**Assignment 6**

**T Karthikeyan**

**CED18I064**

**All my codes can be found here** [**CLICK HERE**](https://drive.google.com/drive/folders/1-fiZa5Scg0Cc_3f3qEPAwyuge_MWOqPX?usp=sharing)

**(1) Generate Armstrong number generation within a range.**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**int arr[100000];**

**long long rrr[100000];**

**long long ArmStrCalculation(int n)**

**{**

**int te = n;**

**int ND = 0;**

**while(te!=0) // number of digits in number**

**{**

**ND++;**

**te/=10;**

**}**

**long long sum = 0;**

**while(n!=0) // armstrong number calculation**

**{**

**int temp = n%10;**

**sum = sum + pow(temp,ND);**

**n/=10;**

**}**

**return sum;**

**}**

**struct block**

**{**

**int x;**

**int y;**

**};**

**void \*runner(void \*param)**

**{**

**struct block \*data = param;**

**for(int y = data->x; y <= data->y; y++)**

**{**

**rrr[y] = ArmStrCalculation(arr[y]);**

**}**

**pthread\_exit(0);**

**}**

**int main()**

**{**

**int a,b;**

**printf("Enter the range [a, b] : ");**

**scanf("%d",&a);**

**scanf("%d",&b);**

**for(int k = a; k <= b; k++)**

**arr[k] = k;**

**int start[4];**

**int end[4];**

**int n = b - a + 1;**

**int mid1 = a + (n/4);**

**int mid2 = a + 2\*(n/4);**

**int mid3 = a + 3\*(n/4);**

**start[0] = a; end[0] = mid1 - 1;**

**start[1] = mid1; end[1] = mid2 - 1;**

**start[2] = mid2; end[2] = mid3 - 1;**

**start[3] = mid3; end[3] = b;**

**pthread\_t tid;**

**pthread\_attr\_t attr;**

**for(int z = 0; z < 4; z++)**

**{**

**struct block \*data = (struct block \*)malloc(sizeof(struct block));**

**data->x = start[z];**

**data->y = end[z];**

**pthread\_t tid;**

**pthread\_attr\_t attr;**

**pthread\_attr\_init(&attr);**

**pthread\_create(&tid, &attr, runner, data);**

**pthread\_join(tid, NULL);**

**}**

**printf("List of Armstrong numbers : ");**

**for(int i = a; i <= b; i++)**

**{**

**if(arr[i] == rrr[i])**

**printf("%d ", arr[i]);**

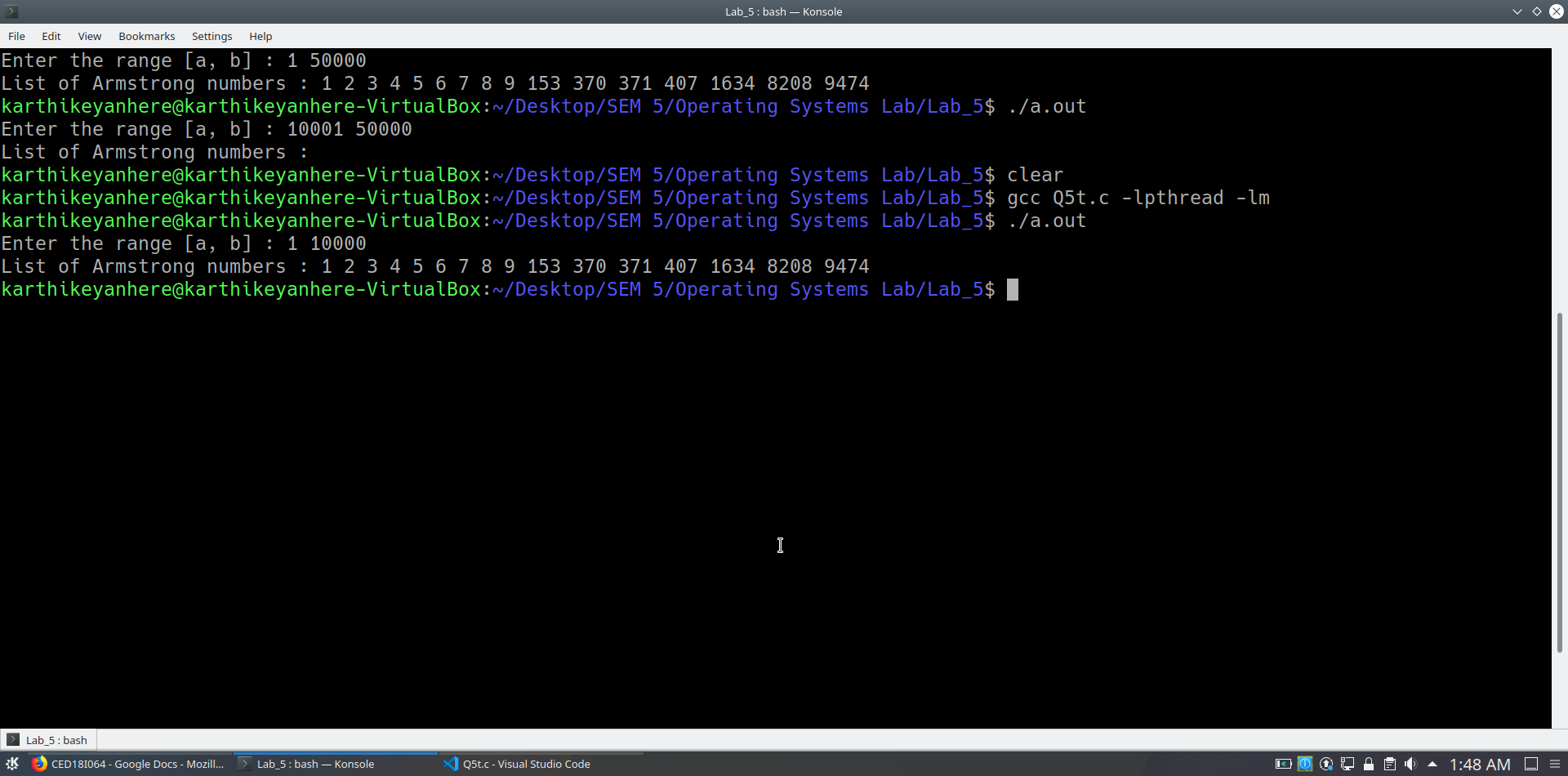
**}**

**printf("\n");**

**return 0;**

**}**

**OUTPUT**

****

**(2) Ascending Order sort and Descending order sort.**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**int n; // size of the array**

