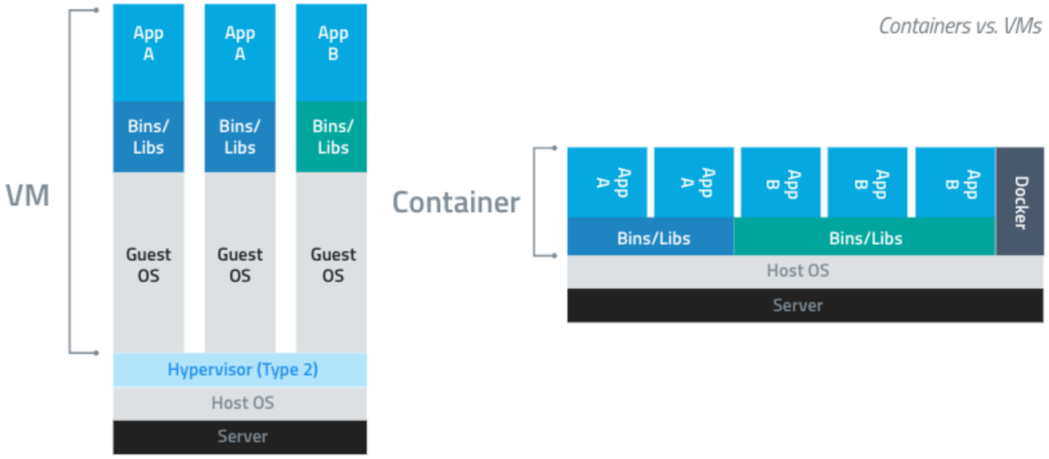
**MUHAMMAD MINHAL**
**20k-0467**

**Mpi code using Docker**

## Introduction

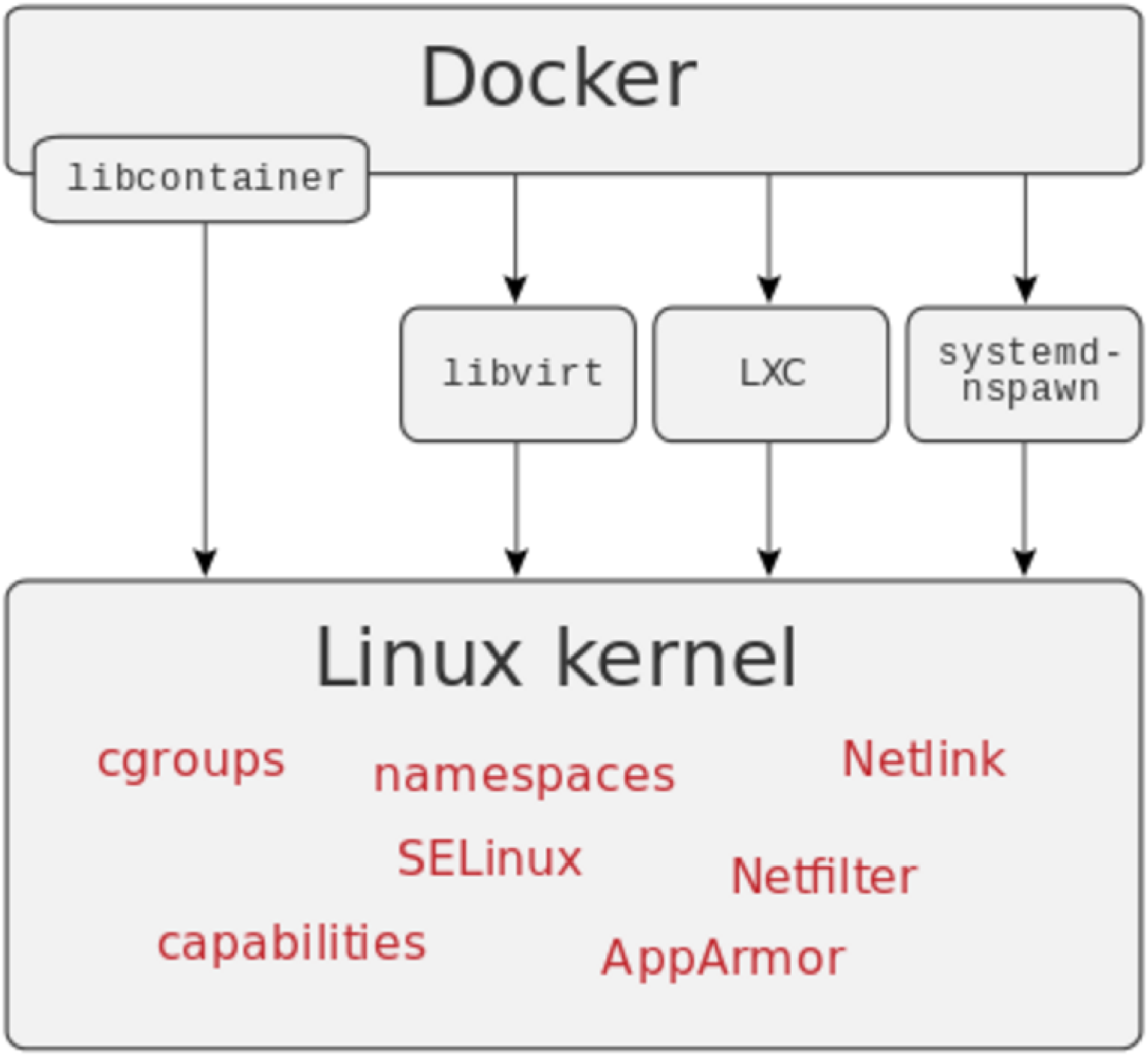
## Virtual machines vs. containers

To put it simply, virtual machines offer isolation at the hardware abstraction layer whereas containers offer OS-level process separation (i.e., hardware virtualization). Therefore, machine virtualization is best suited for IaaS use cases whereas containers are most suited for shipping/packaging portable and modular software. Again, combining the two technologies might be advantageous. For instance, VMs can be used to generate Docker containers, making a solution extremely portable.



## What is Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. This ensures that the application will run reliably on any other machine, regardless of any customized settings that the machine might have that could differ from the machine used for writing and testing the code. Docker provides a way to run these containers in a consistent environment, allowing for further portability and better resource utilization on a host machine.



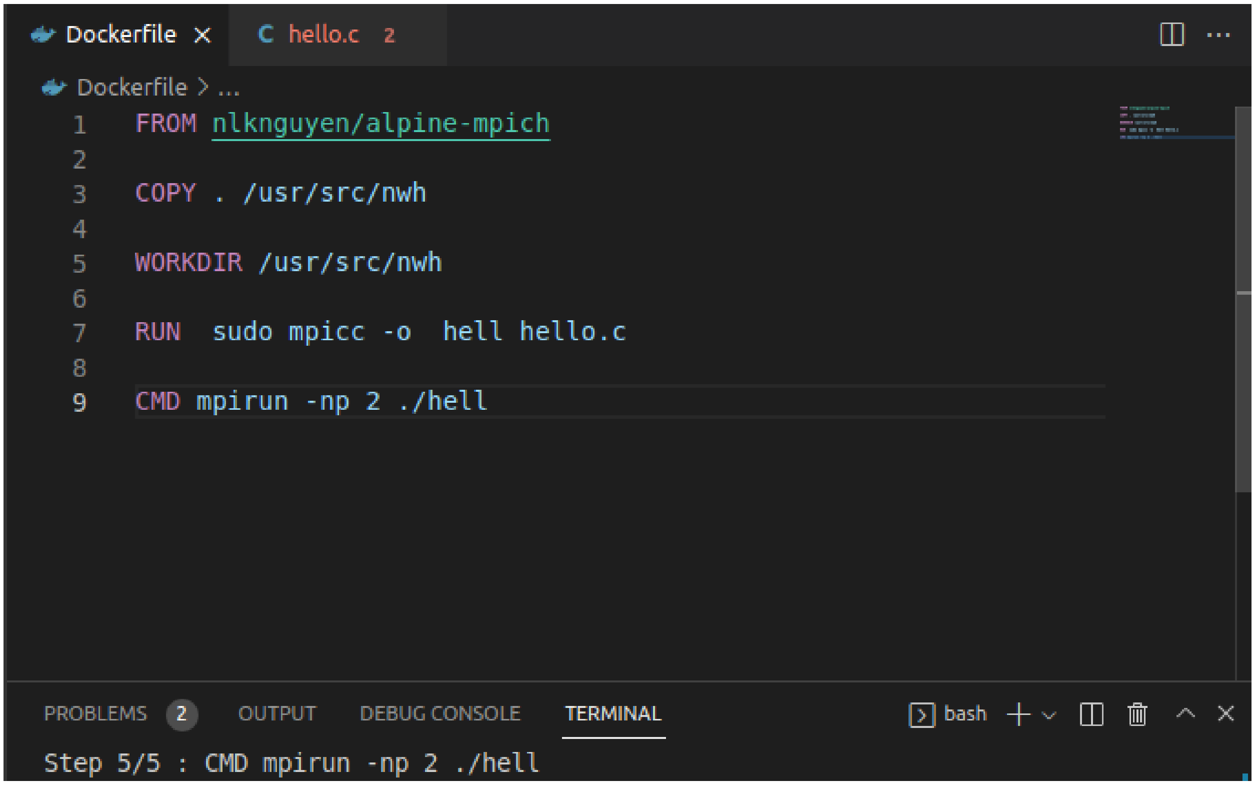
## Docker vs Vmware

Docker and VMware are both tools that can be used to create and manage virtual environments, but they differ in some important ways. Docker uses containers, which are a lightweight way to package and distribute applications. Containers allow for more efficient resource utilization and greater portability because they don't require a full operating system to run. VMware, on the other hand, uses virtual machines (VMs), which are essentially replicas of physical computers that run on top of a host operating system. VMs are typically more heavyweight and require more resources than containers, but they also offer more flexibility and isolation.

