds.

Test 5

- X is not initialized and will always result in the if statement to evaluate to true.

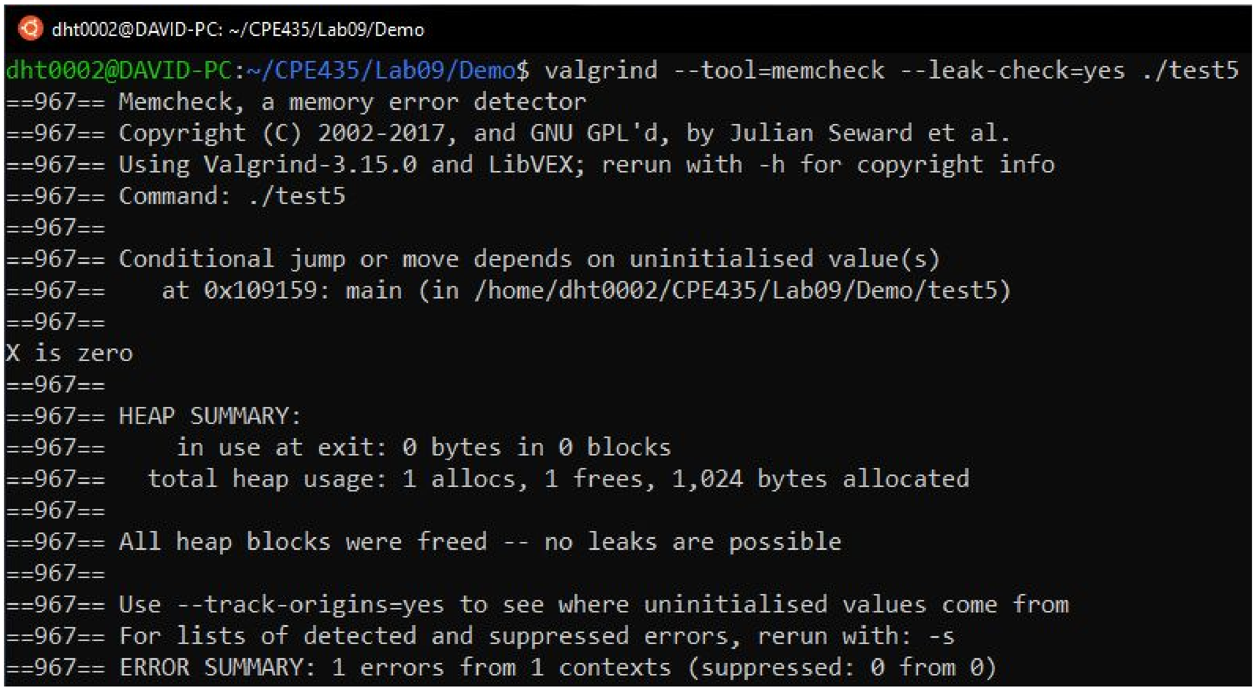
Assignment IV

Figure 4: valgrind --tool=memcheck --leak-check=yes ./test3



```
dht0002@DAVID-PC: ~/CPE435/Lab09/Demo
dht0002@DAVID-PC:~/CPE435/Lab09/Demo$ valgrind --tool=memcheck --leak-check=yes ./test4
==965== Memcheck, a memory error detector
==965== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==965== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==965== Command: ./test4
==965==
==965== Invalid write of size 1
==965==    at 0x10916B: main (in /home/dht0002/CPE435/Lab09/Demo/test4)
==965==  Address 0x4a4d04a is 0 bytes after a block of size 10 alloc'd
==965==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==965==    by 0x10915E: main (in /home/dht0002/CPE435/Lab09/Demo/test4)
==965==
==965==
==965== HEAP SUMMARY:
==965==     in use at exit: 10 bytes in 1 blocks
==965==   total heap usage: 1 allocs, 0 frees, 10 bytes allocated
==965==
==965== 10 bytes in 1 blocks are definitely lost in loss record 1 of 1
==965==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==965==    by 0x10915E: main (in /home/dht0002/CPE435/Lab09/Demo/test4)
==965==
==965== LEAK SUMMARY:
==965==    definitely lost: 10 bytes in 1 blocks
==965==    indirectly lost: 0 bytes in 0 blocks
==965==      possibly lost: 0 bytes in 0 blocks
==965==    still reachable: 0 bytes in 0 blocks
==965==         suppressed: 0 bytes in 0 blocks
==965==
==965== For lists of detected and suppressed errors, rerun with: -s
==965== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Figure 5: valgrind --tool=memcheck --leak-check=yes ./test4



```
dht0002@DAVID-PC: ~/CPE435/Lab09/Demo
dht0002@DAVID-PC:~/CPE435/Lab09/Demo$ valgrind --tool=memcheck --leak-check=yes ./test5
==967== Memcheck, a memory error detector
==967== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==967== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==967== Command: ./test5
==967==
==967== Conditional jump or move depends on uninitialised value(s)
==967==    at 0x109159: main (in /home/dht0002/CPE435/Lab09/Demo/test5)
==967==
X is zero
==967==
==967== HEAP SUMMARY:
==967==     in use at exit: 0 bytes in 0 blocks
==967==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==967==
==967== All heap blocks were freed -- no leaks are possible
==967==
==967== Use --track-origins=yes to see where uninitialised values come from
==967== For lists of detected and suppressed errors, rerun with: -s
==967== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Figure 6: valgrind --tool=memcheck --leak-check=yes ./test5

Test 3

- 100 bytes are allocated and never written to.