**void \*runner1(void \*param);**

**void \*runner2(void \*param);**

**int main()**

**{**

**printf("Enter the number of elements : ");**

**scanf("%d",&n);**

**int \*arr = (int \*)malloc(sizeof(int)\*n);**

**printf("Enter the elements : ");**

**for(int i = 0; i < n; i++)**

**scanf("%d",(arr+i));**

**pthread\_t tid1, tid2;**

**pthread\_attr\_t attr1, attr2;**

**pthread\_attr\_init(&attr1);**

**pthread\_attr\_init(&attr2);**

**pthread\_create(&tid1, &attr1, runner1, arr);**

**pthread\_create(&tid2, &attr2, runner2, arr);**

**pthread\_join(tid1, NULL);**

**pthread\_join(tid2, NULL);**

**return 0;**

**}**

**void \*runner1(void \*param)**

**{**

**int \*arr = param;**

**for(int j = 1; j < n; j++)**

**{**

**int key = arr[j];**

**int i = j-1;**

**while(i>=0 && arr[i]>key)**

**{**

**arr[i+1] = arr[i];**

**i = i-1;**

**}**

**arr[i+1] = key;**

**}**

**printf("Sorted array in ascending order : ");**

**for(int i = 0; i < n; i++)**

**printf("%d ",\*(arr+i));**

**printf("\n");**

**}**

**void \*runner2(void \*param)**

**{**

**int \*arr = param;**

**for(int j = 1; j < n; j++)**

**{**

**int key = arr[j];**

**int i = j-1;**

**while(i>=0 && arr[i]<key)**

**{**

**arr[i+1] = arr[i];**

**i = i-1;**

**}**

**arr[i+1] = key;**

**}**

**printf("Sorted array in descending order : ");**

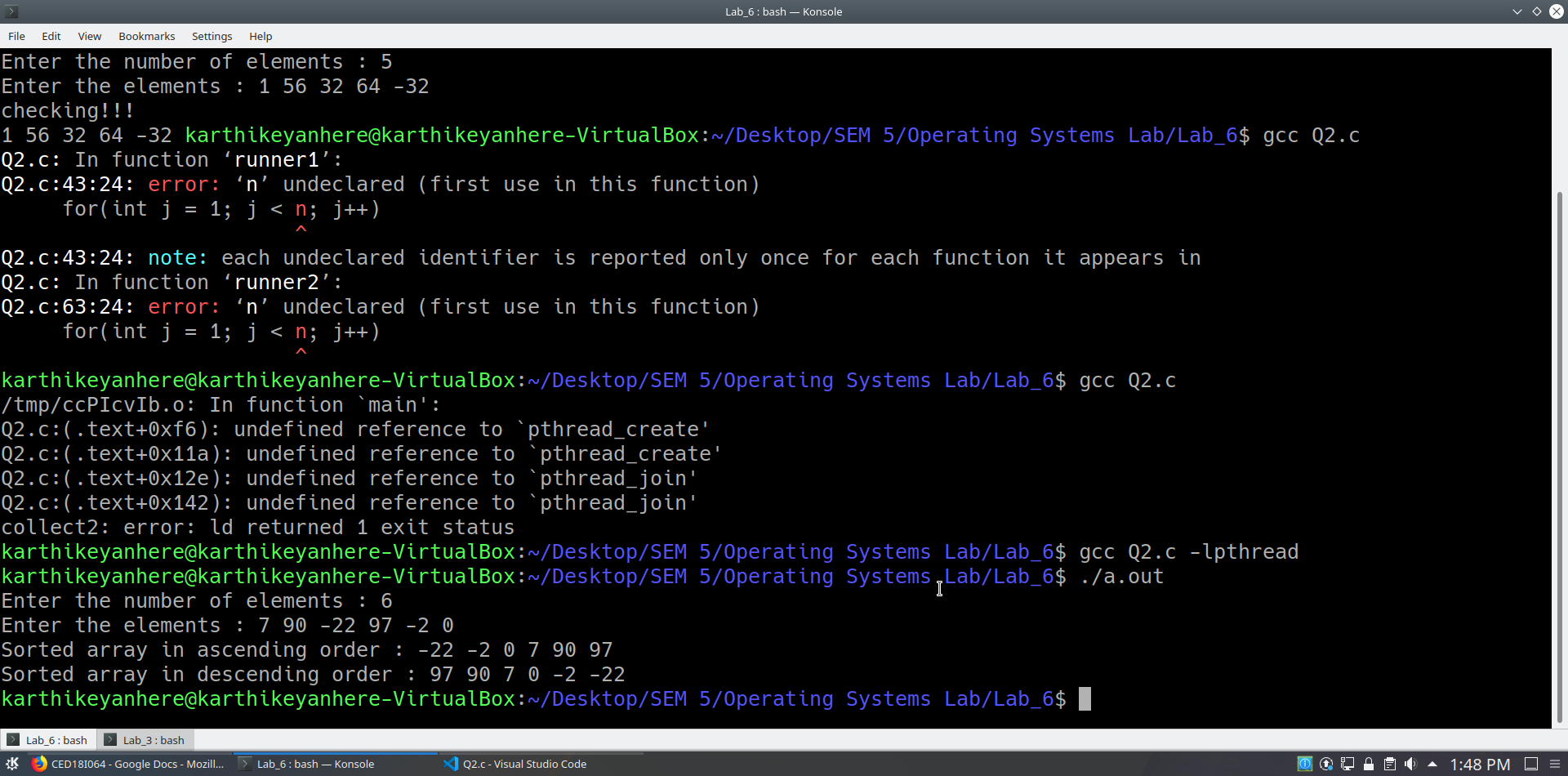
**for(int i = 0; i < n; i++)**

**printf("%d ",\*(arr+i));**

**printf("\n");**

**}**

**OUTPUT**

****

**(3) Implement a multithreaded version of binary search. By default, you can implement a search for the first occurrence and later extend to support multiple occurrence (duplicated elements search as well)**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**int n; // number of elements in an array**

