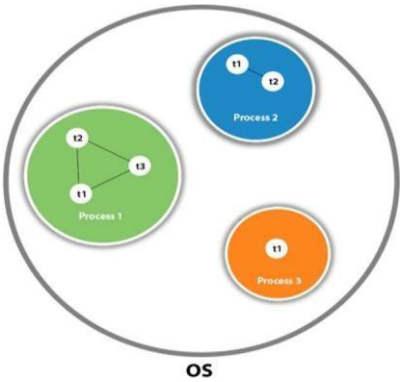
A brief observation of the differences between Threads and Processes using various Sorting Algorithms

CS-2006 Project Report

- M Qasim Bin Naveed (22K-4380)

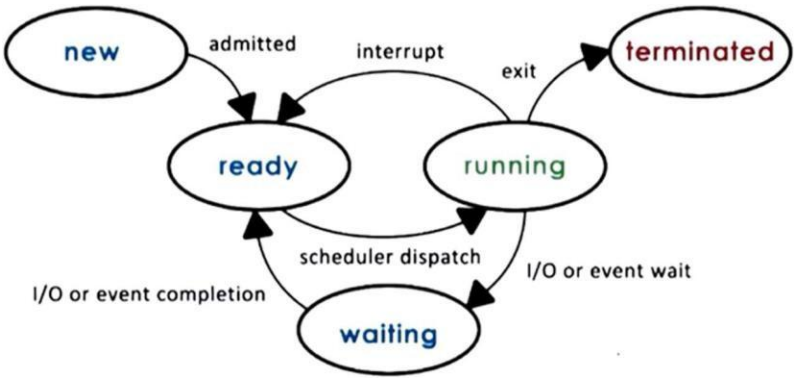


In the following few sections, our report will provide a brief but detailed overview as well as summarizing our efforts made in our research and (towards the end) elaborating on the code itself.

What are Threads and Processes?

Threads are sometimes referred to as subset of processes with the added benefit of them being far more "lightweight. Their need arises most commonly when global variables need to be shared across independent tasks.

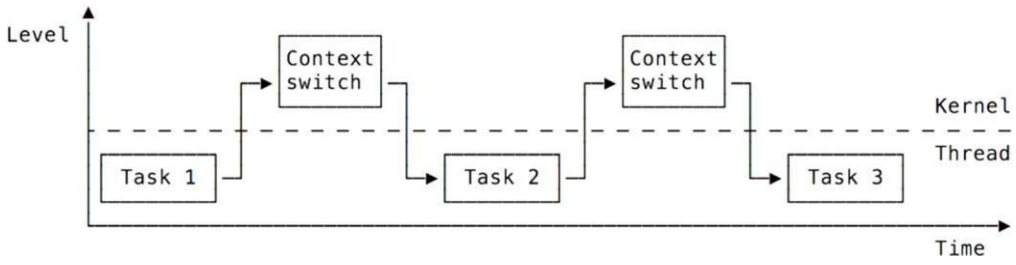
Conversely, a *process* is an instance of a program written, compiled, and run its own separate address space. There can be multiple instances of a same program (each running in their own address spaces. In other words, a program that is loaded into computer's memory and is in a state of execution is called a process.



Part I: *The differences between Threads and Processes*

Let's start from the basics; we already know that a process is a program in execution and a thread is a segment of a program thus making it lighter. Processes also take longer to context switch which begs the question, which of the two amalgams take up more or less time? This leads us into what our project aims to clarify through a visual representation.

The difference between processes and threads is clarified only when it is subjected to be experimented on a "chunk of code". When computed on a specific test subject (sorting algorithms in our scenario), we start to notice subtle changes between the two, mainly the time taken, which is what we have focused on further elaborating.



Part II: *Why Sorting Algorithms?*

To test our theory of threads taking less time overall in solving a computation as opposed to processes, we set about finding what type of code would best challenge both the different ways and finally settled on the 5 fundamental sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Shell Sort and Comb Sort.

We concluded that these various sorting algorithms contained enough variance with a diverse set of time and space complexities to allow our multiple threads and forked processes to have a fair/level playing field to test their times taken in each algorithm.

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
CombSort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)

Tools and Environment

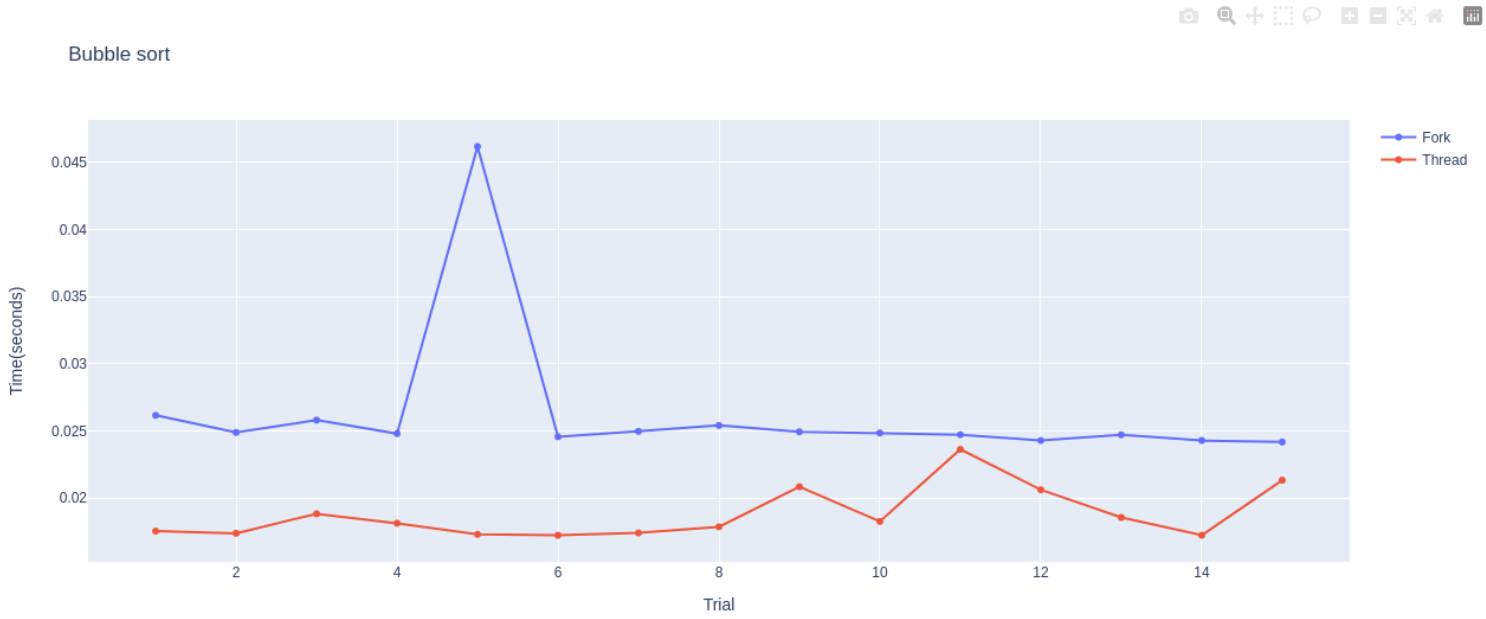
Our project was implemented on the Linux operating system Debian-based distro. Although on our relatively minute scale, using different distros would not have made a significant change that could be quantified, hence we decided to stick with Ubuntu distribution for Linux.