Test 4

- 10 bytes are allocated and never written to. Invalid write is due to writing to index 10.

Test 5

- Conditional jump/move depends on uninitialised value (variable x should be set before if)

Part II: Power Consumption Measurement - Using Hardware Monitor Pro (WSL2)

Assignment V

Quick Sort

- See appendix 1.

Merge Sort

- See appendix 2.

Insertion Sort

- See appendix 3.

Verification

- See appendices 4 and 5.

Assignment VI

Since energy is described by the equation $E = \int P \, dt$, we can simplify this equation to $E = P_{avg} * t$ for the scope and simplicity of this lab, and also to not give me a headache. I have attached the graphs for each of the runs generated by the tool (see appendices 6 - 15). Since I used sleep(3) in my program, this creates a nice separation between each of the sorts in the power graph, as the CPU is not under load for 3 seconds and returns to idle. The table below shows an extremely rough estimation of energy consumed for each sorting algorithm using the equation $E = P_{avg} * t$. The baseline power consumption has been subtracted out already.

	Run 1 (J)	Run 2 (J)	Run 3 (J)	Run 4 (J)	Run 5 (J)	Avg (J)
Insertion Sort	108	125	112	121	118	117
Merge Sort	123	111	124	114	126	120
Quick Sort	94	86	84	90	88	88

Table 1: Energy consumed by each sorting algorithm with optimization level 0.

Assignment VII

The table below shows an extremely rough estimation of energy consumed for each sorting algorithm using the equation $E = P_{avg} * t$. The baseline power consumption has been subtracted out already.

	Run 1 (J)	Run 2 (J)	Run 3 (J)	Run 4 (J)	Run 5 (J)	Avg (J)
Insertion Sort	24	33	29	34	37	31
Merge Sort	63	72	77	72	70	71
Quick Sort	36	38	44	47	42	41

Table 2: Energy consumed by each sorting algorithm with optimization level 3.

Assignment VIII

The best algorithm of the three measured in terms of energy consumption per element is quick sort. Changing the compiler flag to optimization level 3 did not change the relative order of the sorting algorithms. It did however significantly reduce the execution time of each sort.

	Avg -O0 (mJ/element)	Avg -O3 (mJ/element)
Insertion Sort	0.7	0.19
Merge Sort	0.002	0.001
Quick Sort	0.001	0.0005

Table 3: Energy consumption per element with different optimization levels.

Extra Credit Research

1. Why is the power consumption roughly the same (power, not energy) for the different algorithms?

   a. Each algorithm has roughly the same power draw due to the fact that they are executing similar machine instructions.

2. How would you determine if the hardware monitor is corrupting the power measurement?
   a. If the power measurements under heavy load exceed the thermal design power (TDP) rating, then either you have fried your CPU or the monitor is corrupted.

3. Please research and discuss the MSR registers.
   a. “A model-specific register (MSR) is any of various control registers in the x86 instruction set used for debugging, program execution tracing, computer performance monitoring, and toggling certain CPU features.” - Source.
   b. “Processors from the P6 family onwards (including PentiumPro, Pentium II, III, 4 and Intel Core) have a collection of registers that allow configuration of OS-relevant things such as memory type-range, sysenter/sysexit, local APIC, etc. These MSRs are accessed using special instructions such as RDMSR (Read MSR), WRMSR (Write MSR), and RDTSC.” - Source

Conclusion

This lab was successful in expanding my knowledge of memory allocation, code performance analysis, and energy consumption . Demo link

Appendix

Appendix 1: quick_sort.cpp

```
// ****************************************
// Program Title: Lab 09
// Project File: quick_sort.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 03/15/2021
// ****************************************
#include "sorts.h"

void quickRunner(int size, int arr[])
{
      time_t start = time(0);
      quickSort(arr, 0, size - 1);
      time_t end = time(0);
```

```cpp
      double time = difftime(end, start);
      cout << "Quick sort took " << time << " seconds." << endl;
}


void swap(int* a, int* b)
{
      int t = *a;
      *a = *b;
      *b = t;
}


int partition(int arr[], int low, int high)
{
      int pivot = arr[high];
      int i = (low - 1);
      for (int j = low; j <= high - 1; j++)
      {
            if (arr[j] < pivot)
            {
                  i++;
                  swap(&arr[i], &arr[j]);
            }
      }
      swap(&arr[i + 1], &arr[high]);
      return (i + 1);
}


void quickSort(int arr[], int low, int high)
{
      if (low < high)
      {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
      }
}
```

Appendix 2: merge_sort.cpp

```
// ****************************************
// Program Title: Lab 09
// Project File: merge_sort.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 03/15/2021
// ****************************************
#include "sorts.h"

void mergeRunner(int size, int arr[])
{
      time_t start = time(0);
      mergeSort(arr, 0, size - 1); // do the sort
      time_t end = time(0);
      double time = difftime(end, start);
      cout << "Merge sort took " << time << " seconds." << endl;
}


void mergeSort(int arr[], int l, int r)
{
      if (l >= r)
      {
            return;
      }
      int m = l + (r - l) / 2;
      mergeSort(arr, l, m);
      mergeSort(arr, m + 1, r);
      merge(arr, l, m, r);
}


void merge(int arr[], int l, int m, int r)
{
      int n1 = m - l + 1;
      int n2 = r - m;
      int* L = new int[n1];
      int* R = new int[n2];
      for (int i = 0; i < n1; i++)
      {
            L[i] = arr[l + i];
      }
```

```
	for (int j = 0; j < n2; j++)
	{
		R[j] = arr[m + 1 + j];
	}
	int i = 0;
	int j = 0;
	int k = l;
	while (i < n1 && j < n2)
	{
		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++;
	}
}
```