**int p = 0; // number of occurance of particular element**

**int ele; // element to be searched**

**void \*runner1(void \*param);**

**void \*runner2(void \*param);**

**int main()**

**{**

**printf("Enter the number of elements : ");**

**scanf("%d",&n);**

**int \*arr = (int \*)malloc(sizeof(int)\*n);**

**printf("Enter the elements : ");**

**for(int i = 0; i < n; i++)**

**scanf("%d",(arr+i));**

**printf("Enter the element to be searched : ");**

**scanf("%d",&ele);**

**pthread\_t tid1, tid2;**

**pthread\_attr\_t attr1, attr2;**

**pthread\_attr\_init(&attr1);**

**pthread\_attr\_init(&attr2);**

**pthread\_create(&tid1, &attr1, runner1, arr);**

**pthread\_create(&tid2, &attr2, runner2, arr);**

**pthread\_join(tid1, NULL);**

**pthread\_join(tid2, NULL);**

**printf("Element %d is found %d times in an array\n", ele, p);**

**return 0;**

**}**

**void \*runner1(void \*param)**

**{**

**int \*arr = param;**

**for(int i = 0; i < n/2; i++)**

**if(\*(arr+i) == ele)**

**p++;**

**}**

**void \*runner2(void \*param)**

**{**

**int \*arr = param;**

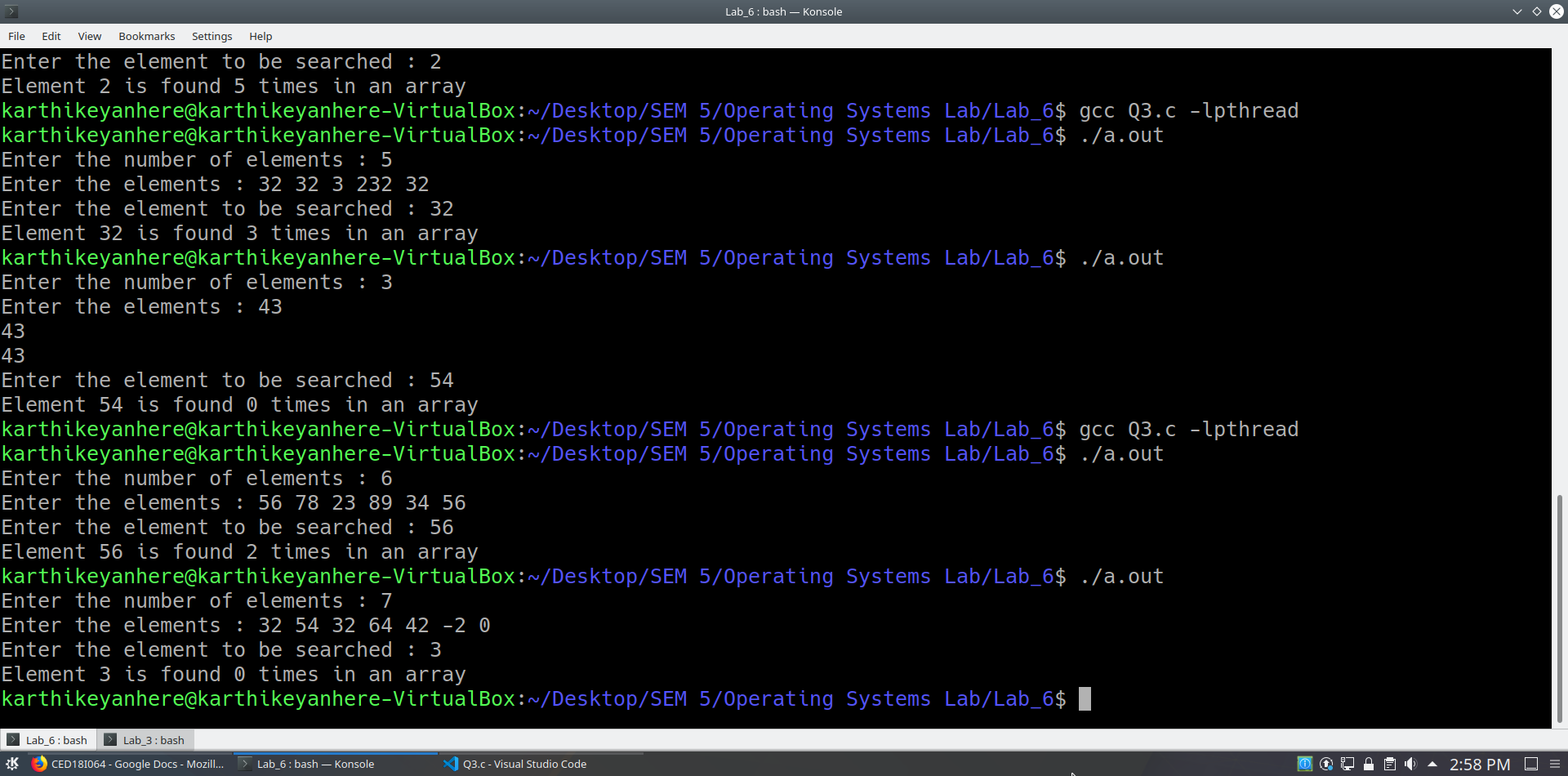
**for(int i = n/2; i < n; i++)**

**if(\*(arr+i) == ele)**

**p++;**

**}**

**OUTPUT**

****

**(4) Generation of Prime Numbers upto a limit supplied as Command Line Parameter.**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**int n; // number upto which prime numbers need to be generated**