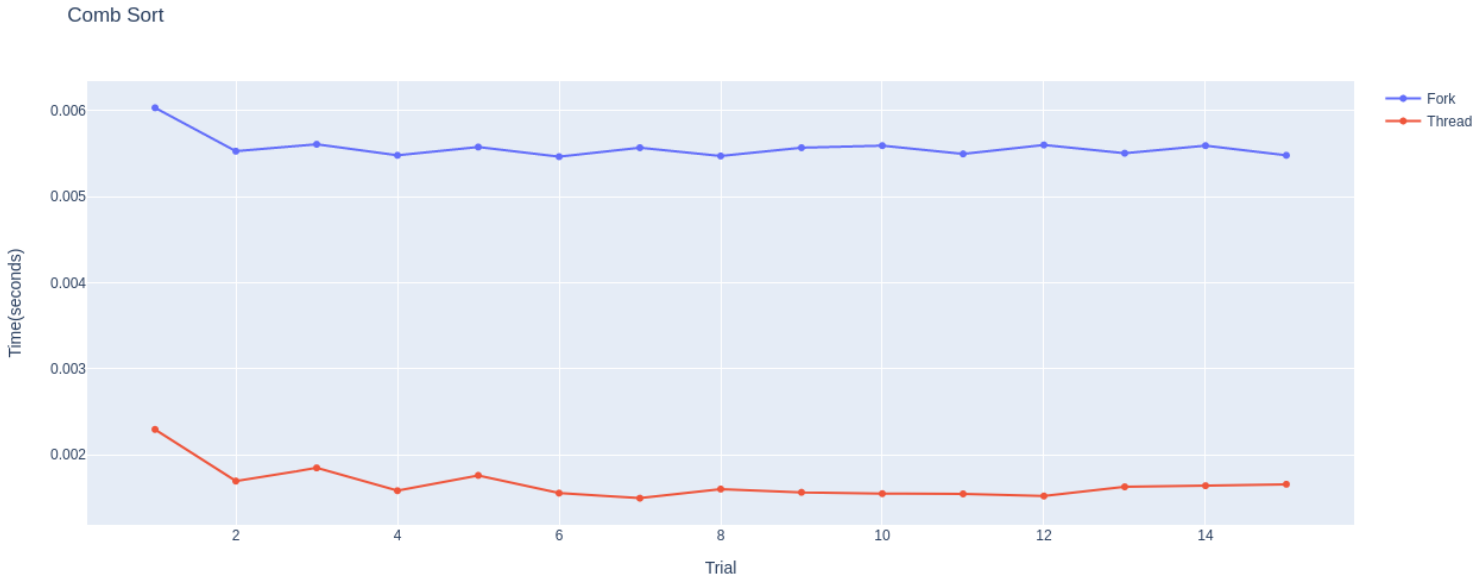
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#define ARR_SIZE 1000
int arr[ARR_SIZE];
```

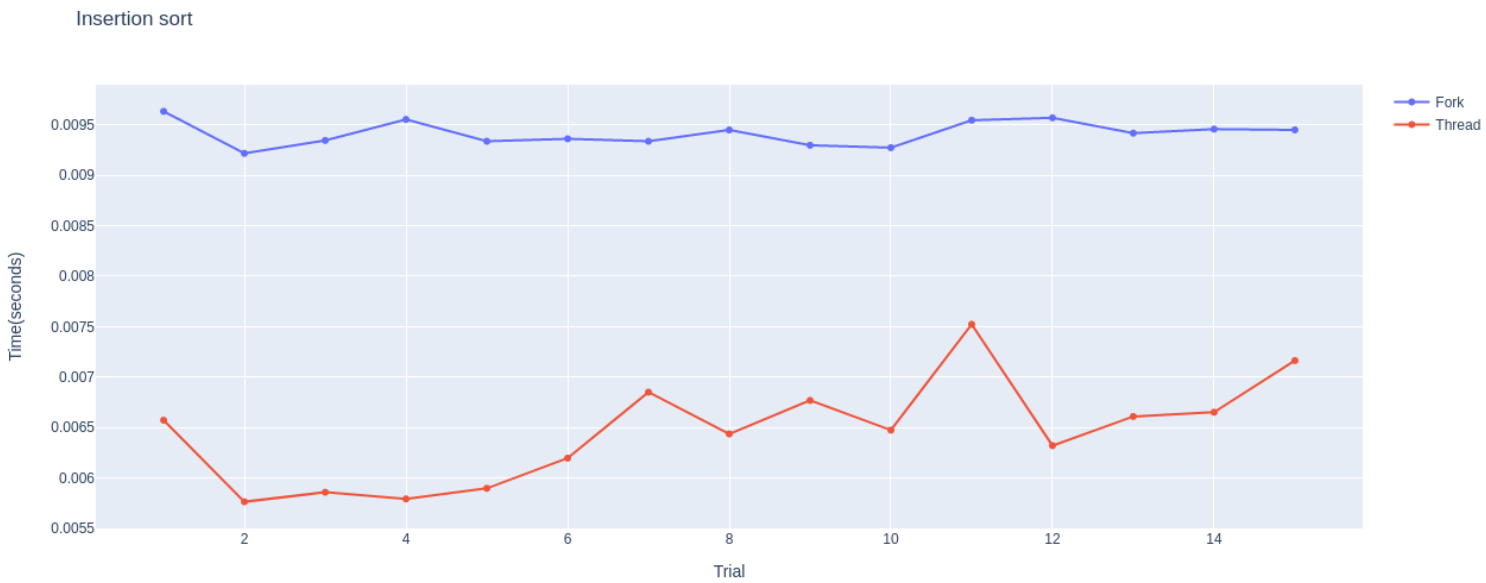
Linux allows us to implement forking more easily as well due to its simpler system calls and low-level API. For multithreading we used the help of POSIX PThread API which can be seen amongst the various libraries used in our project.

Our code was written in the C programming language and has been compiled using the GCC compiler

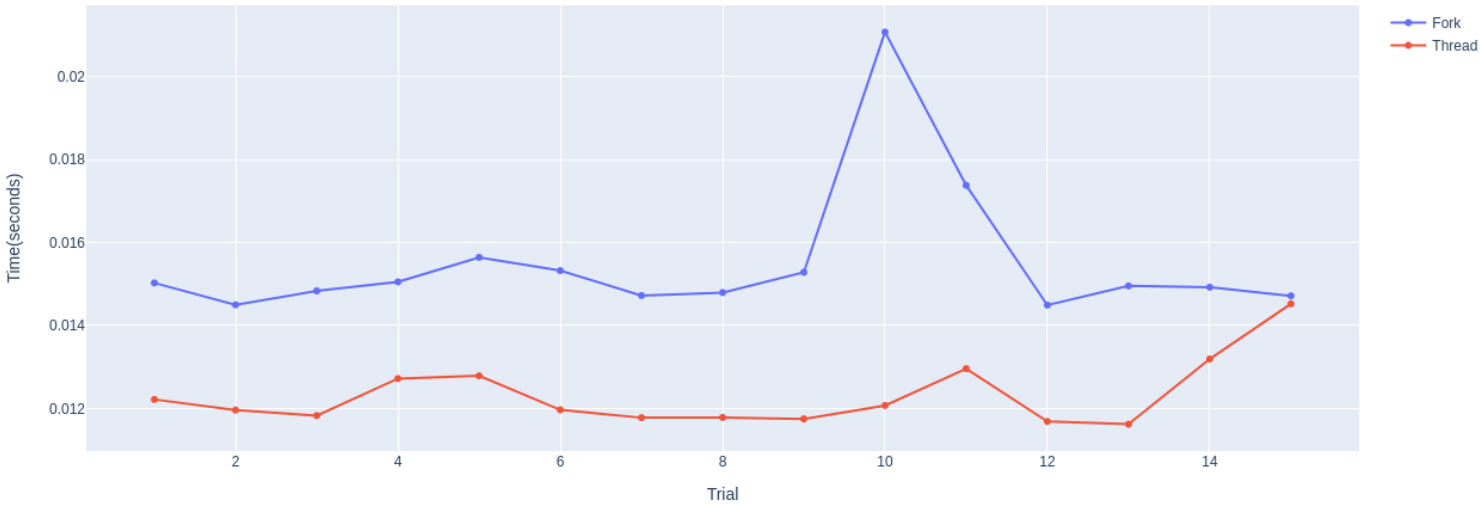
Likewise, to represent our results graphically, we used the Plotly Graph_Objects library in Python by first importing our output times into appropriate .csv files and then reading and plotting the .date on line charts with number of trials (15 for each algorithm) and time taken on the x and y axes respectively.



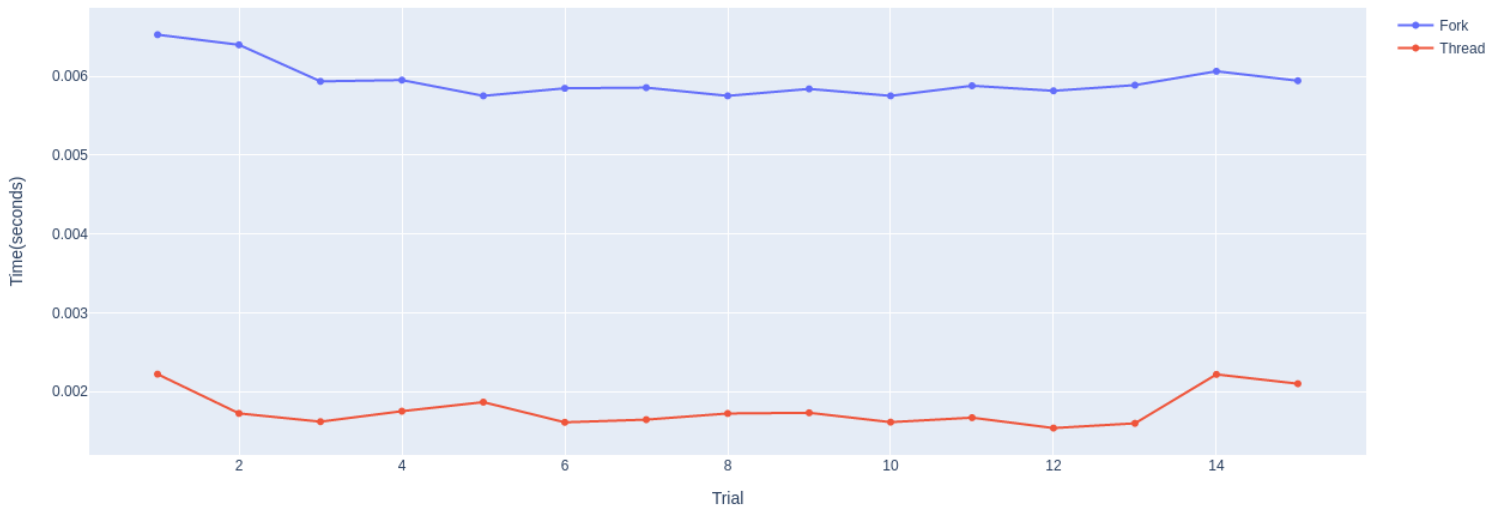

Comb Sort
Time(seconds)
0.006
0.005
0.004
0.003
0.002
2
4
6
8
10
12
14
Trial
Fork
Thread


Insertion sort
Time(seconds)
0.0095
0.009
0.0085
0.008
0.0075
0.007
0.0065
0.006
0.0055
2
4
6
8
10
12
14
Trial
Fork
Thread

Selection sort



Shell sort



Bringing the code to life

Our approach to implementing our project was quite straight-forward. Generate an array of size 1000, containing random positive integer values ranging from 0 to 10,000 and apply the same sorting algorithms across 8 threads and 8 processes while measuring the time taken for the entire collection of threads/processes to complete sorting the array using whichever algorithm the user selects, for 15 trials, and finally writing/appending the result in a .csv file which is utilized in plotting the graph through a Python script we wrote.

Although it sounds easier said than done, we implemented system calls from the exec family of functions in C language, namely execvp to pass the randomly generated array and execute the different C files for forking 8 child *processes* to run the algorithm on the passed array.