Appendix 3: insertion_sort.cpp

```
// ****************************************
// Program Title: Lab 09
// Project File: insertion_sort.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 03/15/2021
// *****************************************
#include "sorts.h"

void insertRunner(int size, int arr[])
{
	time_t start = time(0);
	insertionSort(arr, size); // do the sort
	time_t end = time(0);
	double time = difftime(end, start);
	cout << "Insertion sort took " << time << " seconds." << endl;
}


void insertionSort(int arr[], int n)
{
	int i, key, j;
	for (i = 1; i < n; i++)
	{
		key = arr[i];
		j = i - 1;
		while (j >= 0 && arr[j] > key)
		{
			arr[j + 1] = arr[j];
			j = j - 1;
		}
		arr[j + 1] = key;
	}
}
```

Appendix 4: verification.cpp

```
// *****************************************
// Program Title: Lab 09
// Project File: verification.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 03/15/2021
// *****************************************
#include "sorts.h"
```

```
using namespace std;

int main(int argc, char* argv[])
{
      int mergeArraySize = atoi(argv[1]); // # of ints to generate, default
50000000
      int insertArraySize = mergeArraySize / 300; // insert takes too long
      int quickArraySize = mergeArraySize * 1.5; // quick is too quick
      srand((unsigned int)time(NULL)); // seed for random number generation

      int* array1 = new int[insertArraySize]; // allocate heap memory
      for (int i = 0; i < insertArraySize; i++)
      {
            array1[i] = rand();
      }
      cout << "**********************************************" << endl;
      cout << "Starting insertion sort for "<< insertArraySize << "
elements" << endl;
      insertRunner(insertArraySize, array1);
      bool insertVerify = verify(insertArraySize, array1);
      if (!insertVerify)
      {
            cout << "Insertion sort failed!!!" << endl;
      }
      else if (insertVerify)
      {
            cout << "Insertion sort was successful." << endl;
      }
      cout << "**********************************************" << endl;
      delete[] array1;
      array1 = NULL;

      cout << "Starting merge sort for " << mergeArraySize << " elements"
<< endl;
      int* array2 = new int[mergeArraySize]; // allocate heap memory
      for (int i = 0; i < mergeArraySize; i++)
      {
            array2[i] = rand();
      }
      mergeRunner(mergeArraySize, array2);
      bool mergeVerify = verify(mergeArraySize, array2);
```

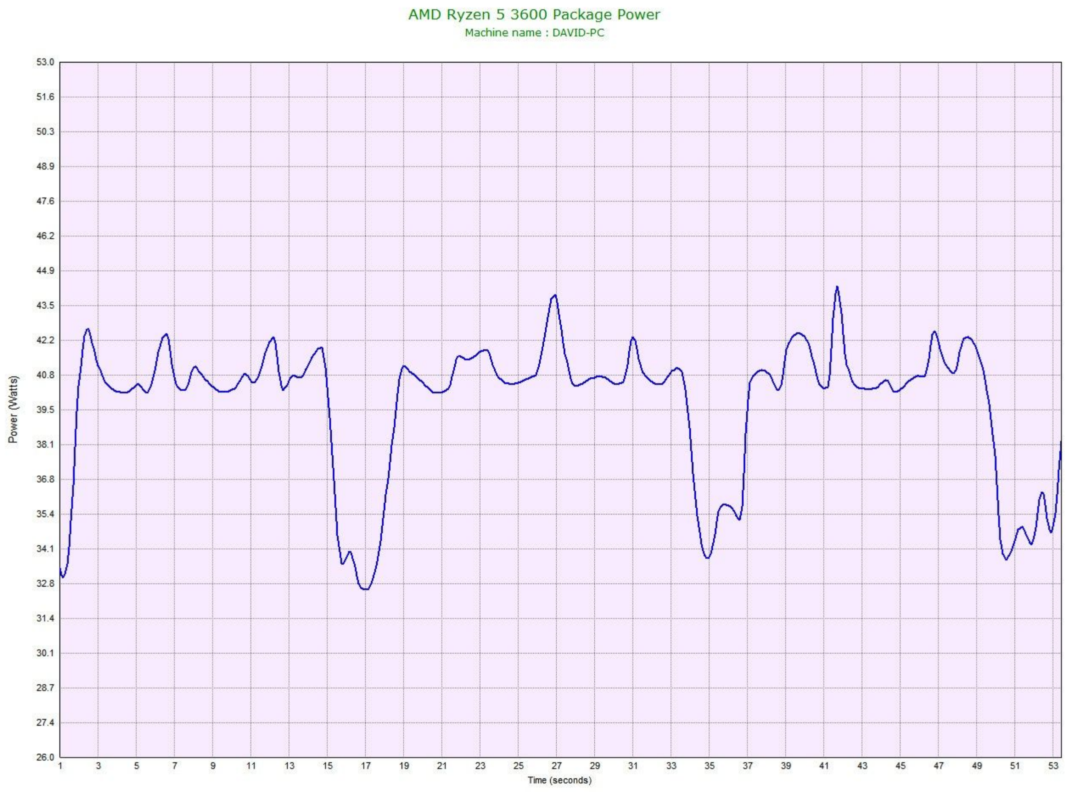
```cpp
	if (!mergeVerify)
	{
		cout << "Merge sort failed!!!" << endl;
	}
	else if (mergeVerify)
	{
		cout << "Merge sort was successful." << endl;
	}
	cout << "*********************************************" << endl;
	delete[] array2;
	array2 = NULL;

	cout << "Starting quick sort for " << quickArraySize << " elements" 
<< endl;
	int* array3 = new int[quickArraySize]; // allocate heap memory
	for (int i = 0; i < quickArraySize; i++)
	{
		array3[i] = rand();
	}
	quickRunner(quickArraySize, array3);
	bool quickVerify = verify(quickArraySize, array3);
	if (!quickVerify)
	{
		cout << "Quick sort failed!!!" << endl;
	}
	else if (quickVerify)
	{
		cout << "Quick sort was successful." << endl;
	}
	cout << "*********************************************" << endl;
	delete[] array3;
	array3 = NULL;
	cout << "End of program" << endl;
	return 0;
}


bool verify(int size, int array[])
{
	for (int i = 0; i < size - 1; i++)
	{
		if (array[i] > array[i + 1])
```

```
            {
                  return false;
            }
      }
      return true;
}
```