**void \*runner(void \*param);**

**\_Bool isprime(int n);**

**int main(int argc, char \*\*argv)**

**{**

**n = atoi(argv[1]);**

**int \*arr = (int \*)malloc(sizeof(int)\*(n+1));**

**for(int i = 0; i <= n; i++)**

**arr[i] = i;**

**pthread\_t tid[n+1];**

**pthread\_attr\_t attr[n+1];**

**for(int i = 0; i <= n; i++)**

**pthread\_attr\_init(&attr[i]);**

**for(int i = 0; i <= n; i++)**

**pthread\_create(&tid[i], &attr[i], runner, &arr[i]);**

**for(int j = 0; j <= n; j++)**

**pthread\_join(tid[j], NULL);**

**printf("\n");**

**return 0;**

**}**

**void \*runner(void \*param)**

**{**

**int \*z = param;**

**if(isprime(\*z))**

**printf("%d ",\*z);**

**pthread\_exit(0);**

**}**

**\_Bool isprime(int n)**

**{**

**int i, flag = 1;**

**// Iterate from 2 to n/2**

**for (i = 2; i <= sqrt(n); i++) {**

**// If n is divisible by any number between**

**// 2 and n/2, it is not prime**

**if (n % i == 0) {**

**flag = 0;**

**break;**

**}**

**}**

**if(n<=1)**

**flag=0;**

**else if(n==2)**

**flag=1;**

**if (flag == 1) {**

**return 1;**

**}**

**else {**

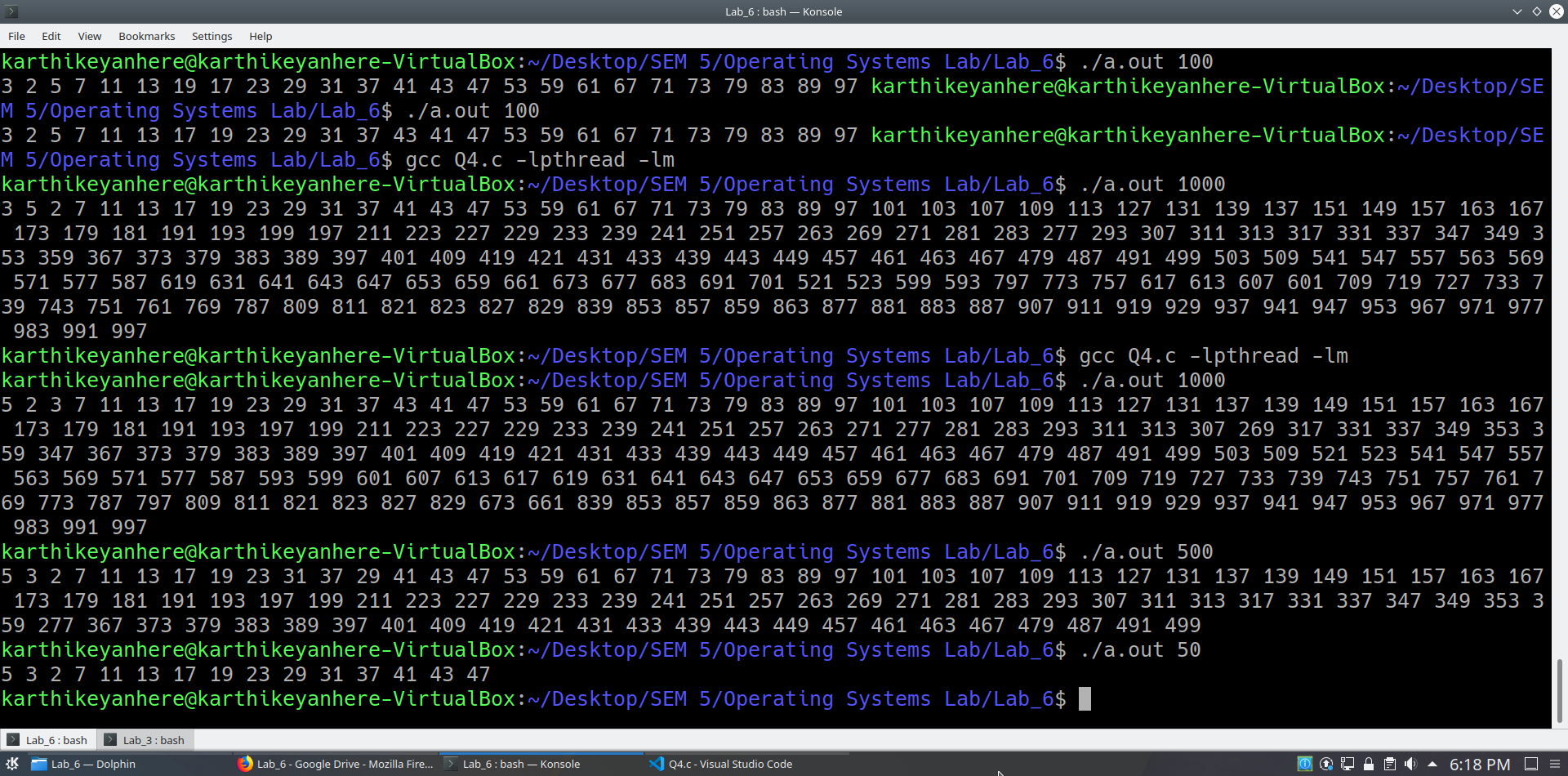
**return 0;**

**}**

**return 0;**

**}**

**OUTPUT**

****

**(5) Computation of Mean, Median, Mode for an array of integers.**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**int n; // number of elements in an array**