What we Concluded…

Throughout our results, threads constantly took ***less*** overall time to be be created and to sort the array as compared to the forked processes. This indicates that not only are threads faster in computing the same algorithm across multiple platforms but also their lightweight context t switching and ability to share information across multiple threads help lp streamline the overhead taken up by the more *complex* sorting algorithms.

This, mixed with small titbits of facts such as threads using less resources as compared to forked processes, requiring less time for creation and deletion and all the user level threads being treated as a **single** task by the OS help explain how we arrived at our conclusion.

Our conclusion is further supported by our graphs which illustrate the same result. *Threads* almost always take **less** time than *Processes.*

Code:

Every Sorting algorithm file contains that sorting algorithm which is a process and the project.c file solving all the processes through the concept of threading and joins all of the solutions through fork whereas the os_project.py file launches all the charts on the default browser of the system.

Bubble.c:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#define ARR_SIZE 1000
```

```c
int arr[ARR_SIZE];  //creating global array

//HERE the parameter of func converts the elements which it recieved as string form
//from previous file into integer type
void ParseParameters(char *argv[])
{
  for (int i = 0, j = 1; j < ARR_SIZE + 1; i++, j++)
    sscanf(argv[j], "%d", &arr[i]);
}

//same sorting algorithm as in previous/main file
void BubbleSort()
{
  int temp;
  for (int i = 0; i < ARR_SIZE; i++)
  {
    for (int j = 0; j < ARR_SIZE - i - 1; j++)
    {