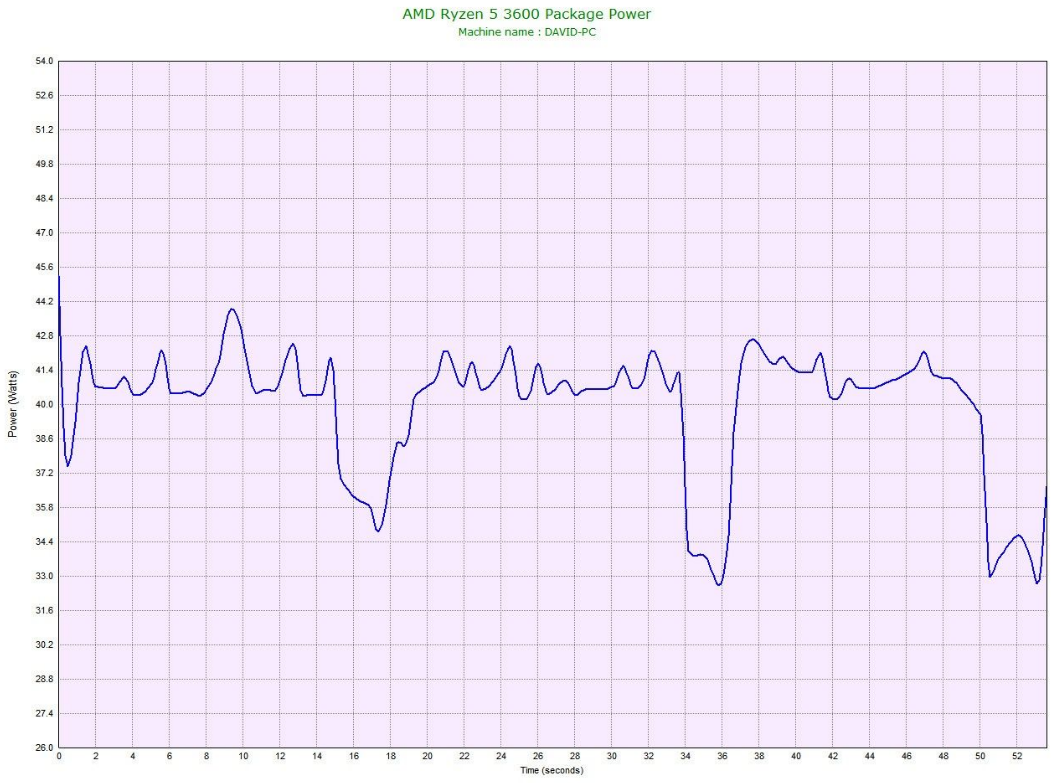
Appendix 5: sorts.h

```
// ****************************************
// Program Title: Lab 09
// Project File: sorts.h
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 03/15/2021
// ****************************************
// C++ program for insertion sort   adapted from GeeksforGeeks -
rathbhupendra
// C++ program for quick sort       adapted from GeeksforGeeks -
rathbhupendra
// C++ program for merge sort       adapted from GeeksforGeeks - Mayank
Tyagi

#ifndef SORTS
#define SORTS
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
using namespace std;

// Sorting functions
void quickSort(int arr[], int low, int high);
int partition(int arr[], int low, int high);
void merge(int arr[], int l, int m, int r);
void mergeSort(int arr[], int l, int r);
void insertionSort(int arr[], int n);
void swap(int* a, int* b);

// Runner and verify functions
void insertRunner(int size, int arr[]);
void mergeRunner(int size, int arr[]);
void quickRunner(int size, int arr[]);
bool verify(int size, int arr[]);

#endif
```