**float mean, median;**

**void \*runner1(void \*param);**

**void \*runner2(void \*param);**

**void \*runner3(void \*param);**

**int main()**

**{**

**printf("Enter the number of elements : ");**

**scanf("%d",&n);**

**int \*arr = (int \*)malloc(sizeof(int)\*n);**

**printf("Enter the elements : ");**

**for(int i = 0; i < n; i++)**

**scanf("%d",(arr+i));**

**pthread\_t tid1, tid2, tid3;**

**pthread\_attr\_t attr1, attr2, attr3;**

**pthread\_attr\_init(&attr1);**

**pthread\_attr\_init(&attr2);**

**pthread\_attr\_init(&attr3);**

**pthread\_create(&tid1, &attr1, runner1, arr);**

**pthread\_create(&tid2, &attr2, runner2, arr);**

**pthread\_create(&tid3, &attr3, runner3, arr);**

**pthread\_join(tid1, NULL);**

**pthread\_join(tid2, NULL);**

**pthread\_join(tid3, NULL);**

**printf("Mean is %f\n", mean);**

**printf("Median is %f\n", median);**

**return 0;**

**}**

**void \*runner1(void \*param) // mean**

**{**

**int \*arr = param;**

**long sum = 0;**

**for(int i = 0; i < n; i++)**

**sum += \*(arr+i);**

**mean = (float)sum / n;**

**pthread\_exit(0);**

**}**

**void \*runner2(void \*param) // median**

**{**

**int \*arr = param;**

**for(int j = 1; j < n; j++)**

**{**

**int key = arr[j];**

**int i = j-1;**

**while(i>=0 && arr[i]>key)**

**{**

**arr[i+1] = arr[i];**

**i = i-1;**

**}**

**arr[i+1] = key;**

**}**

**if(n%2) //odd**

**median = arr[n/2];**

**else**

**median = (arr[n/2] + arr[n/2 - 1])/(float)2;**

**pthread\_exit(0);**

**}**

**void \*runner3(void \*param) // mode**

**{**

**int \*arr = param;**

**int maxi = 0; // highest value of brr**

**int nm = 0; // number of modes**

**int brr[n];**

**for(int j = 0; j < n; j++)**

**brr[j] = 1;**

**for(int j = 0; j < n-1; j++)**

**{**

**if(brr[j]!=-1)**

**{**

**for(int k = j+1; k < n; k++)**

**{**

**if(arr[j] == arr[k])**

**{**

**brr[j]++;**

**brr[k] = -1;**

**}**

**}**

**}**

**}**

**for(int k = 0; k < n; k++)**

**{**

**if(brr[k] > maxi)**

**maxi = brr[k];**

**}**

**if(maxi != 1)**

**{**

**for(int i = 0; i < n; i++)**

**if(brr[i] == maxi)**

**nm++;**

**if(nm == 1)**

**{**

**printf("Mode is ");**

**for(int i = 0; i < n; i++)**

**if(brr[i] == maxi)**

**printf("%d", arr[i]);**

**printf("\n");**

**}**

**else**

**{**

**printf("Modes are ");**

**for(int i = 0; i < n; i++)**

**if(brr[i] == maxi)**

**printf("%d ", arr[i]);**

**printf("\n");**

**}**

**}**

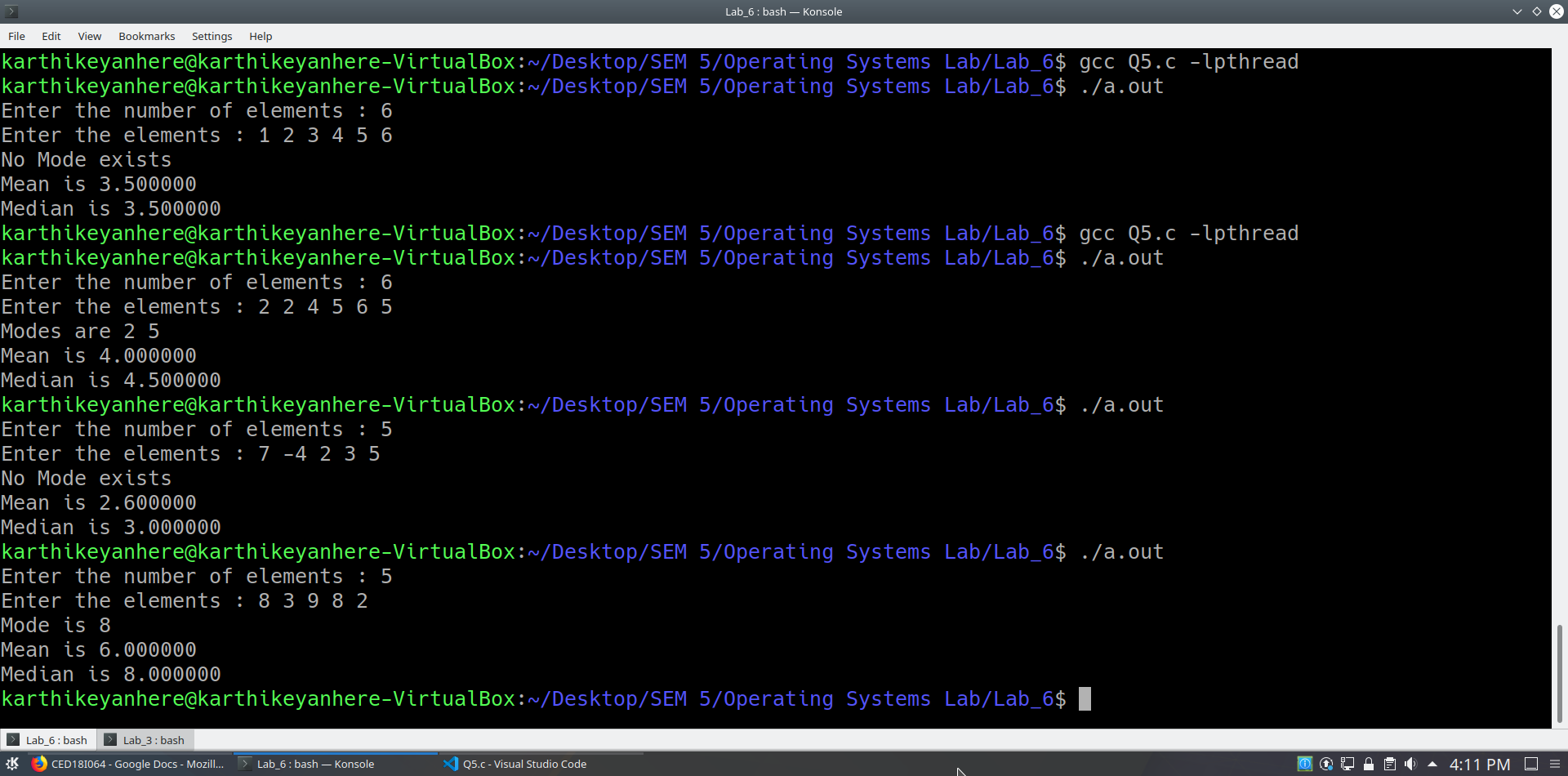
**else**

**printf("No Mode exists\n");**

**pthread\_exit(0);**

**}**

**OUTPUT**

****

**(6) Implement Merge Sort and Quick Sort in a multithreaded fashion.