      if (arr[j] > arr[j + 1])
      {
        temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

int main(int argc, char *argv[])
{
  FILE *fp;

  //THIS TIME we open the file in append mode rather than write
  //because we want to append the results for 15 trials
  fp = fopen("BubbleFork.csv", "a");
  if(fp == NULL)
    printf("ERROR!\n");

  int k;
  int id = getpid();

  double timetaken1;
  //the clock starts here
  clock_t start1 = clock();

  //running 3 fork statements will create 2^3=8 processes, this is y we used 8 threads aswell
  fork();
  fork();
```

```
  fork();

  //the clock stop for parent process BECAUSE
  //we need the time taken for process CREATION, and we are assumign here all processes
  //take the same time for creation so we will just add it everytime our process runs the algo
  if (getpid() == id)
  {
    clock_t end1 = clock();
    timetaken1 = (double)(end1 - start1) / CLOCKS_PER_SEC;
  }

  //similarly 15 trials here aswell
  for (k = 0; k < 15; k++)
  {
    clock_t start = clock();

    //we parse the parameters EVERYTIME SO EACH PROCESS DOSENT RECIEVE AN "ALREADY SORTED" ARRAY
    ParseParameters(argv);
    BubbleSort();

    //stopping the clock once the process has finished sorting
    clock_t end = clock();

  //here we add the time taken for PROCESS CREATIONS aswell BECAUSE in threading we included
  //the time taken for THREAD CREATION aswell.
    double timetaken = ((double)(end - start) / CLOCKS_PER_SEC) + timetaken1;


  if (getpid() == id)
    {
      if(k==14)
        fprintf(fp,"%f,15", timetaken *8);
      else
        fprintf(fp, "%f,%d\n", (timetaken * 8),(k+1));

      //here we print the time taken for 8 different processes
      printf("Time taken by 8 \x1b[1mPROCESSES\x1b[0m:\x1b[92m %f "
             "seconds\x1b[0m\n\n",
             (timetaken * 8));
    }
  }

  //and finally close the file
  fclose(fp);
}
```