### Is Docker Faster Than VMware?

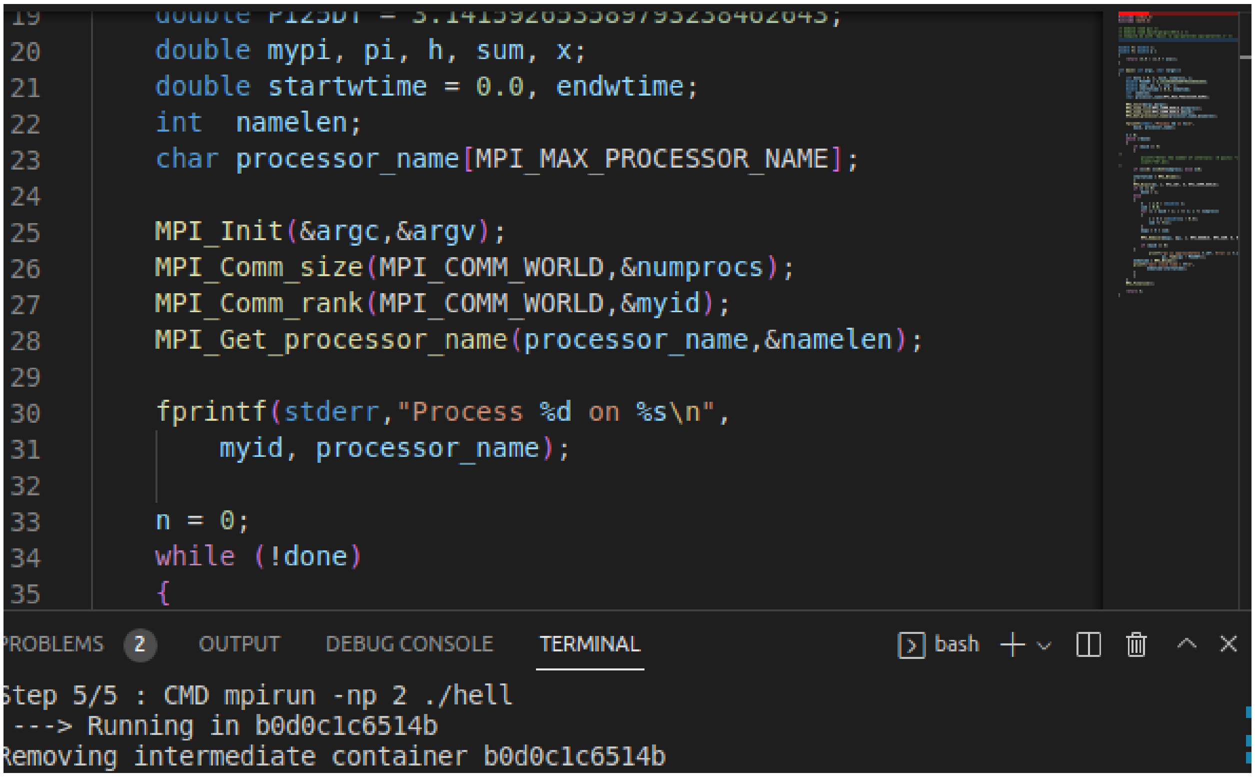
In general, Docker can be faster than VMware because containers are more lightweight and require fewer resources to run than VMs. This can make Docker a better choice for applications that need to be scaled quickly or run on limited hardware. However, this is not always the case, and the performance of Docker and VMware will depend on a variety of factors, including the specific applications and workloads being run and the underlying hardware. In some cases, VMware may be able to provide better performance, especially for applications that require a high degree of isolation or that have specific hardware requirements. It's important to evaluate the specific needs of your applications and choose the best solution for your use case.

# GENERATING PI AND IT’s ERROR



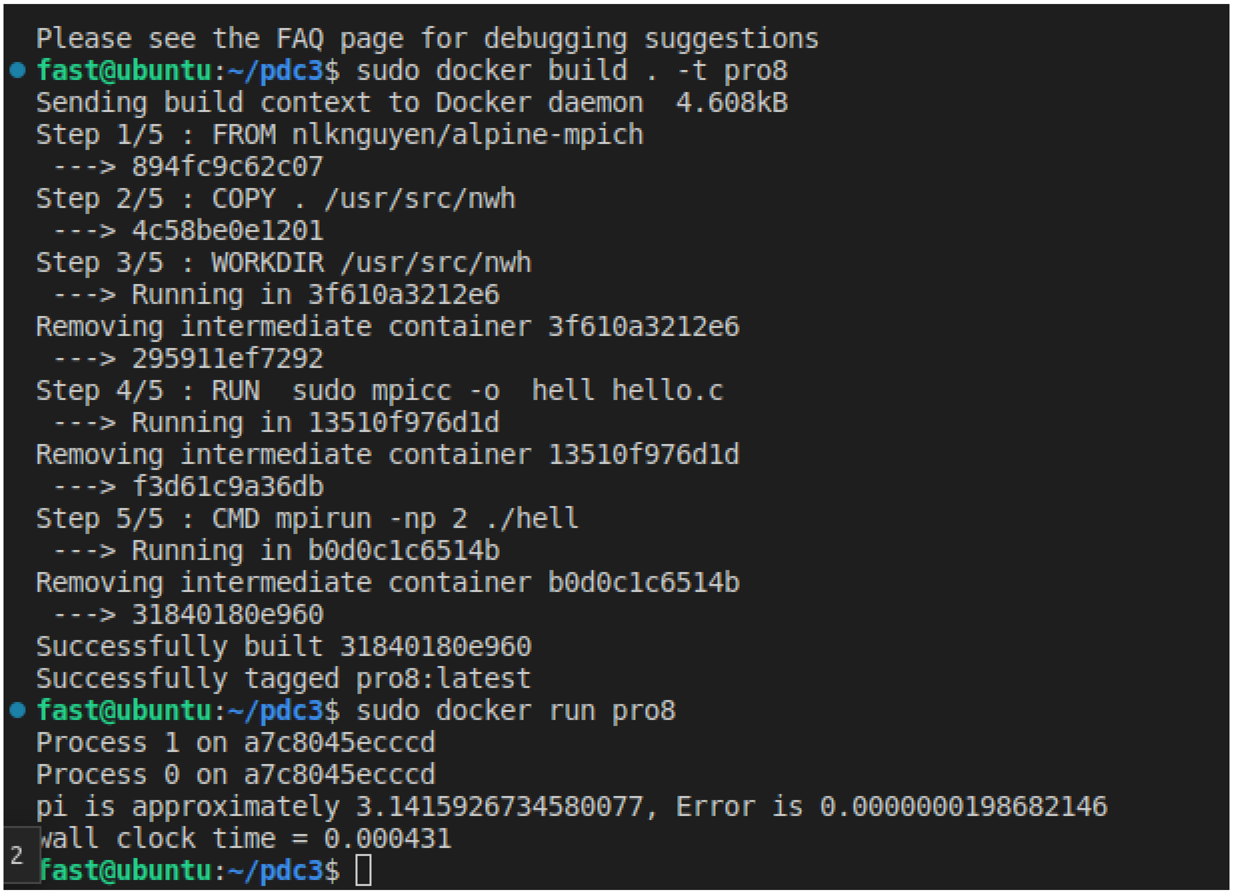
```
FROM nlknguyen/alpine-mpich

COPY . /usr/src/nwh

WORKDIR /usr/src/nwh

RUN  sudo mpicc -o  hell hello.c

CMD mpirun -np 2 ./hell
```

```
Step 5/5 : CMD mpirun -np 2 ./hell
```



```
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n",
        myid, processor_name);

    n = 0;
    while (!done)
    {
```

```
Step 5/5 : CMD mpirun -np 2 ./hell
 ---> Running in b0d0c1c6514b
Removing intermediate container b0d0c1c6514b
```



```
Please see the FAQ page for debugging suggestions
fast@ubuntu:~/pdc3$ sudo docker build . -t pro8
Sending build context to Docker daemon  4.608kB
Step 1/5 : FROM nlknguyen/alpine-mpich
 ---> 894fc9c62c07
Step 2/5 : COPY . /usr/src/nwh
 ---> 4c58be0e1201
Step 3/5 : WORKDIR /usr/src/nwh
 ---> Running in 3f610a3212e6
Removing intermediate container 3f610a3212e6
 ---> 295911ef7292
Step 4/5 : RUN  sudo mpicc -o  hell hello.c
 ---> Running in 13510f976d1d
Removing intermediate container 13510f976d1d
 ---> f3d61c9a36db
Step 5/5 : CMD mpirun -np 2 ./hell
 ---> Running in b0d0c1c6514b
Removing intermediate container b0d0c1c6514b
 ---> 31840180e960
Successfully built 31840180e960
Successfully tagged pro8:latest
fast@ubuntu:~/pdc3$ sudo docker run pro8
Process 1 on a7c8045ecccd
Process 0 on a7c8045ecccd
pi is approximately 3.1415926734580077, Error is 0.0000000198682146
wall clock time = 0.000431
fast@ubuntu:~/pdc3$ 
```