**

**Merge Sort**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**#include <time.h>**

**struct block**

**{**

**int \*arr;**

**int start;**

**int end;**

**};**

**void merge(int \*arr, int l, int m, int r)**

**{**

**int n1 = m - l + 1;**

**int n2 = r - m;**

**// Create temp arrays**

**int L[n1], R[n2];**

**// Copy data to temp arrays L[] and R[]**

**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];**

**// Merge the temp arrays back into arr[l..r]**

**// Initial index of first subarray**

**int i = 0;**

**// Initial index of second subarray**

**int j = 0;**

**// Initial index of merged subarray**

**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++;**

**}**

**// Copy the remaining elements of**

**// L[], if there are any**

**while (i < n1)**

**{**

**arr[k] = L[i];**

**i++;**

**k++;**

**}**

**// Copy the remaining elements of**

**// R[], if there are any**

**while (j < n2)**

**{**

**arr[k] = R[j];**

**j++;**

**k++;**

**}**

**}**

**// l is for left index and r is**

**// right index of the sub-array**

**// of arr to be sorted \*/**

**void \*runner(void \*param)**

**{**

**struct block \*b = param;**

**if ((b->start) < (b->end))**

**{**

**// Same as (l+r)/2, but avoids**

**// overflow for large l and h**

**int m = (b->start) + ((b->end) - (b->start)) / 2;**

**struct block lb;**

**struct block rb;**

**lb.arr = b->arr;**

**lb.start = b->start;**

**lb.end = m;**

**rb.arr = b->arr;**

**rb.start = m+1;**

**rb.end = b->end;**

**pthread\_t tid1, tid2;**

**pthread\_attr\_t attr1, attr2;**

**pthread\_attr\_init(&attr1);**

**pthread\_attr\_init(&attr2);**

**pthread\_create(&tid1, &attr1, runner, &lb);**

**pthread\_create(&tid2, &attr2, runner, &rb);**

**// Sort first and second halves**

**// mergeSort((b->arr), (b->start), m);**

**// mergeSort((b->arr), m + 1, (b->end));**

**pthread\_join(tid1, NULL);**

**pthread\_join(tid2, NULL);**

**merge((b->arr), (b->start), m, (b->end));**

**}**

**pthread\_exit(0);**

**}**

**// UTILITY FUNCTIONS**

**// Function to print an array**

**void printArray(int \*A, int size)**

**{**

**for(int i = 0; i < size; i++)**

**printf("%d ",\*(A+i));**

**}**

**// Driver code**

**int main()**

**{**

**int arr\_size;**

**printf("Enter the size of the array : ");**

**scanf("%d", &arr\_size);**

**int arr[arr\_size];**

**printf("Enter the elements of the array : ");**

**for(int y = 0; y < arr\_size; y++)**

**scanf("%d",&arr[y]);**

**struct block b;**

**b.arr = arr;**

**b.start = 0;**

**b.end = arr\_size - 1;**

**pthread\_t tid;**

**pthread\_attr\_t attr;**

**pthread\_attr\_init(&attr);**

**pthread\_create(&tid, &attr, runner, &b);**

**//mergeSort(arr, 0, arr\_size - 1);**

**pthread\_join(tid, NULL);**

**printf("Sorted array : ");**

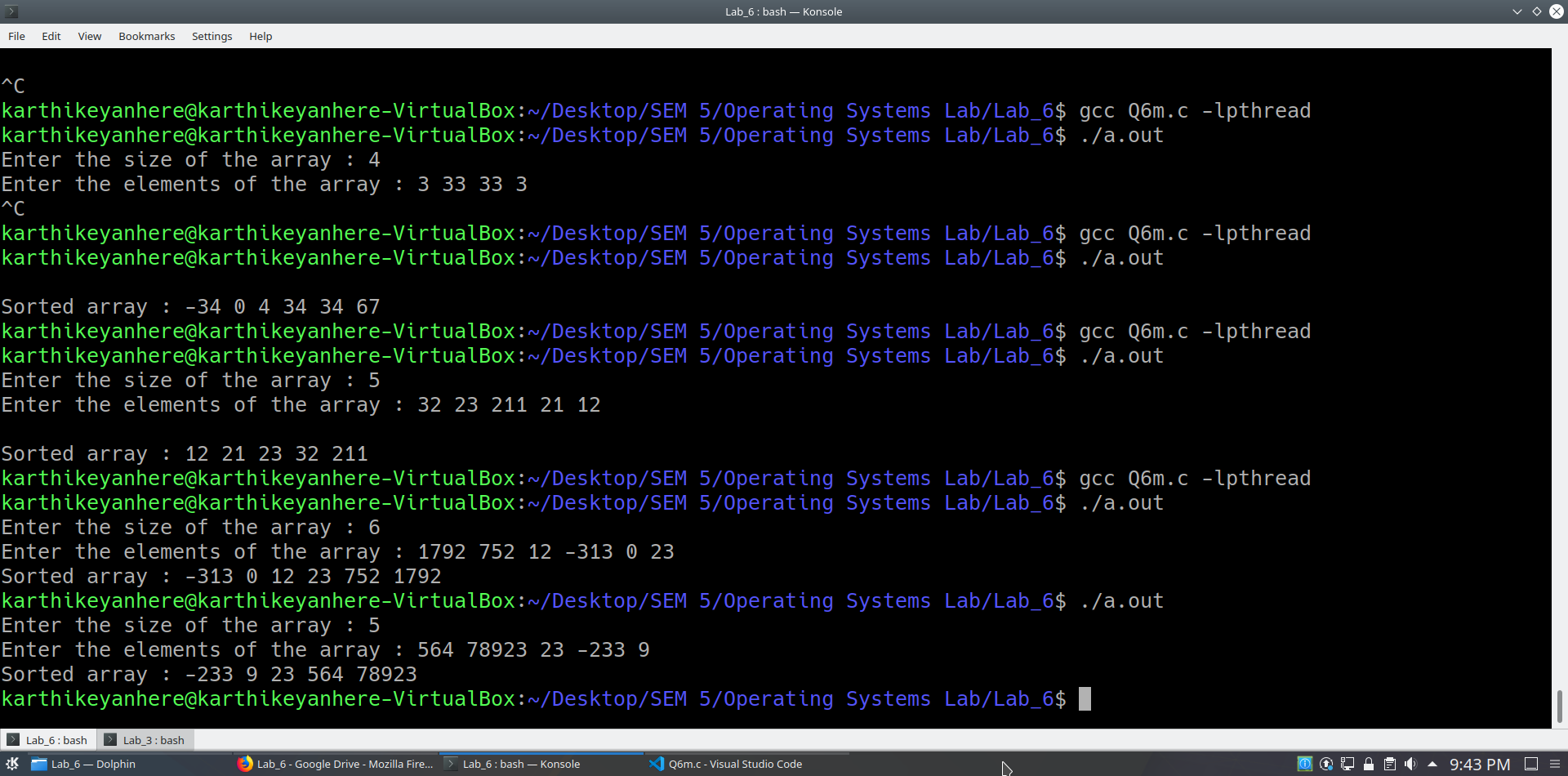