Comb.c:

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#define ARR_SIZE 1000

int arr[ARR_SIZE];

void ParseParameters(char *argv[]) {
  for (int i = 0, j = 1; j < ARR_SIZE + 1; i++, j++)
    sscanf(argv[j], "%d", &arr[i]);
}

int getNextGap(int gap) {
  // Shrink gap by Shrink factor
  gap = (gap * 10) / 13;
  if (gap < 1)
    return 1;
  return gap;
}

void CombSort() {
  int n = ARR_SIZE;
  int gap = n;
  int i, temp;
  int swapped = 1;

  while (gap != 1 || swapped == 1) {
    gap = getNextGap(gap);
    swapped = 0;
    for (i = 0; i < n - gap; i++) {
      if (arr[i] > arr[i + gap]) {

        temp = arr[i];
        arr[i] = arr[i + gap];
        arr[i + gap] = temp;

        swapped = 1;
      }
    }
  }
}

int main(int argc, char *argv[]) {
  FILE *fp;
```

```
  fp = fopen("CombFork.csv", "a");
  if(fp == NULL)
    printf("ERROR!\n");

  // fprintf(fp,"time,trial\n");
  int k;
  int id = getpid();
  double times[15];
  double timetaken1;
  clock_t start1 = clock();

  fork();
  fork();
  fork();
  if (getpid() == id)
  {
    clock_t end1 = clock();
    timetaken1 = (double)(end1 - start1) / CLOCKS_PER_SEC;
  }

  for (k = 0; k < 15; k++)
  {
    clock_t start = clock();
    ParseParameters(argv);
    CombSort();
    clock_t end = clock();

    double timetaken = ((double)(end - start) / CLOCKS_PER_SEC) + timetaken1;
    if (getpid() == id)
    {
      if(k==14)
        fprintf(fp,"%f,15", timetaken *8);
      else
        fprintf(fp, "%f,%d\n", (timetaken * 8),(k+1));

      printf("Time taken by 8 \x1b[1mPROCESSES\x1b[0m:\x1b[92m %f "
             "seconds\x1b[0m\n\n",
             (timetaken * 8));
    }
  }
  fclose(fp);
}
```

Insertion.c:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#define ARR_SIZE 1000

int arr[ARR_SIZE];

void ParseParameters(char *argv[])
{
  for (int i = 0, j = 1; j < ARR_SIZE + 1; i++, j++)
    sscanf(argv[j], "%d", &arr[i]);
}

void InsertionSort()
{
  int n = ARR_SIZE;
  int temp = 0;
  int i, j, key;

  for (i = 1; i < n; i++)
  {
    key = arr[i];
    j = i - 1;

    /* Move elements of arr[0..i-1], that are
      greater than key, to one position ahead
      of their current position */
    while (j >= 0 && arr[j] > key)
    {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
}

int main(int argc, char *argv[])
{
  FILE *fp;
  fp = fopen("InsertionFork.csv", "a");
  if(fp == NULL)
    printf("ERROR!\n");

  // fprintf(fp,"time,trial\n");
  int k;
  int id = getpid();
  double times[15];
  double timetaken1;
```

```
  clock_t start1 = clock();

  fork();
  fork();
  fork();
  if (getpid() == id)
  {
    clock_t end1 = clock();
    timetaken1 = (double)(end1 - start1) / CLOCKS_PER_SEC;
  }

  for (k = 0; k < 15; k++)
  {
    clock_t start = clock();
    ParseParameters(argv);
    InsertionSort();
    clock_t end = clock();

    double timetaken = ((double)(end - start) / CLOCKS_PER_SEC) + timetaken1;
    if (getpid() == id)
    {
      if(k==14)
        fprintf(fp,"%f,15", timetaken *8);
      else
        fprintf(fp, "%f,%d\n", (timetaken * 8),(k+1));

      printf("Time taken by 8 \x1b[1mPROCESSES\x1b[0m:\x1b[92m %f "
             "seconds\x1b[0m\n\n",
             (timetaken * 8));
    }
  }
  fclose(fp);
}
```

Selection.c:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#define ARR_SIZE 1000

int arr[ARR_SIZE];
```

```c
void ParseParameters(char *argv[])
{
  for (int i = 0, j = 1; j < ARR_SIZE + 1; i++, j++)
    sscanf(argv[j], "%d", &arr[i]);
}

void SelectionSort()
{
  int n = ARR_SIZE;
  int temp = 0;
  int i, j, min_idx;

  for (i = 0; i < n - 1; i++)
  {
    // Find the minimum element in unsorted array
    min_idx = i;
    for (j = i + 1; j < n; j++)
    {
      if (arr[j] < arr[min_idx])
      {
        min_idx = j;
      }
    }

    // Swap the found minimum element with the first element
    temp = arr[min_idx];
    arr[min_idx] = arr[i];
    arr[i] = temp;
  }
}

int main(int argc, char *argv[])
{
  FILE *fp;
  fp = fopen("SelectionFork.csv", "a");
  if(fp == NULL)
    printf("ERROR!\n");

  // fprintf(fp,"time,trial\n");
  int k;
  int id = getpid();
  double times[15];
  double timetaken1;
  clock_t start1 = clock();

  fork();
  fork();
  fork();
  if (getpid() == id)
```

```
  {
    clock_t end1 = clock();
    timetaken1 = (double)(end1 - start1) / CLOCKS_PER_SEC;
  }

  for (k = 0; k < 15; k++)
  {
    clock_t start = clock();
    ParseParameters(argv);
    SelectionSort();
    clock_t end = clock();

    double timetaken = ((double)(end - start) / CLOCKS_PER_SEC) + timetaken1;
    if (getpid() == id)
    {
      if(k==14)
        fprintf(fp,"%f,15", timetaken *8);
      else
        fprintf(fp, "%f,%d\n", (timetaken * 8),(k+1));