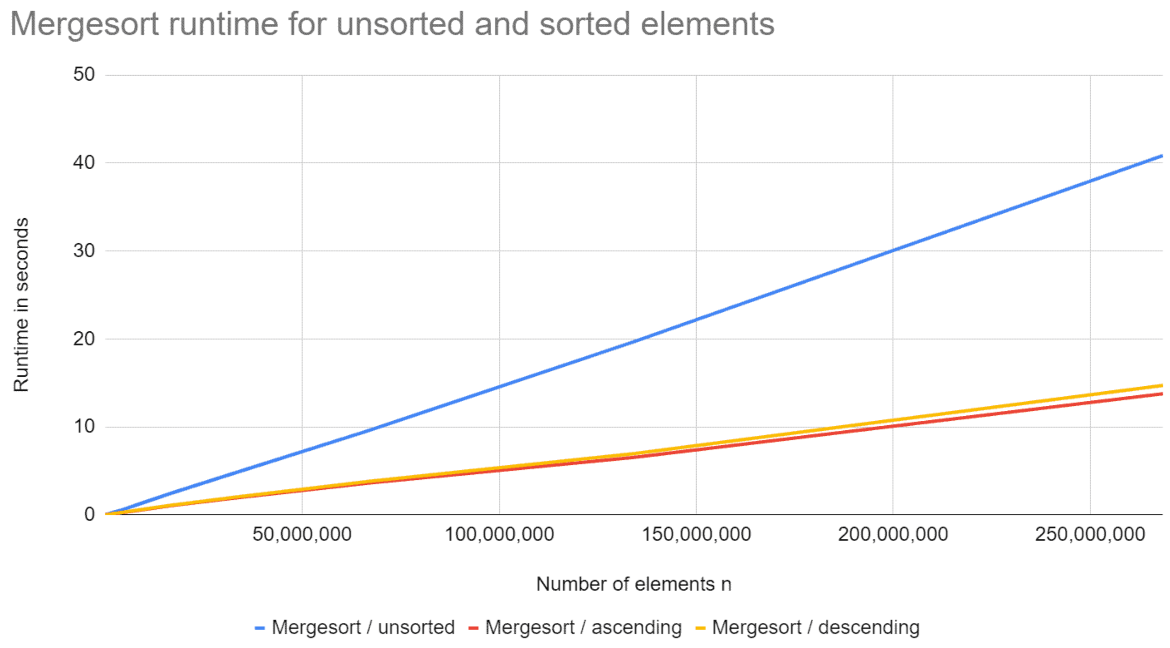
Code

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a )
{
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
```

```
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  MPI_Get_processor_name(processor_name,&namelen);

  fprintf(stderr,"Process %d on %s\n",
        myid, processor_name);

  n = 0;
  while (!done)
  {
    if (myid == 0)
    {
/*
      printf("Enter the number of intervals: (0 quits) ");
      scanf("%d",&n);
*/
        if (n==0) n=1024*numprocs; else n=0;

        startwtime = MPI_Wtime();
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0)
      done = 1;
    else
    {
      h   = 1.0 / (double) n;
      sum = 0.0;
      for (i = myid + 1; i <= n; i += numprocs)
      {
        x = h * ((double)i - 0.5);
        sum += f(x);
      }
      mypi = h * sum;

      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

      if (myid == 0)
        {
        printf("pi is approximately %.16f, Error is %.16f\n",
           pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("wall clock time = %f\n",
               endwtime-startwtime);
```

```
                }
        }
    }
    MPI_Finalize();

    return 0;
}
```

# MERGE SORT



```
#include <mpi.h>     // MPI
#include <stdio.h>   // printf
#include <stdlib.h>  // malloc, free, rand(), srand()
#include <time.h>    // time for random generator
#include <math.h>    // log2
#include <string.h>  // memcpy
#include <limits.h>  // INT_MAX

/* Declaration of functions */
```

```
void powerOfTwo(int id, int numberProcesses);
void getInput(int argc, char* argv[], int id, int numProcs, int* arraySize);
void fillArray(int array[], int arraySize, int id);
void printList(int id, char arrayName[], int array[], int arraySize);
int compare(const void* a_p, const void* b_p);
int* merge(int half1[], int half2[], int mergeResult[], int size);
int* mergeSort(int height, int id, int localArray[], int size, MPI_Comm comm, int
globalArray[]);

/*-------------------------------------------------------------------
 * Function:   powerOfTwo
 * Purpose:    Check number of processes, if not power of 2 prints message
 * Params:     id, rank of the current process
 *                     numberProcesses, number of processes
 */

void powerOfTwo(int id, int numberProcesses) {
        int power;
        power = (numberProcesses != 0) && ((numberProcesses &
(numberProcesses - 1)) == 0);
        if (!power) {
                if (id == 0) printf("number of processes must be power of 2 \n");
                MPI_Finalize();
                exit(-1);
        }
}

/*-------------------------------------------------------------------
 * Function:   getInput
 * Purpose:    Get input from user for array size
 * Params:     argc, argument count
 *                     argv[], points to argument vector
 *              id, rank of the current process
 *                     numProcs, number of processes
 *                     arraySize, points to array size
 */
```

```
void getInput(int argc, char* argv[], int id, int numProcs, int* arraySize){
    if (id == 0){
        if (id % 2 != 0){
                              fprintf(stderr, "usage: mpirun -n <p> %s <size of array> \n",
argv[0]);
            fflush(stderr);
            *arraySize = -1;
        } else if (argc != 2){
            fprintf(stderr, "usage: mpirun -n <p> %s <size of array> \n", argv[0]);
            fflush(stderr);
            *arraySize = -1;
        } else if ((atoi(argv[1])) % numProcs != 0) {
                        fprintf(stderr, "size of array must be divisible by number of
processes \n");
            fflush(stderr);
            *arraySize = -1;
                    } else {
            *arraySize = atoi(argv[1]);
        }
    }
    // broadcast arraySize to all processes
    MPI_Bcast(arraySize, 1, MPI_INT, 0, MPI_COMM_WORLD);

    // negative arraySize ends the program
    if (*arraySize <= 0) {
        MPI_Finalize();
        exit(-1);
    }
}

/*-------------------------------------------------------------------
 * Function:   fillArray
 * Purpose:    Fill array with random integers
 * Params:     array, the array being filled
 *              arraySize, size of the array
 *              id, rank of the current process
 */
```

```
void fillArray(int array[], int arraySize, int id) {
        int i;
        // use current time as seed for random generator
        srand(id + time(0));
        for (i = 0; i < 100000; i++) {
                array[i] = rand() % 100; //INT_MAX
        }
}

/*------------------------------------------------------------------
 * Function:   printList
 * Purpose:    Prints the contents of a given list of a process
 * Params:     id, rank of the current process
 *                       arrayName, name of array
 *              array, array to print
 *              arraySize, size of array
 */

void printList(int id, char arrayName[], int array[], int arraySize) {
    printf("Process %d, %s: ", id, arrayName);
    for (int i = 0; i < arraySize; i++) {
        printf(" %d", array[i]);
    }
    printf("\n");
}

/*------------------------------------------------------------------
 * Function:    Compare - An Introduction to Parallel Programming by Pacheco
 * Purpose:     Compare 2 ints, return -1, 0, or 1, respectively, when
 *          the first int is less than, equal, or greater than
 *          the second.  Used by qsort.
 */

int compare(const void* a_p, const void* b_p) {
   int a = *((int*)a_p);
   int b = *((int*)b_p);
```