**printArray(arr, arr\_size);**

**printf("\n");**

**return 0;**

**}**

**OUTPUT**

****

**Quick sort**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**#include <time.h>**

**struct block**

**{**

**int \*arr;**

**int start;**

**int end;**

**};**

**// A utility function to swap two elements**

**void swap(int\* a, int\* b)**

**{**

**int t = \*a;**

**\*a = \*b;**

**\*b = t;**

**}**

**/\* This function takes last element as pivot, places**

**the pivot element at its correct position in sorted**

**array, and places all smaller (smaller than pivot)**

**to left of pivot and all greater elements to right**

**of pivot \*/**

**int partition (int arr[], int low, int high)**

**{**

**int pivot = arr[high]; // pivot**

**int i = (low - 1); // Index of smaller element**

**for (int j = low; j <= high- 1; j++)**

**{**

**// If current element is smaller than the pivot**

**if (arr[j] < pivot)**

**{**

**i++; // increment index of smaller element**

**swap(&arr[i], &arr[j]);**

**}**

**}**

**swap(&arr[i + 1], &arr[high]);**

**return (i + 1);**

**}**

**/\* The main function that implements QuickSort**

**arr[] --> Array to be sorted,**

**low --> Starting index,**

**high --> Ending index \*/**

**void \*runner(void \*param)**

**{**

**struct block \*b = param;**

**if ((b->start) < (b->end))**

**{**

**/\* pi is partitioning index, arr[p] is now**

**at right place \*/**

**int pi = partition(b->arr, b->start, b->end);**

**struct block lb;**

**struct block rb;**

**lb.arr = b->arr;**

**lb.start = b->start;**

**lb.end = pi - 1;**

**rb.arr = b->arr;**

**rb.start = pi + 1;**

**rb.end = b->end;**

**pthread\_t tid1, tid2;**

**pthread\_attr\_t attr1, attr2;**

**pthread\_attr\_init(&attr1);**

**pthread\_attr\_init(&attr2);**

**pthread\_create(&tid1, &attr1, runner, &lb);**

**pthread\_create(&tid2, &attr2, runner, &rb);**

**// Separately sort elements before**

**// partition and after partition**

**// quickSort(arr, low, pi - 1);**

**// quickSort(arr, pi + 1, high);**

**pthread\_join(tid1, NULL);**

**pthread\_join(tid2, NULL);**

**}**

**pthread\_exit(0);**

**}**

**/\* Function to print an array \*/**

**void printArray(int arr[], int size)**

**{**

**int i;**

**for (i=0; i < size; i++)**

**printf("%d ", arr[i]);**

**printf("\n");**

**}**

**// Driver program to test above functions**

**int main()**

**{**

**int arr\_size;**

**printf("Enter the size of the array : ");**

**scanf("%d", &arr\_size);**

**int arr[arr\_size];**

**printf("Enter the elements of the array : ");**

**for(int y = 0; y < arr\_size; y++)**

**scanf("%d",&arr[y]);**

**struct block b;**

**b.arr = arr;**

**b.start = 0;**

**b.end = arr\_size - 1;**

**pthread\_t tid;**

**pthread\_attr\_t attr;**

**pthread\_attr\_init(&attr);**

**pthread\_create(&tid, &attr, runner, &b);**

**//quickSort(arr, 0, n-1);**

**pthread\_join(tid, NULL);**

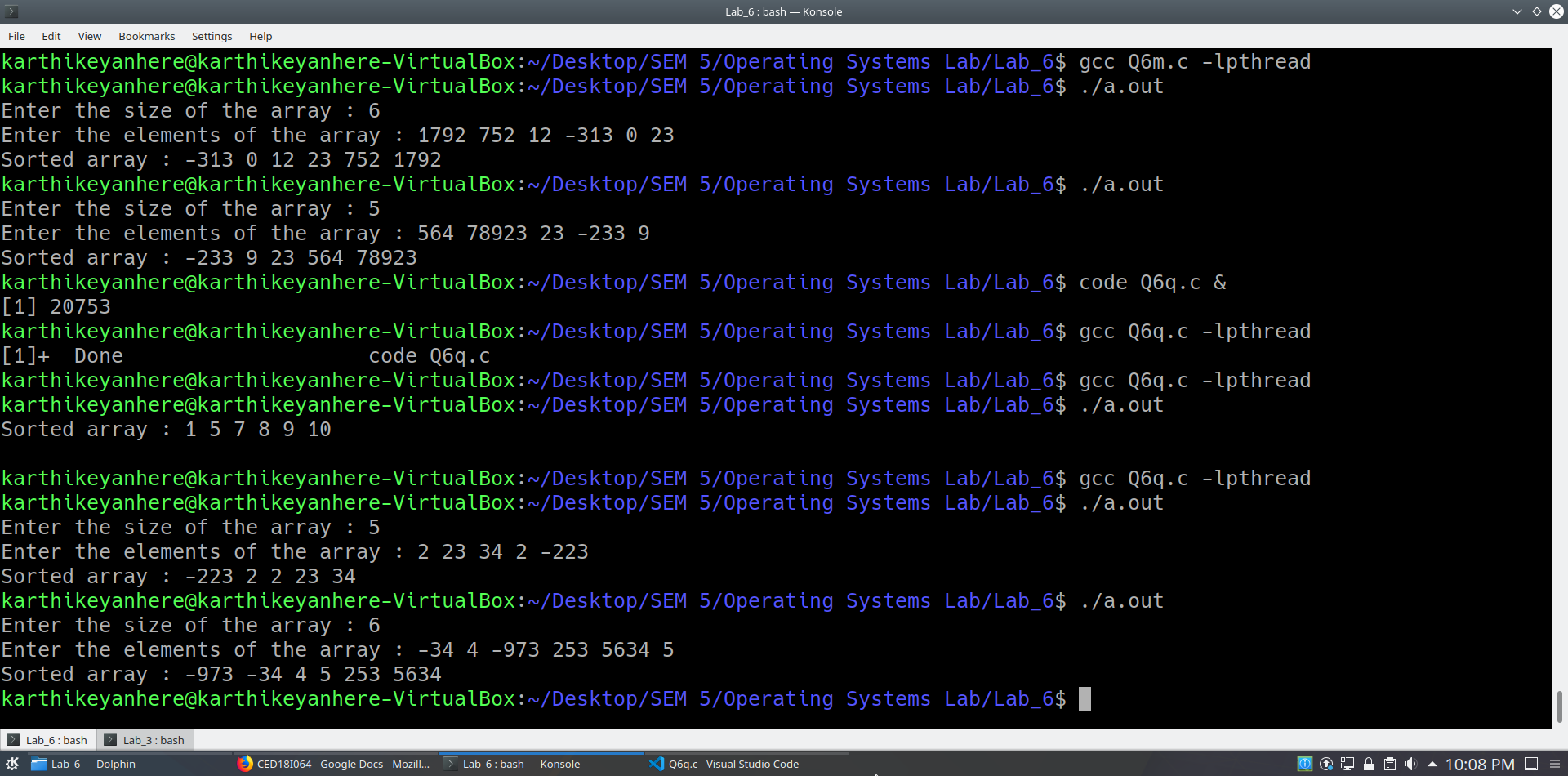
**printf("Sorted array : ");**

**printArray(arr, arr\_size);**

**return 0;**

**}**

**OUTPUT**

****

**(7) Estimation of PI Value using Monte Carlo simulation technique (refer the internet for the method..) using threads.**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**#include <time.h>**