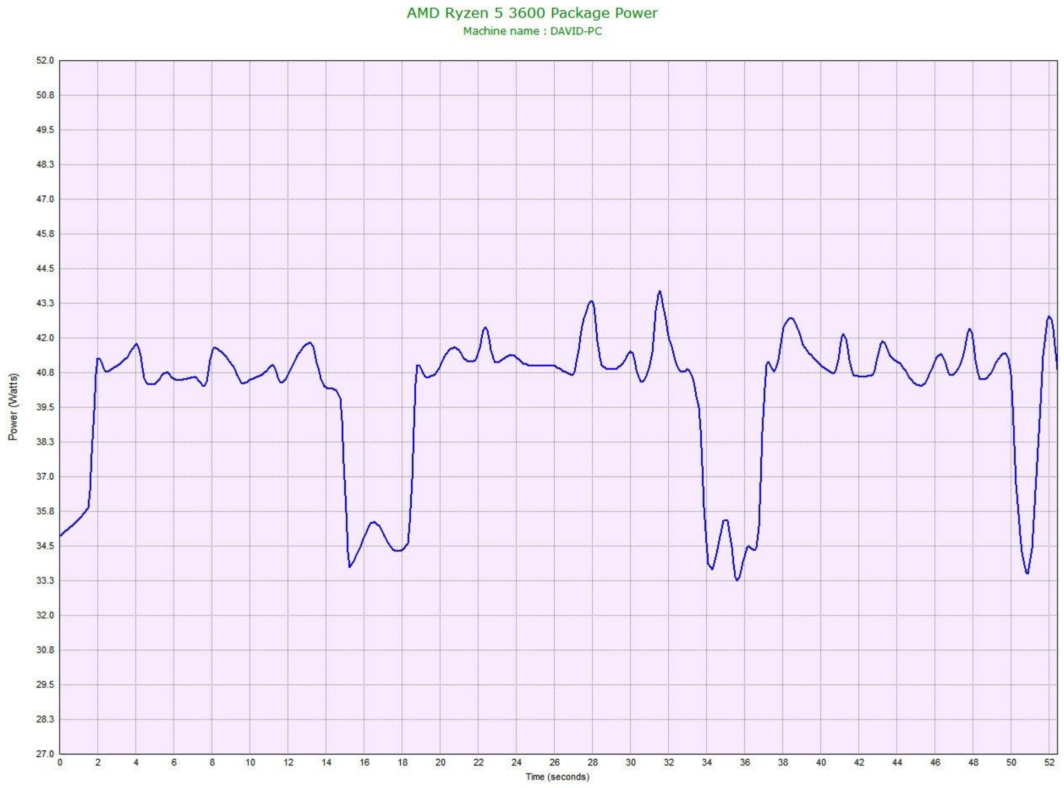
Appendix 6: Optimization Level 0 Package Power Run 1



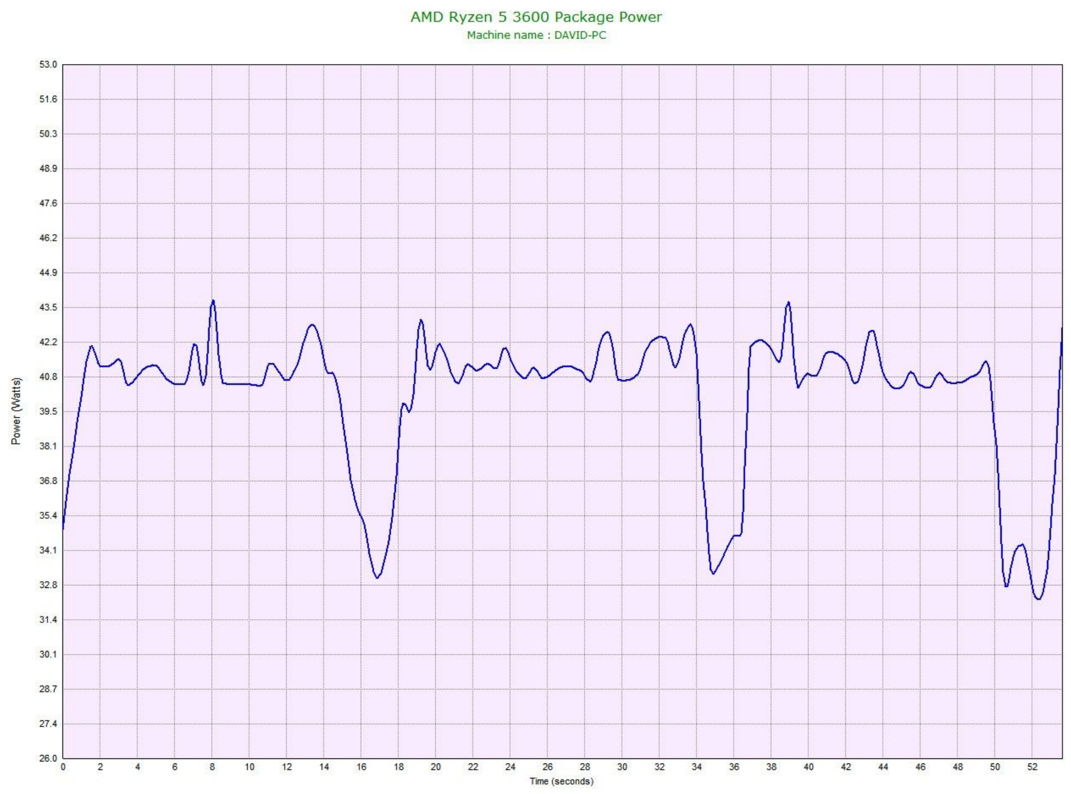
Appendix 7: Optimization Level 0 Package Power Run 2