```
  if (a < b)
    return -1;
  else if (a == b)
    return 0;
  else /* a > b */
    return 1;
}

/*-----------------------------------------------------------------
 * Function:    merge
 * Purpose:     Merges half1 array and half2 array into mergeResult
 * Params:      half1, first half of array to merge
 *                   half2, second half of array to merge
 *                          mergeResult, array to store merged result
 *                          size, size of half1 and half2
 */
int* merge(int half1[], int half2[], int mergeResult[], int size){
   int ai, bi, ci;
   ai = bi = ci = 0;
   // integers remain in both arrays to compare
   while ((ai < size) && (bi < size)){
      if (half1[ai] <= half2[bi]){
                    mergeResult[ci] = half1[ai];
                    ai++;
             } else {
                    mergeResult[ci] = half2[bi];
                    bi++;
             }
                    ci++;
       }
       // integers only remain in rightArray
       if (ai >= size){
      while (bi < size) {
                    mergeResult[ci] = half2[bi];
                    bi++; ci++;
             }
```

```
        }
        // integers only remain in localArray
        if (bi >= size){
                while (ai < size) {
                        mergeResult[ci] = half1[ai];
                        ai++; ci++;
                }
        }
        return mergeResult;
}

/*-----------------------------------------------------------------
 * Function:    mergeSort
 * Purpose:     implements merge sort: merges sorted arrays from
 *            processes until we have a single array containing all
 *                      integers in sorted order
 * Params:              height, height of merge sort tree
 *                              id, rank of the current process
 *                              localArray, local array containing integers of current
process
 *                              size, size of localArray on current process
 *                              comm, MPI communicator
 *                              globalArray, globalArray contains either all integers
 *                              if process 0 or NULL for other processes
 */

int* mergeSort(int height, int id, int localArray[], int size, MPI_Comm comm, int
globalArray[]){
   int parent, rightChild, myHeight;
   int *half1, *half2, *mergeResult;

   myHeight = 0;
   qsort(localArray, size, sizeof(int), compare); // sort local array
   half1 = localArray;  // assign half1 to localArray

   while (myHeight < height) { // not yet at top
      parent = (id & (~(1 << myHeight)));
```

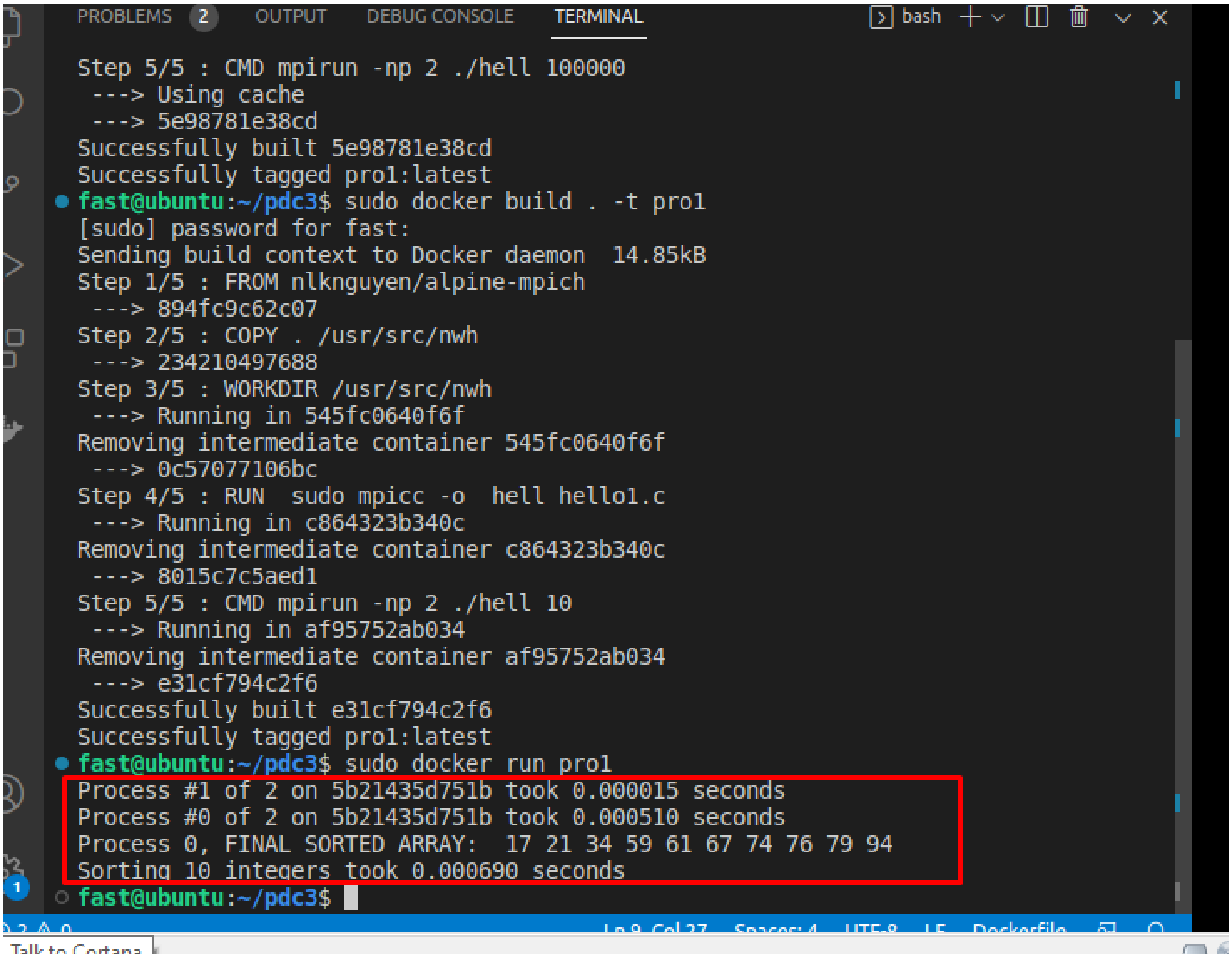
```
        if (parent == id) { // left child
                        rightChild = (id | (1 << myHeight));

                        // allocate memory and receive array of right child
                        half2 = (int*) malloc (size * sizeof(int));
                        MPI_Recv(half2, size, MPI_INT, rightChild, 0,
                                        MPI_COMM_WORLD, MPI_STATUS_IGNORE);

                        // allocate memory for result of merge
                        mergeResult = (int*) malloc (size * 2 * sizeof(int));
                        // merge half1 and half2 into mergeResult
                        mergeResult = merge(half1, half2, mergeResult, size);
                        // reassign half1 to merge result
            half1 = mergeResult;
                              size = size * 2;  // double size

                              free(half2);
                              mergeResult = NULL;

            myHeight++;

        } else { // right child
                                // send local array to parent
              MPI_Send(half1, size, MPI_INT, parent, 0, MPI_COMM_WORLD);
              if(myHeight != 0) free(half1);
              myHeight = height;
        }
    }

    if(id == 0){
                    globalArray = half1;   // reassign globalArray to half1
          }
          return globalArray;
}

/*---------------------------------------------------------------*/
```

```
int main(int argc, char** argv) {
   int numProcs, id, globalArraySize, localArraySize, height;
   int *localArray, *globalArray;
   double startTime, localTime, totalTime;
   double zeroStartTime, zeroTotalTime, processStartTime, processTotalTime;;
   int length = -1;
   char myHostName[MPI_MAX_PROCESSOR_NAME];

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
   MPI_Comm_rank(MPI_COMM_WORLD, &id);

   MPI_Get_processor_name (myHostName, &length);

   // check for odd processes
   powerOfTwo(id, numProcs);

   // get size of global array
   getInput(argc, argv, id, numProcs, &globalArraySize);

   // calculate total height of tree
   height = log2(numProcs);

   // if process 0, allocate memory for global array and fill with values
   if (id==0){
           globalArray = (int*) malloc (globalArraySize * sizeof(int));
           fillArray(globalArray, globalArraySize, id);
           //printList(id, "UNSORTED ARRAY", globalArray, globalArraySize);
// Line A
       }

   // allocate memory for local array, scatter to fill with values and print
   localArraySize = globalArraySize / numProcs;
   localArray = (int*) malloc (localArraySize * sizeof(int));
   MPI_Scatter(globalArray, localArraySize, MPI_INT, localArray,
           localArraySize, MPI_INT, 0, MPI_COMM_WORLD);
```