      printf("Time taken by 8 \x1b[1mPROCESSES\x1b[0m:\x1b[92m %f "
             "seconds\x1b[0m\n\n",
             (timetaken * 8));
    }
  }
  fclose(fp);
}
```

Shell.c:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#define ARR_SIZE 1000

int arr[ARR_SIZE];

void ParseParameters(char *argv[]) {
  for (int i = 0, j = 1; j < ARR_SIZE + 1; i++, j++)
    sscanf(argv[j], "%d", &arr[i]);
}

int ShellSort() {
  int n = ARR_SIZE;
```

```
  int temp = 0;
  int i, j, key;

  for (int gap = n / 2; gap > 0; gap /= 2) {

    for (int i = gap; i < n; i += 1) {
      int temp = arr[i];
      int j;
      for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
        arr[j] = arr[j - gap];

      arr[j] = temp;
    }
  }
  return 0;
}

int main(int argc, char *argv[]) {
  FILE *fp;
  fp = fopen("ShellFork.csv", "a");
  if(fp == NULL)
    printf("ERROR!\n");

  // fprintf(fp,"time,trial\n");
  int k;
  int id = getpid();
  double times[15];
  double timetaken1;
  clock_t start1 = clock();

  fork();
  fork();
  fork();
  if (getpid() == id)
  {
    clock_t end1 = clock();
    timetaken1 = (double)(end1 - start1) / CLOCKS_PER_SEC;
  }

  for (k = 0; k < 15; k++)
  {
    clock_t start = clock();
    ParseParameters(argv);
    ShellSort();
    clock_t end = clock();

    double timetaken = ((double)(end - start) / CLOCKS_PER_SEC) + timetaken1;
    if (getpid() == id)
    {
```

```
      if(k==14)
        fprintf(fp,"%f,15", timetaken *8);
      else
        fprintf(fp, "%f,%d\n", (timetaken * 8),(k+1));

      printf("Time taken by 8 \x1b[1mPROCESSES\x1b[0m:\x1b[92m %f "
             "seconds\x1b[0m\n\n",
             (timetaken * 8));
    }
  }
  fclose(fp);
}
```

Project.c:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>

#define ARR_SIZE 1000 //we are taking an array of size 1000 always
int arr[ARR_SIZE];


//elementart sorting algorithms in their respective functions
void BubbleSort()
{
    int temp = 0;
    int i, j;
    int temparr[ARR_SIZE];

    //temp arr here so each thread dosent recieve an "ALREADY SORTED" array
    //and it dosent disturb the global array
    for (i = 0; i < ARR_SIZE; i++)
    {
        temparr[i] = arr[i];
    }

    for (i = 0; i < ARR_SIZE - 1; i++)
    {
        for (j = 0; j < ARR_SIZE - i - 1; j++)
        {
            if (temparr[j] > temparr[j + 1])
```

```
            {
                temp = temparr[j];
                temparr[j] = temparr[j + 1];
                temparr[j + 1] = temp;
            }
        }
    }
}

void SelectionSort()
{
    int n = ARR_SIZE;
    int temp = 0;
    int i, j, min_idx;
    int temparr[ARR_SIZE];

    for (i = 0; i < ARR_SIZE; i++)
    {
        temparr[i] = arr[i];
    }

    for (i = 0; i < n - 1; i++)
    {

        min_idx = i;
        for (j = i + 1; j < n; j++)
        {
            if (temparr[j] < temparr[min_idx])
            {
                min_idx = j;
            }
        }

        temp = temparr[min_idx];
        temparr[min_idx] = temparr[i];
        temparr[i] = temp;
    }
}

int ShellSort()
{
    int n = ARR_SIZE;
    int temp = 0;
    int i, j, key;
    int temparr[ARR_SIZE];

    for (i = 0; i < ARR_SIZE; i++)
    {
        temparr[i] = arr[i];
```

```
    }

    for (int gap = n / 2; gap > 0; gap /= 2)
    {

        for (int i = gap; i < n; i += 1)
        {

            int temp = temparr[i];

            int j;
            for (j = i; j >= gap && temparr[j - gap] > temp; j -= gap)
                temparr[j] = temparr[j - gap];

            temparr[j] = temp;
        }
    }
    return 0;
}

void InsertionSort()
{
    int n = ARR_SIZE;
    int temp = 0;
    int i, j, key;
    int temparr[ARR_SIZE];

    for (i = 0; i < ARR_SIZE; i++)
    {
        temparr[i] = arr[i];
    }

    for (i = 1; i < n; i++)
    {
        key = temparr[i];
        j = i - 1;


        while (j >= 0 && temparr[j] > key)
        {
            temparr[j + 1] = temparr[j];
            j = j - 1;
        }
        temparr[j + 1] = key;
    }
}

int getNextGap(int gap)
{
    gap = (gap * 10) / 13;
```

```
    if (gap < 1)
        return 1;
    return gap;
}

void CombSort()
{
    int n = ARR_SIZE;

    int gap = n;
    int i, temparr[ARR_SIZE], temp;

    for (i = 0; i < ARR_SIZE; i++)
    {
        temparr[i] = arr[i];
    }

    int swapped = 1;

    while (gap != 1 || swapped == 1)
    {

        gap = getNextGap(gap);

        swapped = 0;

        for (i = 0; i < n - gap; i++)
        {
            if (temparr[i] > temparr[i + gap])
            {

                temp = temparr[i];
                temparr[i] = temparr[i + gap];
                temparr[i + gap] = temp;

                swapped = 1;
            }
        }
    }
}

//filing function that initializes file and adds the axis headings needed for later
//in python plotly library
//filingUwU is more for the forking program than the threading one
void filingUwU(char arr[50])
{
    FILE *fio;
    fio = fopen(arr, "w");
```

```
    fprintf(fio, "time,trial\n");
    fclose(fio);

}


int main()
{
    FILE *fp;

    int i, k, ch;

    //initializing the array with random values ranging between
    //0 and 10,000
    srand(time(0));
    for (i = 0; i < ARR_SIZE; i++)
    {
        arr[i] = rand() % 10000;
    }

    // Create a dynamic string array to pass data to other files
    char **strArr = calloc(ARR_SIZE + 2, sizeof(char *));
    for (int i = 0; i < ARR_SIZE + 2; i++)
        strArr[i] = calloc(50, sizeof(char));   //strArr contains our filename, array and NULL
        //these are needed when passing parameters to execvp later on