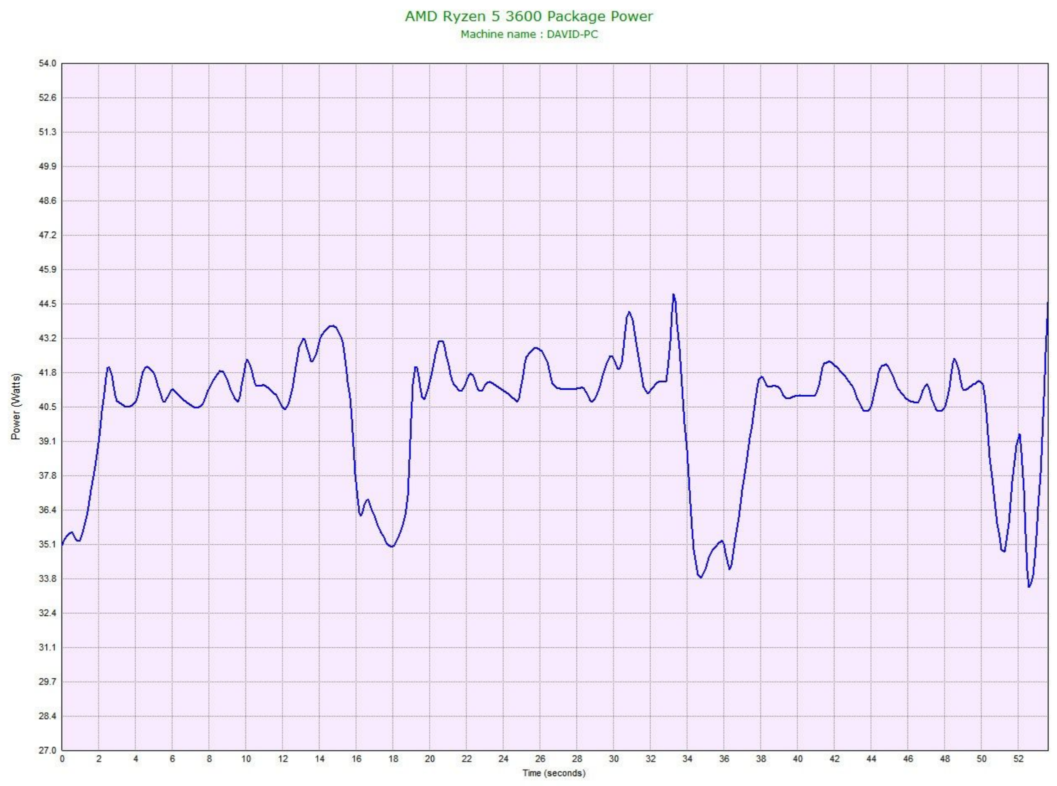
Appendix 8: Optimization Level 0 Package Power Run 3



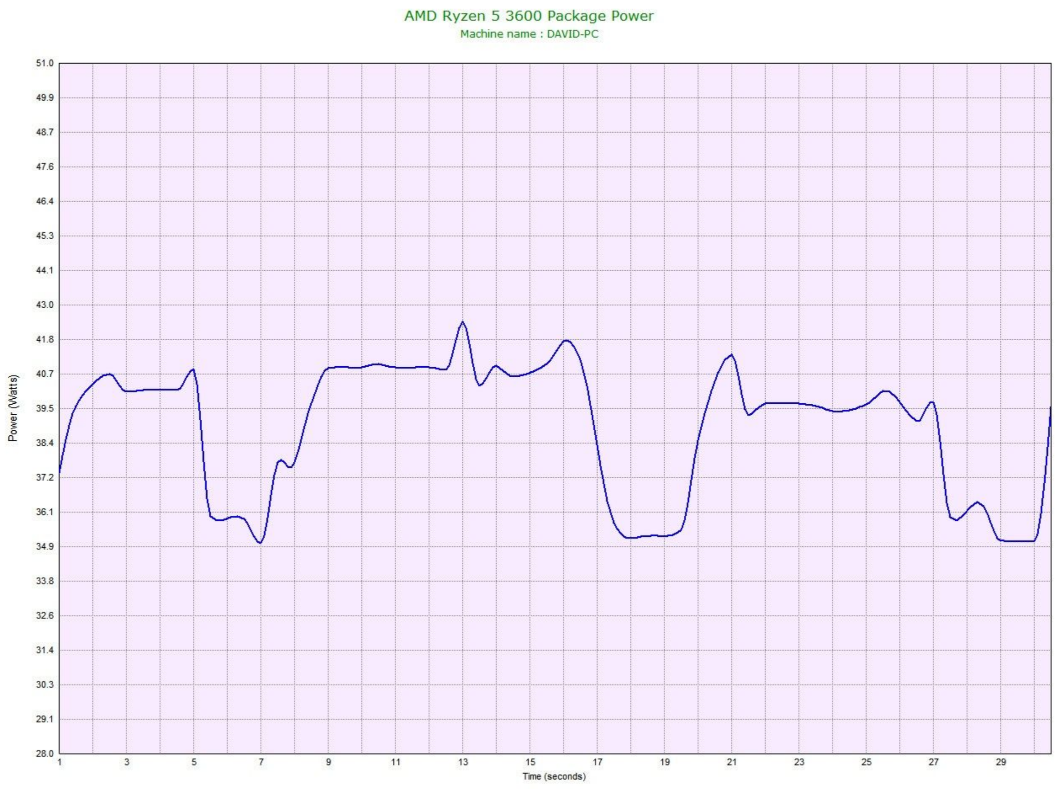
Appendix 9: Optimization Level 0 Package Power Run 4