```
    //printList(id, "localArray", localArray, localArraySize);   // Line B

    //Start timing
    startTime = MPI_Wtime();
    //Merge sort
    if (id == 0) {
                zeroStartTime = MPI_Wtime();
                globalArray = mergeSort(height, id, localArray, localArraySize,
MPI_COMM_WORLD, globalArray);
                zeroTotalTime = MPI_Wtime() - zeroStartTime;
                printf("Process #%d of %d on %s took %f seconds \n",
                        id, numProcs, myHostName, zeroTotalTime);
        }
        else {
                processStartTime = MPI_Wtime();
              mergeSort(height, id, localArray, localArraySize,
MPI_COMM_WORLD, NULL);
                processTotalTime = MPI_Wtime() - processStartTime;
                printf("Process #%d of %d on %s took %f seconds \n",
                        id, numProcs, myHostName, processTotalTime);
        }
    //End timing
    localTime = MPI_Wtime() - startTime;
    MPI_Reduce(&localTime, &totalTime, 1, MPI_DOUBLE,
        MPI_MAX, 0, MPI_COMM_WORLD);

    if (id == 0) {
                printList(0, "FINAL SORTED ARRAY", globalArray, globalArraySize);
// Line C
                printf("Sorting %d integers took %f seconds \n",
globalArraySize,totalTime);
                free(globalArray);
        }

    free(localArray);
    MPI_Finalize();
    return 0;
```

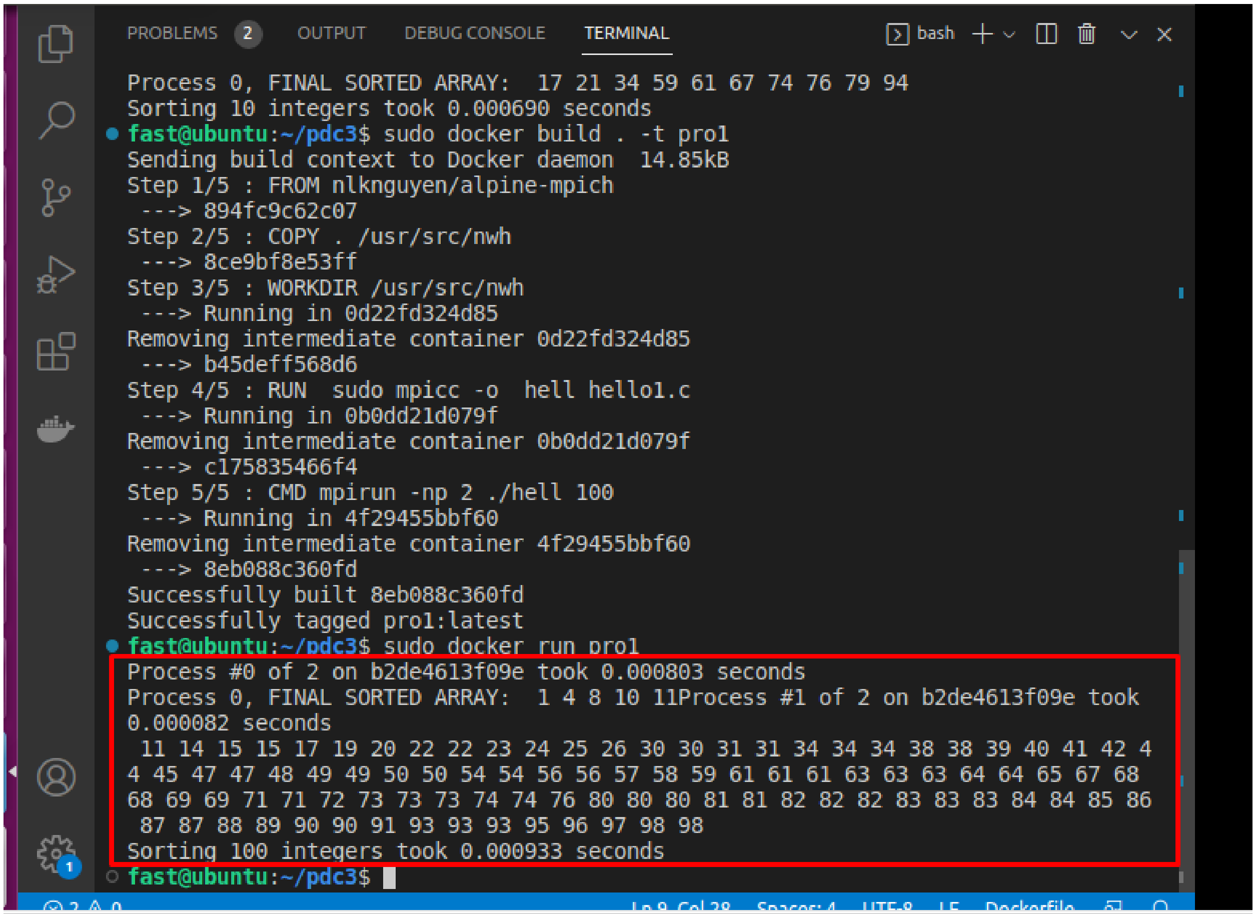
}

## 10 Integers



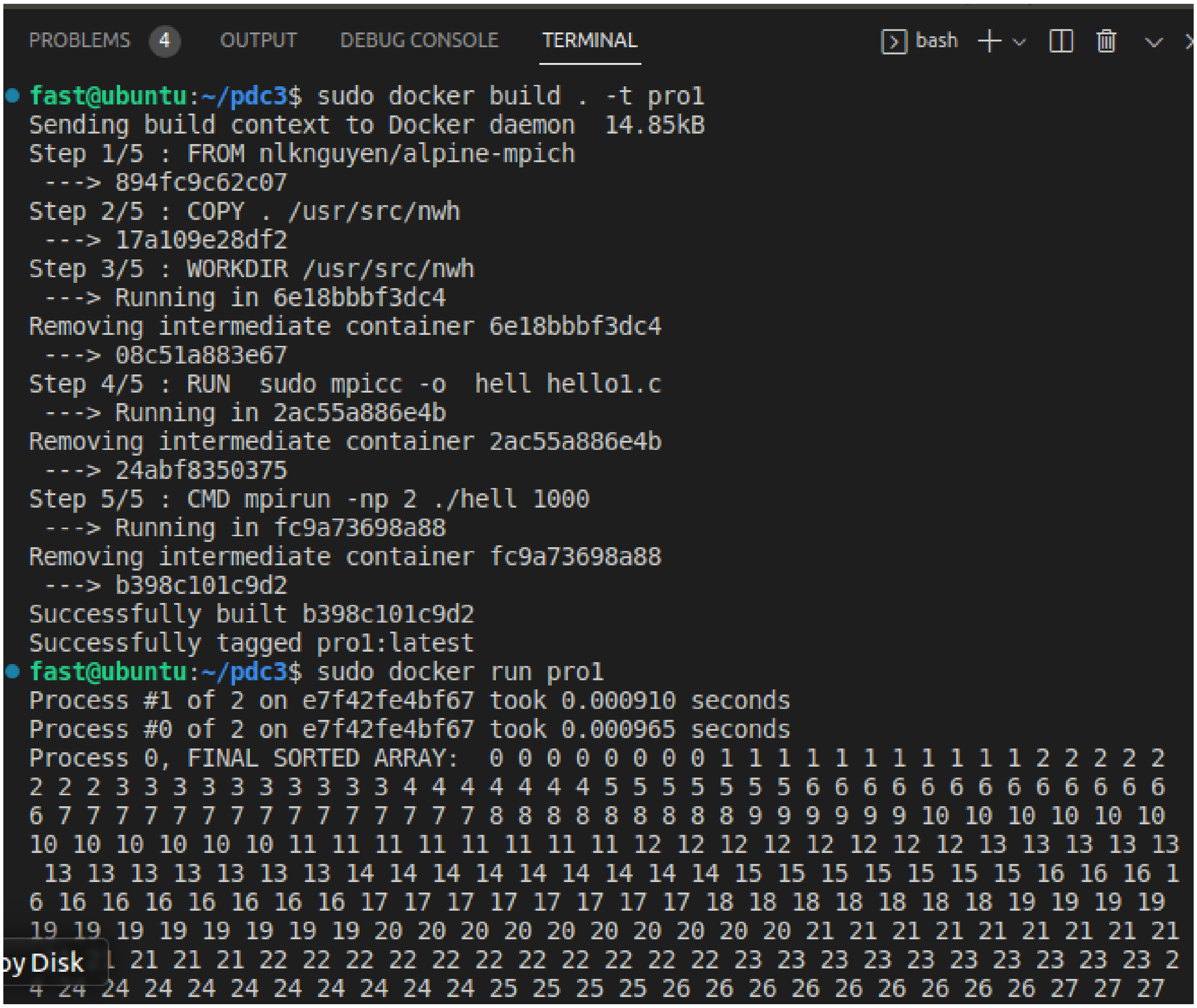
```
Step 5/5 : CMD mpirun -np 2 ./hell 100000
 ---> Using cache
 ---> 5e98781e38cd
Successfully built 5e98781e38cd
Successfully tagged pro1:latest
fast@ubuntu:~/pdc3$ sudo docker build . -t pro1
[sudo] password for fast:
Sending build context to Docker daemon  14.85kB
Step 1/5 : FROM nlknguyen/alpine-mpich
 ---> 894fc9c62c07
Step 2/5 : COPY . /usr/src/nwh
 ---> 234210497688
Step 3/5 : WORKDIR /usr/src/nwh
 ---> Running in 545fc0640f6f
Removing intermediate container 545fc0640f6f
 ---> 0c57077106bc
Step 4/5 : RUN  sudo mpicc -o  hell hello1.c
 ---> Running in c864323b340c
Removing intermediate container c864323b340c
 ---> 8015c7c5aed1
Step 5/5 : CMD mpirun -np 2 ./hell 10
 ---> Running in af95752ab034
Removing intermediate container af95752ab034
 ---> e31cf794c2f6
Successfully built e31cf794c2f6
Successfully tagged pro1:latest
fast@ubuntu:~/pdc3$ sudo docker run pro1
Process #1 of 2 on 5b21435d751b took 0.000015 seconds
Process #0 of 2 on 5b21435d751b took 0.000510 seconds
Process 0, FINAL SORTED ARRAY:  17 21 34 59 61 67 74 76 79 94
Sorting 10 integers took 0.000690 seconds
fast@ubuntu:~/pdc3$
```

## 100 Integers

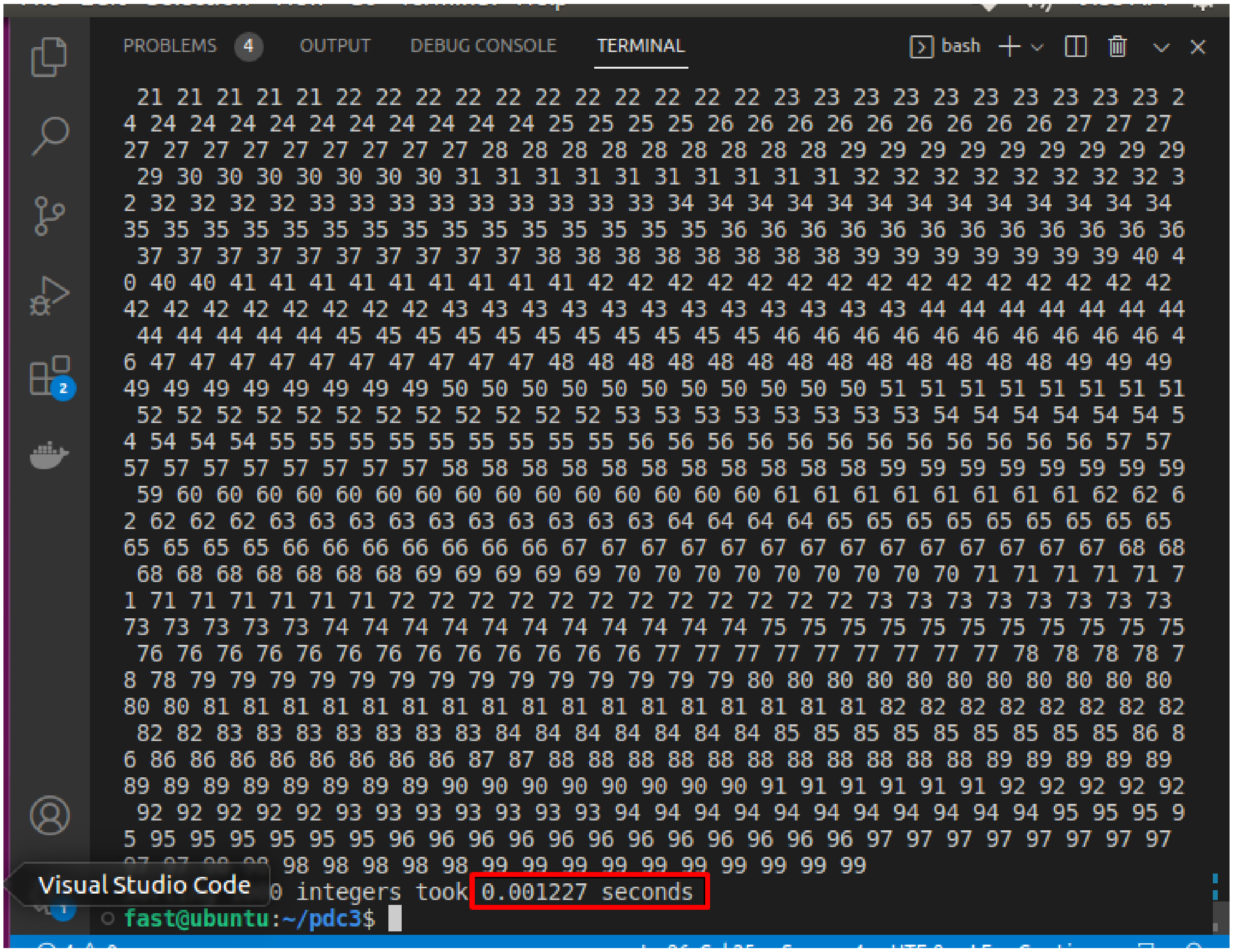