    // Convert the data to string and populate the string array
    for (int i = 1; i < ARR_SIZE + 1; i++)
        sprintf(strArr[i], "%d", arr[i]);

    strArr[ARR_SIZE + 1] = NULL; // Last index of string array must point to NULL

    pthread_t t[8]; //creating an array of 8 threads

    //menu type program for user to select which type of sorting to run
    printf("Enter which sorting algorithm you would like to utilize through "
           "Multithreading and Forking: \n");
    printf("1. Bubble Sort\n");
    printf("2. Selection Sort\n");
    printf("3. Insertion Sort\n");
    printf("4. Shell Sort\n");
    printf("5. Comb Sort\n");

    printf("\nEnter choice: ");
    scanf("%d", &ch);

    if (ch == 1)
    {
        //creating a csv file for BubbleSort done in forking
        filingUwU("BubbleFork.csv");
```

```
    printf("\n\n\x1b[94m~~~~~~~~~~BUBBLE SORT~~~~~~~~~~\x1b[0m");

    //creating a csv file for BubbleSort done in threading
    //we have not used the filingUwU here because we would need to open it again anyways
    //cuz filingUwU closes the file
    fp = fopen("BubbleThread.csv", "w");
    fprintf(fp,"time,trial\n");

    //for 15 different number of trials we will create and join 8 different threads
    for (k = 0; k < 15; k++)
    {
        //starting the clock
        clock_t start = clock();
        for (i = 0; i < 8; i++)
            pthread_create(&t[i], NULL, (void *)&BubbleSort, NULL);

        for (i = 0; i < 8; i++)
            pthread_join(t[i], NULL);

        //ending the clock
        clock_t end = clock();

        double timetaken = (double)(end - start) / CLOCKS_PER_SEC;
        if (k == 14)
            fprintf(fp, "%f,15", timetaken);
        else
            fprintf(fp, "%f,%d\n", timetaken,k+1);

        printf(
            "\nTime taken by 8 \x1b[1mTHREADS\x1b[0m:\x1b[92m %f seconds\x1b[0m\n",
            timetaken);
    }
    fclose(fp);

    // First index of string array must be the file name to execute
    strcpy(strArr[0], "./bubble");

    //and now after achieving results of threading, we will send the array
    //to a forking program to get results of processes.
    if (execvp("./bubble", strArr) == -1)
        printf("Error!\n");
}

else if (ch == 2)
{
    filingUwU("SelectionFork.csv");
    printf("\n\n\x1b[94m~~~~~~~~~~SELECTION SORT~~~~~~~~~~\x1b[0m");
    fp = fopen("SelectionThread.csv", "w");
```

```
        fprintf(fp,"time,trial\n");

        for (k = 0; k < 15; k++)
        {

            clock_t start = clock();
            for (i = 0; i < 8; i++)
                pthread_create(&t[i], NULL, (void *)&SelectionSort, NULL);

            for (i = 0; i < 8; i++)
                pthread_join(t[i], NULL);

            clock_t end = clock();

            double timetaken = (double)(end - start) / CLOCKS_PER_SEC;
            if (k == 14)
                fprintf(fp, "%f,15", timetaken);
            else
                fprintf(fp, "%f,%d\n", timetaken,k+1);

            printf(
                "\nTime taken by 8 \x1b[1mTHREADS\x1b[0m:\x1b[92m %f seconds\x1b[0m\n",
                timetaken);
        }
        fclose(fp);
        // First index of string array must be the file name to execute
        strcpy(strArr[0], "./selection");
        if (execvp("./selection", strArr) == -1)
            printf("Error!\n");
    }

    else if (ch == 3)
    {
        filingUwU("InsertionFork.csv");
        printf("\n\n\x1b[94m~~~~~~~~~~INSERTION SORT~~~~~~~~~~\x1b[0m");
        fp = fopen("InsertionThread.csv", "w");
        fprintf(fp,"time,trial\n");

        for (k = 0; k < 15; k++)
        {
            clock_t start = clock();
            for (i = 0; i < 8; i++)
                pthread_create(&t[i], NULL, (void *)&InsertionSort, NULL);

            for (i = 0; i < 8; i++)
                pthread_join(t[i], NULL);

            clock_t end = clock();
            double timetaken = (double)(end - start) / CLOCKS_PER_SEC;
```

```
        if (k == 14)
            fprintf(fp, "%f,15", timetaken);
        else
            fprintf(fp, "%f,%d\n", timetaken,k+1);

        printf(
            "\nTime taken by 8 \x1b[1mTHREADS\x1b[0m:\x1b[92m %f seconds\x1b[0m\n",
            timetaken);
    }
    fclose(fp);

    // First index of string array must be the file name to execute
    strcpy(strArr[0], "./insertion");
    if (execvp("./insertion", strArr) == -1)
        printf("Error!\n");
}

else if (ch == 4)
{
    filingUwU("ShellFork.csv");
    printf("\n\n\x1b[94m~~~~~~~~~~SHELL  SORT~~~~~~~~~~\x1b[0m");
    fp = fopen("ShellThread.csv", "w");
    fprintf(fp,"time,trial\n");

    for (k = 0; k < 15; k++)
    {

        clock_t start = clock();
        for (i = 0; i < 8; i++)
            pthread_create(&t[i], NULL, (void *)&ShellSort, NULL);

        for (i = 0; i < 8; i++)
            pthread_join(t[i], NULL);

        clock_t end = clock();

        double timetaken = (double)(end - start) / CLOCKS_PER_SEC;
        if (k == 14)
            fprintf(fp, "%f,15", timetaken);
        else
            fprintf(fp, "%f,%d\n", timetaken,k+1);

        printf(
            "\nTime taken by 8 \x1b[1mTHREADS\x1b[0m:\x1b[92m %f seconds\x1b[0m\n",
            timetaken);
    }
    fclose(fp);