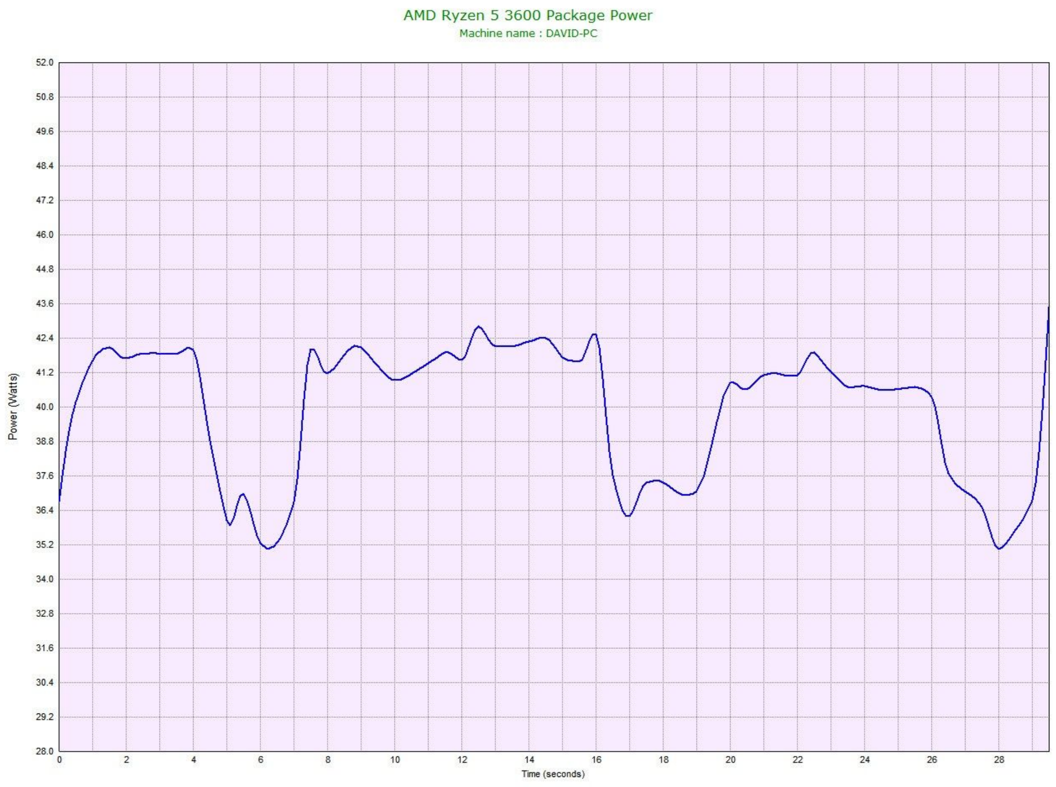
Appendix 10: Optimization Level 0 Package Power Run 5



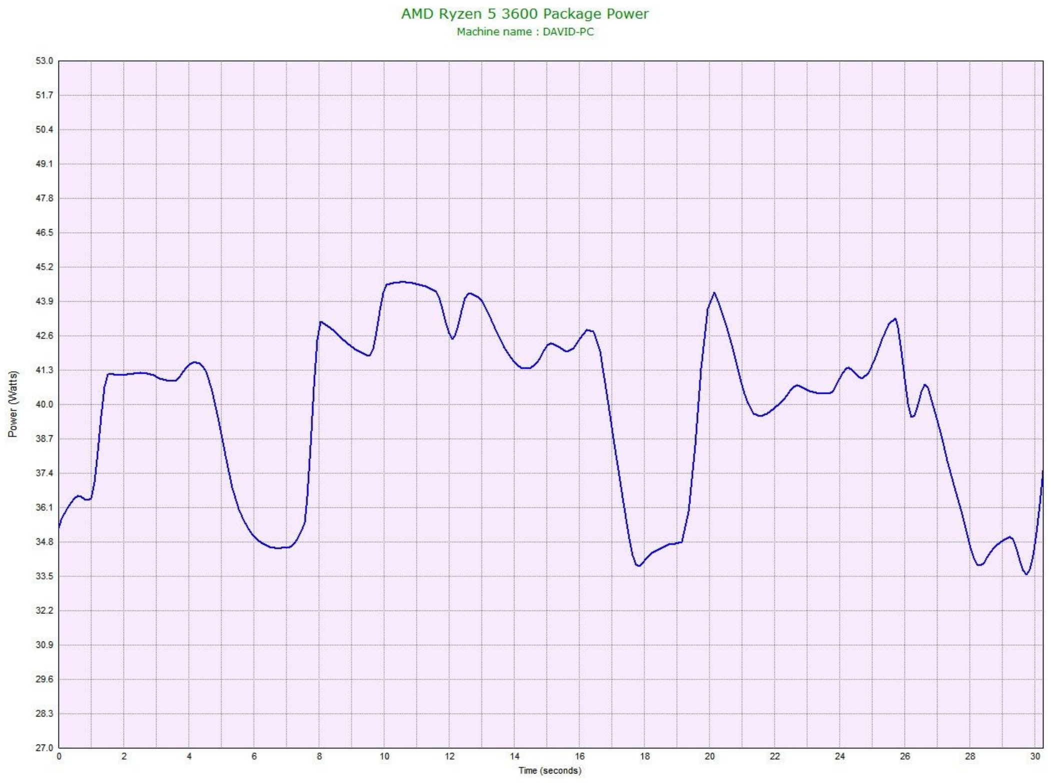
Appendix 11: Optimization Level 3 Package Power Run 1