```
Process 0, FINAL SORTED ARRAY:  17 21 34 59 61 67 74 76 79 94
Sorting 10 integers took 0.000690 seconds
fast@ubuntu:~/pdc3$ sudo docker build . -t pro1
Sending build context to Docker daemon  14.85kB
Step 1/5 : FROM nlknguyen/alpine-mpich
 ---> 894fc9c62c07
Step 2/5 : COPY . /usr/src/nwh
 ---> 8ce9bf8e53ff
Step 3/5 : WORKDIR /usr/src/nwh
 ---> Running in 0d22fd324d85
Removing intermediate container 0d22fd324d85
 ---> b45deff568d6
Step 4/5 : RUN  sudo mpicc -o  hell hello1.c
 ---> Running in 0b0dd21d079f
Removing intermediate container 0b0dd21d079f
 ---> c175835466f4
Step 5/5 : CMD mpirun -np 2 ./hell 100
 ---> Running in 4f29455bbf60
Removing intermediate container 4f29455bbf60
 ---> 8eb088c360fd
Successfully built 8eb088c360fd
Successfully tagged pro1:latest
fast@ubuntu:~/pdc3$ sudo docker run pro1
Process #0 of 2 on b2de4613f09e took 0.000803 seconds
Process 0, FINAL SORTED ARRAY:  1 4 8 10 11Process #1 of 2 on b2de4613f09e took
0.000082 seconds
 11 14 15 15 17 19 20 22 22 23 24 25 26 30 30 31 31 34 34 34 38 38 39 40 41 42 4
4 45 47 47 48 49 49 50 50 54 54 56 56 57 58 59 61 61 61 63 63 63 64 64 65 67 68
68 69 69 71 71 72 73 73 73 74 74 76 80 80 80 81 81 82 82 82 83 83 83 84 84 85 86
 87 87 88 89 90 90 91 93 93 93 95 96 97 98 98
Sorting 100 integers took 0.000933 seconds
fast@ubuntu:~/pdc3$ 
```

## 1000 Integers



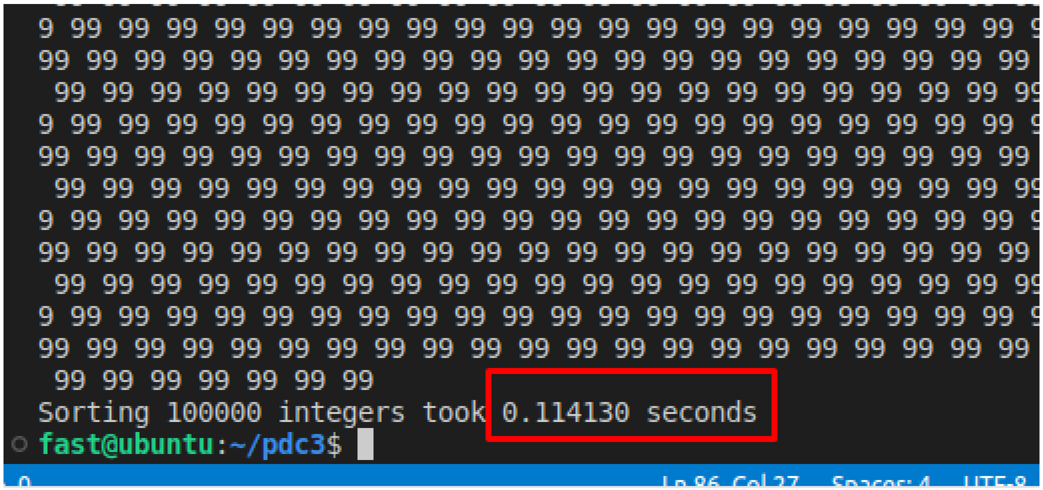
```
● fast@ubuntu:~/pdc3$ sudo docker build . -t pro1
  Sending build context to Docker daemon  14.85kB
  Step 1/5 : FROM nlknguyen/alpine-mpich
   ---> 894fc9c62c07
  Step 2/5 : COPY . /usr/src/nwh
   ---> 17a109e28df2
  Step 3/5 : WORKDIR /usr/src/nwh
   ---> Running in 6e18bbbf3dc4
  Removing intermediate container 6e18bbbf3dc4
   ---> 08c51a883e67
  Step 4/5 : RUN  sudo mpicc -o  hell hello1.c
   ---> Running in 2ac55a886e4b
  Removing intermediate container 2ac55a886e4b
   ---> 24abf8350375
  Step 5/5 : CMD mpirun -np 2 ./hell 1000
   ---> Running in fc9a73698a88
  Removing intermediate container fc9a73698a88
   ---> b398c101c9d2
  Successfully built b398c101c9d2
  Successfully tagged pro1:latest
● fast@ubuntu:~/pdc3$ sudo docker run pro1
  Process #1 of 2 on e7f42fe4bf67 took 0.000910 seconds
  Process #0 of 2 on e7f42fe4bf67 took 0.000965 seconds
  Process 0, FINAL SORTED ARRAY:  0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
  2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6
  6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 10 10 10 10 10 10
  10 10 10 10 10 10 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 12 13 13 13 13 13
   13 13 13 13 13 13 13 14 14 14 14 14 14 14 14 14 15 15 15 15 15 15 15 16 16 16 1
  6 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 19 19 19 19
  19 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 21 21 21 21 21 21 21 21 21
py Disk 1 21 21 21 22 22 22 22 22 22 22 22 22 22 22 23 23 23 23 23 23 23 23 23 23 2
  4 24 24 24 24 24 24 24 24 24 24 25 25 25 25 26 26 26 26 26 26 26 26 27 27 27
```