    // First index of string array must be the file name to execute
```

```
        strcpy(strArr[0], "./shell");
        if (execvp("./shell", strArr) == -1)
            printf("Error!\n");
    }
    else if (ch == 5)
    {
        filingUwU("CombFork.csv");
        printf("\n\n\x1b[94m~~~~~~~~~~COMB SORT~~~~~~~~~~\x1b[0m");
        fp = fopen("CombThread.csv", "w");
        fprintf(fp,"time,trial\n");

        for (k = 0; k < 15; k++)
        {

            clock_t start = clock();
            for (i = 0; i < 8; i++)
                pthread_create(&t[i], NULL, (void *)&CombSort, NULL);

            for (i = 0; i < 8; i++)
                pthread_join(t[i], NULL);

            clock_t end = clock();

            double timetaken = (double)(end - start) / CLOCKS_PER_SEC;
            if (k == 14)
                fprintf(fp, "%f,15", timetaken);
            else
                fprintf(fp, "%f,%d\n", timetaken,k+1);

            printf(
                "\nTime taken by 8 \x1b[1mTHREADS\x1b[0m:\x1b[92m %f seconds\x1b[0m\n",
                timetaken);
        }
        fclose(fp);

        // First index of string array must be the file name to execute
        strcpy(strArr[0], "./comb");
        if (execvp("./comb", strArr) == -1)
            printf("Error!\n");
    }

    else
    {
        printf("INVALID ENTRY!");
    }

    return 0;
}
```

Os_project.py:

```
# **Bubble Sort**


import plotly.graph_objects as go
import pandas as pd

df_usgsn03 = pd.read_csv('BubbleFork.csv',sep=',')
df_usgsn04 = pd.read_csv('BubbleThread.csv',sep=',')


df_usgsn03_sorted = df_usgsn03.sort_values(by='trial')
df_usgsn04_sorted = df_usgsn04.sort_values(by='trial')


fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x=df_usgsn03_sorted['trial'],
        y=df_usgsn03_sorted['time'],
        name = 'Fork'
    ))

fig.add_trace(
    go.Scatter(
        x=df_usgsn04_sorted['trial'],
        y=df_usgsn04_sorted['time'],
        name = 'Thread'
    ))

fig.update_layout(title = 'Bubble sort', xaxis_title='Trial',yaxis_title='Time(seconds)')

fig.show()



"""

# **Comb Sort**

"""

import plotly.graph_objects as go
import pandas as pd
```

```
df_usgsn03 = pd.read_csv('CombFork.csv',sep=',')
df_usgsn04 = pd.read_csv('CombThread.csv',sep=',')


df_usgsn03_sorted = df_usgsn03.sort_values(by='trial')
df_usgsn04_sorted = df_usgsn04.sort_values(by='trial')


fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x=df_usgsn03_sorted['trial'],
        y=df_usgsn03_sorted['time'],
        name = 'Fork'
    ))

fig.add_trace(
    go.Scatter(
        x=df_usgsn04_sorted['trial'],
        y=df_usgsn04_sorted['time'],
        name = 'Thread'
    ))

fig.update_layout(title = 'Comb Sort', xaxis_title='Trial',yaxis_title='Time(seconds)')

fig.show()



"""#**Insertion Sort**"""

import plotly.graph_objects as go
import pandas as pd

df_usgsn03 = pd.read_csv('InsertionFork.csv',sep=',')
df_usgsn04 = pd.read_csv('InsertionThread.csv',sep=',')


df_usgsn03_sorted = df_usgsn03.sort_values(by='trial')
df_usgsn04_sorted = df_usgsn04.sort_values(by='trial')


fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x=df_usgsn03_sorted['trial'],
        y=df_usgsn03_sorted['time'],
        name = 'Fork'
    ))
```

```
fig.add_trace(
    go.Scatter(
        x=df_usgsn04_sorted['trial'],
        y=df_usgsn04_sorted['time'],
        name = 'Thread'
    ))

fig.update_layout(title = 'Insertion sort', xaxis_title='Trial',yaxis_title='Time(seconds)')

fig.show()



"""#**Selection Sort**"""

import plotly.graph_objects as go
import pandas as pd

df_usgsn03 = pd.read_csv('SelectionFork.csv',sep=',')
df_usgsn04 = pd.read_csv('SelectionThread.csv',sep=',')


df_usgsn03_sorted = df_usgsn03.sort_values(by='trial')
df_usgsn04_sorted = df_usgsn04.sort_values(by='trial')


fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x=df_usgsn03_sorted['trial'],
        y=df_usgsn03_sorted['time'],
        name = 'Fork'
    ))

fig.add_trace(
    go.Scatter(
        x=df_usgsn04_sorted['trial'],
        y=df_usgsn04_sorted['time'],
        name = 'Thread'
    ))

fig.update_layout(title = 'Selection sort', xaxis_title='Trial',yaxis_title='Time(seconds)')

fig.show()



"""#**Shell Sort**"""

import plotly.graph_objects as go
```

```
import pandas as pd

df_usgsn03 = pd.read_csv('ShellFork.csv',sep=',')
df_usgsn04 = pd.read_csv('ShellThread.csv',sep=',')


df_usgsn03_sorted = df_usgsn03.sort_values(by='trial')
df_usgsn04_sorted = df_usgsn04.sort_values(by='trial')


fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x=df_usgsn03_sorted['trial'],
        y=df_usgsn03_sorted['time'],
        name = 'Fork'
    ))

fig.add_trace(
    go.Scatter(
        x=df_usgsn04_sorted['trial'],
        y=df_usgsn04_sorted['time'],
        name = 'Thread'

    ))

fig.update_layout(title = 'Shell sort', xaxis_title='Trial',yaxis_title='Time(seconds)')

fig.show()
```

THANK YOU