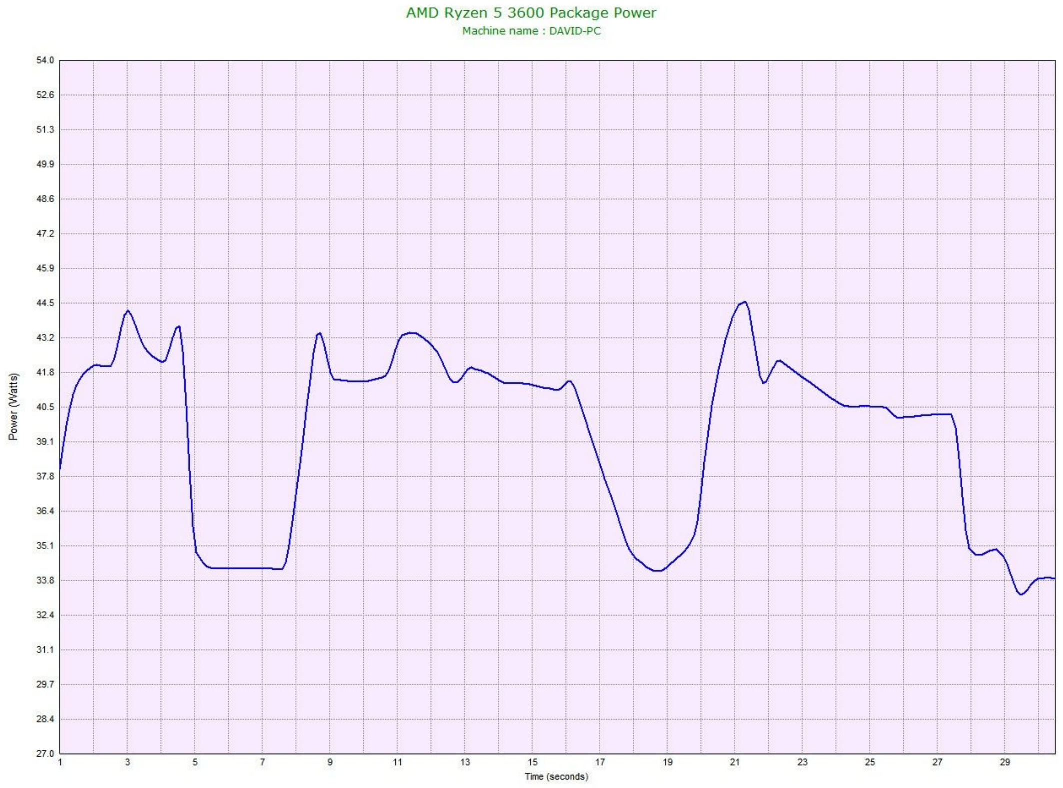
Appendix 12: Optimization Level 3 Package Power Run 2



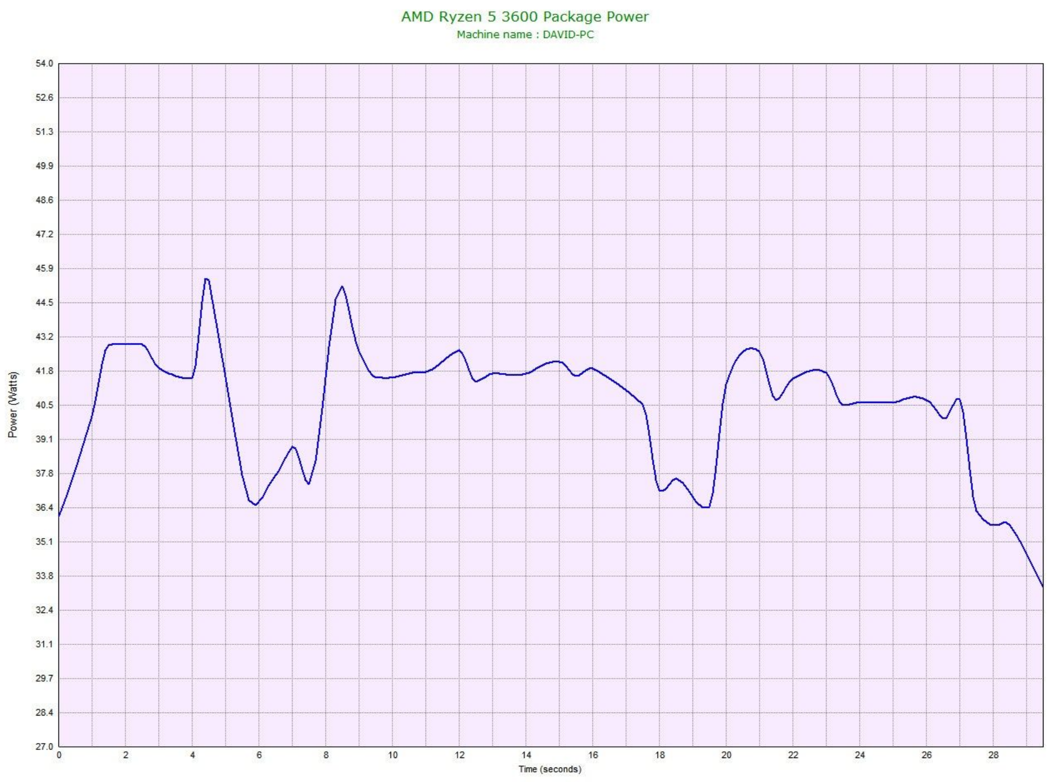
Appendix 13: Optimization Level 3 Package Power Run 3



Appendix 14: Optimization Level 3 Package Power Run 4



Appendix 15: Optimization Level 3 Package Power Run 5


AMD Ryzen 5 3600 Package Power
Machine name : DAVID-PC
Power (Watts)
Time (seconds)