```
 21 21 21 21 21 22 22 22 22 22 22 22 22 22 22 22 23 23 23 23 23 23 23 23 23 23 2
4 24 24 24 24 24 24 24 24 24 24 25 25 25 25 26 26 26 26 26 26 26 26 26 27 27 27
27 27 27 27 27 27 27 27 27 28 28 28 28 28 28 28 28 28 29 29 29 29 29 29 29 29 29
 29 30 30 30 30 30 30 30 31 31 31 31 31 31 31 31 31 31 32 32 32 32 32 32 32 32 3
2 32 32 32 32 33 33 33 33 33 33 33 33 33 34 34 34 34 34 34 34 34 34 34 34 34 34
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 36 36 36 36 36 36 36 36 36 36 36 36
 37 37 37 37 37 37 37 37 37 37 38 38 38 38 38 38 38 38 39 39 39 39 39 39 39 40 4
0 40 40 41 41 41 41 41 41 41 41 41 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
42 42 42 42 42 42 42 42 43 43 43 43 43 43 43 43 43 43 43 43 44 44 44 44 44 44 44
 44 44 44 44 44 45 45 45 45 45 45 45 45 45 45 45 46 46 46 46 46 46 46 46 46 46 4
6 47 47 47 47 47 47 47 47 47 47 48 48 48 48 48 48 48 48 48 48 48 48 48 49 49 49
49 49 49 49 49 49 49 49 50 50 50 50 50 50 50 50 50 50 50 51 51 51 51 51 51 51 51
 52 52 52 52 52 52 52 52 52 52 52 52 53 53 53 53 53 53 53 53 54 54 54 54 54 54 5
4 54 54 54 55 55 55 55 55 55 55 55 55 56 56 56 56 56 56 56 56 56 56 56 56 57 57
57 57 57 57 57 57 57 57 58 58 58 58 58 58 58 58 58 58 58 59 59 59 59 59 59 59 59
 59 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 61 61 61 61 61 61 61 61 62 62 6
2 62 62 62 63 63 63 63 63 63 63 63 63 63 64 64 64 64 65 65 65 65 65 65 65 65 65
65 65 65 65 66 66 66 66 66 66 66 67 67 67 67 67 67 67 67 67 67 67 67 67 67 68 68
 68 68 68 68 68 68 68 69 69 69 69 69 70 70 70 70 70 70 70 70 70 71 71 71 71 71 7
1 71 71 71 71 71 71 72 72 72 72 72 72 72 72 72 72 72 72 73 73 73 73 73 73 73 73
73 73 73 73 73 74 74 74 74 74 74 74 74 74 74 75 75 75 75 75 75 75 75 75 75 75 75
 76 76 76 76 76 76 76 76 76 76 76 76 76 77 77 77 77 77 77 77 77 77 78 78 78 78 7
8 78 79 79 79 79 79 79 79 79 79 79 79 79 79 79 80 80 80 80 80 80 80 80 80 80 80
80 80 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 82 82 82 82 82 82 82 82
 82 82 83 83 83 83 83 83 83 84 84 84 84 84 84 84 85 85 85 85 85 85 85 85 85 86 8
6 86 86 86 86 86 86 86 86 87 87 88 88 88 88 88 88 88 88 88 88 88 89 89 89 89 89
89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 91 91 91 91 91 91 92 92 92 92 92
 92 92 92 92 92 93 93 93 93 93 93 93 94 94 94 94 94 94 94 94 94 94 94 95 95 95 9
5 95 95 95 95 95 95 96 96 96 96 96 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97
97 97 98 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99
0 integers took 0.001227 seconds
fast@ubuntu:~/pdc3$
```

Time = 0.001227 Sec

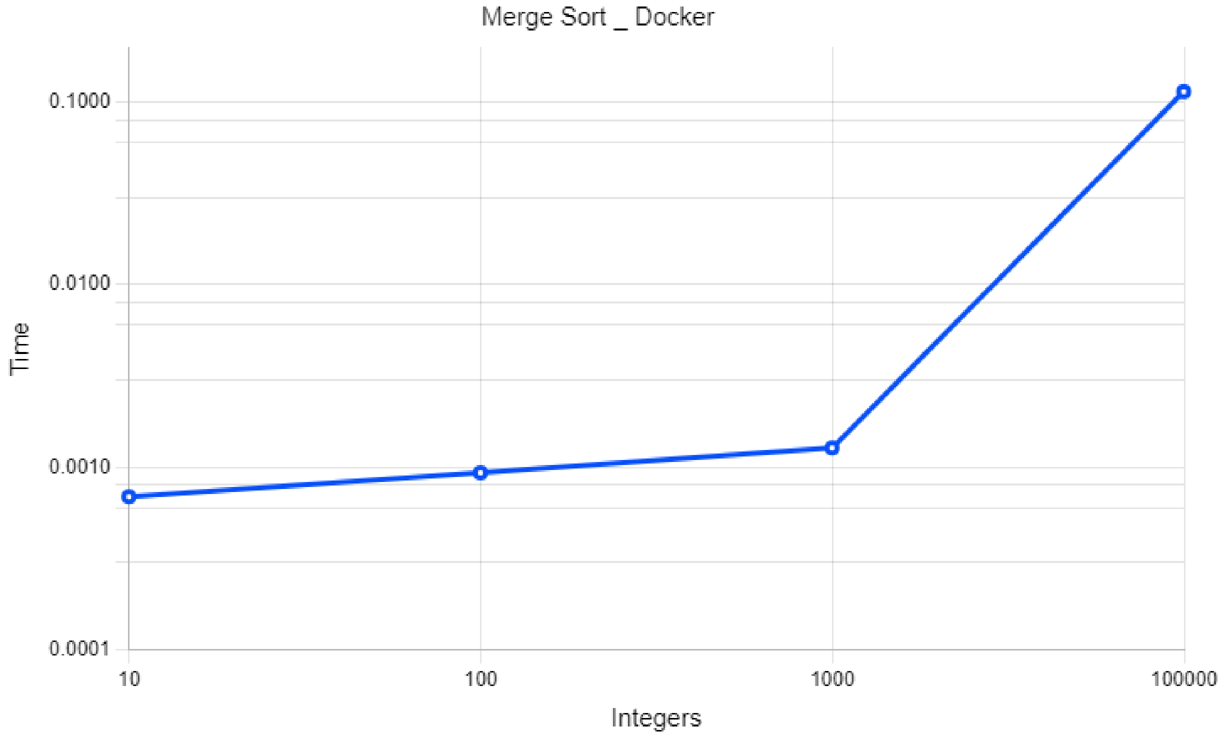
Using
100,000 Integers



```
9 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 9
99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99
 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99
9 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 9
99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99
 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99
9 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 9
99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99
 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99
9 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 9
99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99
 99 99 99 99 99 99 99 99
Sorting 100000 integers took 0.114130 seconds
fast@ubuntu:~/pdc3$
```

Time = 0.114130 Secs

## Graph

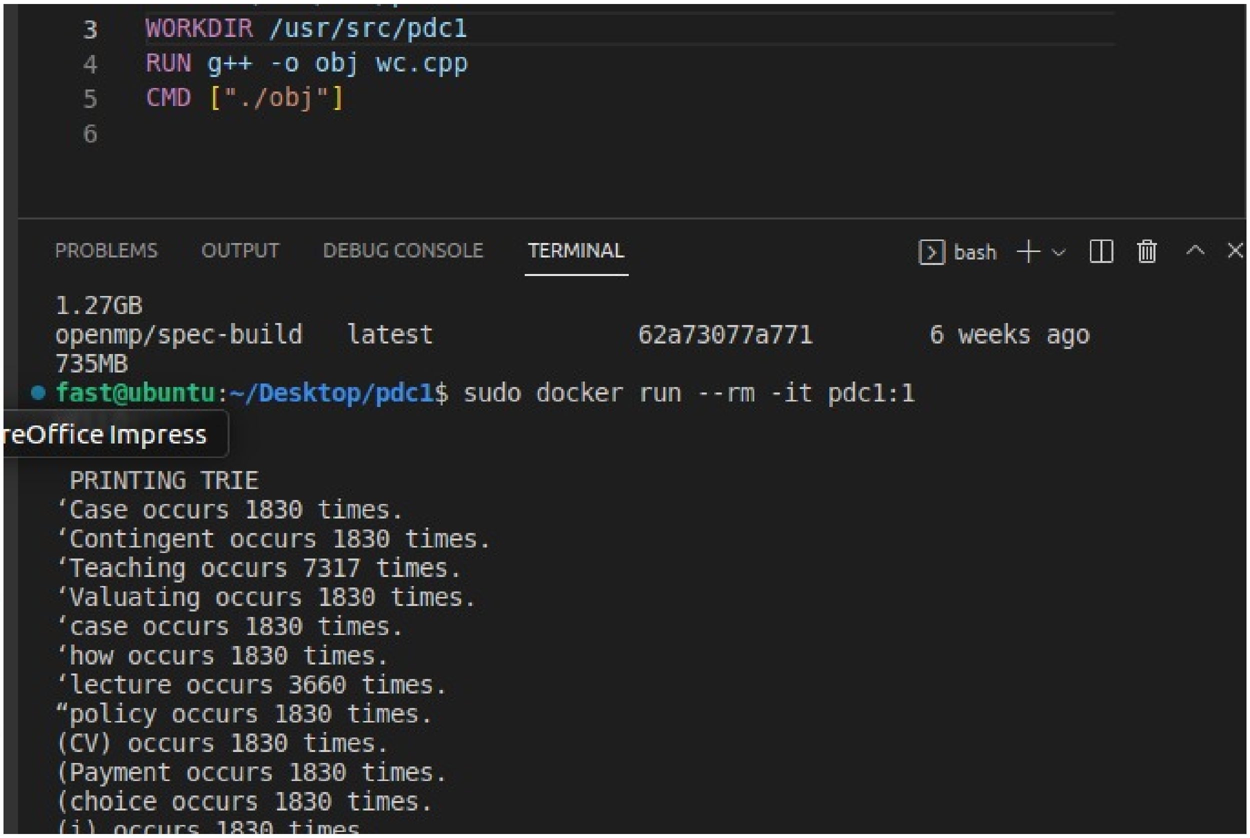


Program to produce word count
I helped the other group and did this code with them hence pasted it here because I didnot have any group members and I did all of above codes myself and following with the other group/helped them. Donot please penalize on basis of plagiarism.

```
#include<iostream>
#include<cstdio>
#include<string>
#include<vector>
#include<algorithm>
#include<cmath>
#include<map>
#include<string.h>
#include<fstream>
#include<omp.h>
#include<time.h>

using namespace std;

struct TrieNode
{
	map<char,TrieNode*> children;
	int occurance;
	bool endofword;

	TrieNode()
	{
		occurance=0;
```

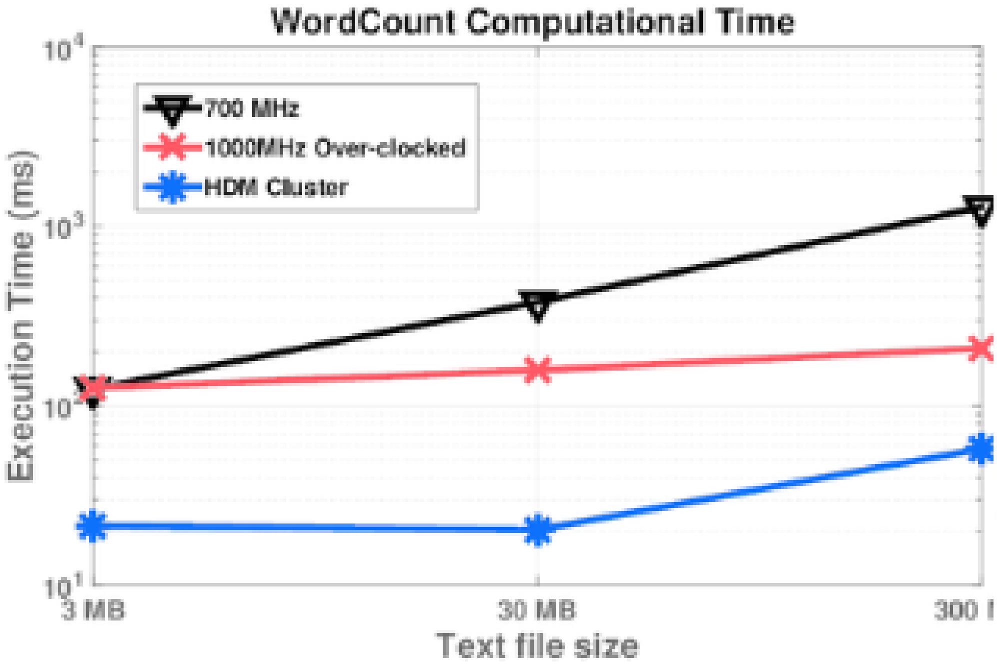
```
                endofword=false;
        }
};

void insert(TrieNode *root,string word)
{

        TrieNode *current=root;

        for(int i=0;i<word.size();i++)
        {
                char ch=word[i];
                TrieNode *node=current->children[ch];
                if(!node)
                {
                        node=new TrieNode();
                        current->children[word[i]]=node;
                }
                current=node; // like temp = temp -> next    starting current
will be at root then at next character
        }
        current->endofword=true;
        current->occurance++;
}
void printWords(TrieNode* crnt,string word)
{
        //cout<<word<<endl;
    if(crnt->endofword==true)
    {
       cout<<word<<" occurs "<<crnt->occurance<<" times."<<endl;
    }
    map<char,TrieNode*>::iterator it;
    it=crnt->children.begin();
    while(it!=crnt->children.end())
    {
        word += it->first;
        printWords(it->second,word);
        word.erase(word.length()-1);  //erase last prefix character
        it++;

    }
}



void insertion(TrieNode* root,char* filename){



        FILE *ptr_readfile;

        string word; /* or some other suitable maximum line size */

        ifstream infile;
    infile.open(filename);
        while (infile >> word)
    {
        insert(root,word);
    }

        infile.close();

}
```

```
int main()
{
    TrieNode *root=new TrieNode();
    char* file_names[5]={"file_part1","file_part2","file_part3"};
    clock_t t;
  t = clock();
    #pragma omp parallel num_threads(3) shared(root)
    {
        #pragma omp sections
        {
            #pragma omp section
                insertion(root,file_names[0]);
            #pragma omp section
                insertion(root,file_names[1]);
            #pragma omp section
                insertion(root,file_names[2]);

        }
    }

    cout<<"HELLO:"<<endl;
    cout<<"\n PRINTING TRIE"<<endl;
    printWords(root , "");
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC;
    cout<<endl<<"Elapsed time = "<<time_taken<<"
seconds"<<endl<<endl;
    return 0;
}
```



```
WORKDIR /usr/src/pdc1
RUN g++ -o obj wc.cpp
CMD ["./obj"]
```

```
1.27GB
openmp/spec-build   latest            62a73077a771        6 weeks ago
735MB
fast@ubuntu:~/Desktop/pdc1$ sudo docker run --rm -it pdc1:1

 PRINTING TRIE
‘Case occurs 1830 times.
‘Contingent occurs 1830 times.
‘Teaching occurs 7317 times.
‘Valuating occurs 1830 times.
‘case occurs 1830 times.
‘how occurs 1830 times.
‘lecture occurs 3660 times.
“policy occurs 1830 times.
(CV) occurs 1830 times.
(Payment occurs 1830 times.
(choice occurs 1830 times.
(i) occurs 1830 times
```



## Conclusion:

Docker containers can be faster to start and stop than VMs, because they share the host operating system kernel and do not need to boot a full OS.