**double piarr[1024];**

**void \*runner(void \*param);**

**\_Bool isInside(float x, float y); // whether point is inside circle of diameter 1**

**int main()**

**{**

**double pi = 0.0;**

**int \*arr = (int \*)malloc(sizeof(int)\*(1024));**

**for(int i = 0; i < 1024; i++)**

**arr[i] = i;**

**// pi is estimated using monte carlo simulation using 1024 threads**

**pthread\_t tid[1024];**

**pthread\_attr\_t attr[1024];**

**for(int i = 0; i < 1024; i++)**

**pthread\_attr\_init(&attr[i]);**

**for(int i = 0; i < 1024; i++)**

**pthread\_create(&tid[i], &attr[i], runner, &arr[i]);**

**for(int j = 0; j < 1024; j++)**

**pthread\_join(tid[j], NULL);**

**for(int i = 0; i < 1024; i++)**

**pi += (piarr[i]\*10000);**

**pi /= (1024\*10000);**

**printf("Estimated pi value using monte carlo simulation is %lf\n", pi);**

**return 0;**

**}**

**void \*runner(void \*param)**

**{**

**int \*i = param;**

**double pi;**

**int inside\_circle = 0;**

**int inside\_square = 0;**

**float x[10000], y[10000];**

**for(int i = 0; i < 10000; i++)**

**{**

**x[i] = ((float)rand()/(float)RAND\_MAX)\*2;**

**y[i] = ((float)rand()/(float)RAND\_MAX)\*2;**

**}**

**for(int k = 0; k < 10000; k++)**

**{**

**if(isInside(x[k], y[k]))**

**inside\_circle++;**

**inside\_square++;**

**}**

**pi = 4\*(double)inside\_circle/inside\_square;**

**piarr[\*i] = pi;**

**}**

**\_Bool isInside(float x, float y)**

**{**

**if((((x-1.0)\*(x-1.0))+((y-1.0)\*(y-1.0))) < 1.0)**

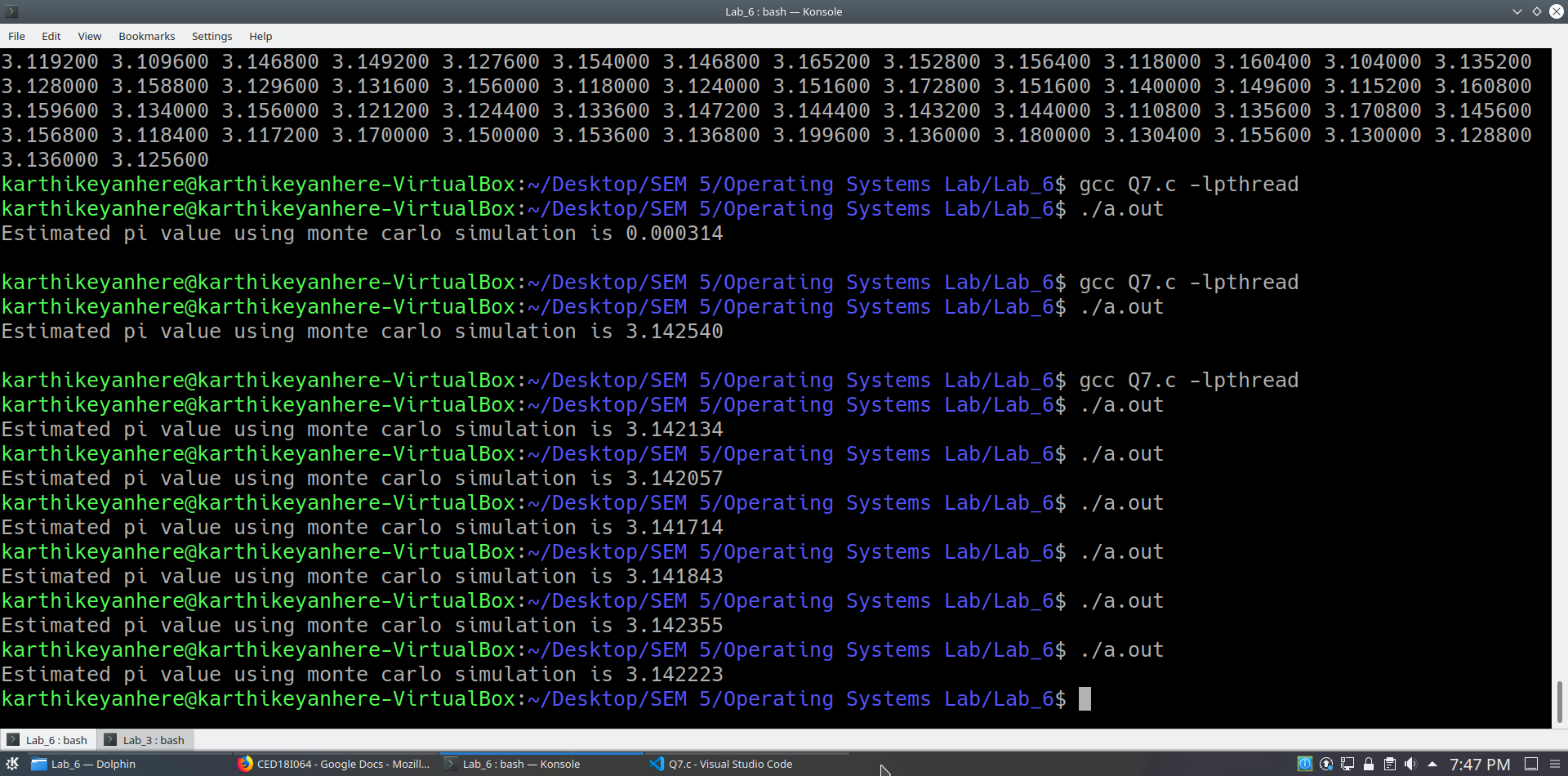
**return 1;**

**else**

**return 0;**

**}**

**OUTPUT**

****

**Optional:**

**(8) Computation of a Matrix Inverse using Determinant, Cofactor threads, etc.**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**#include <time.h>**

**#define N 3**

**struct block\_adj**

**{**

**int \*\*arr;**

**int \*\*adrr;**

**int i;**

**int j;**

**};**

**struct block\_inv**

**{**

**int \*\*arr;**

**float \*\*inver;**

**int i;**

**int j;**

**int det;**

**};**

**struct block\_transpose**

**{**

**int \*\*arr;**

**int \*\*trr;**

**int resp;**

**};**

**void \*runner\_adj(void \*param)**

**{**

**struct block\_adj \*b = param;**

**int \*\*temp = malloc(sizeof(int)\*(N-1));**

**for(int i = 0; i < (N-1); i++)**

**{**

**temp[i] = malloc(sizeof(int)\*(N-1));**

**}**

**int i = 0, j = 0;**

**for (int row = 0; row < N; row++)**

**{**

**for (int col = 0; col < N; col++)**

**{**

**if (row != (b->i) && col != (b->j))**

**{**

**temp[i][j++] = b->arr[row][col];**

**if (j == (N - 1))**

**{**

**j = 0;**

**i++;**

**}**

**}**

**}**

**}**

**int sign = 1;**

**if(((b->i)+(b->j))%2)**

**sign = -1;**

**b->adrr[b->i][b->j] = sign\*((temp[0][0]\*temp[1][1])-(temp[0][1]\*temp[1][0]));**

**}**

**void \*runner\_transpose(void \*param)**

**{**

**struct block\_transpose \*bt = param;**

**for(int i = 0; i < N; i++)**

**{**

**bt->trr[i][bt->resp] = bt->arr[bt->resp][i];**

**}**

**}**

**void \*runner\_inv(void \*param)**

**{**

**struct block\_inv \*b = param;**

**b->inver[b->i][b->j] = (float)b->arr[b->i][b->j] / b->det;**

**}**

**void printarray(int \*\*arr, int size)**

**{**

**for(int i = 0; i < size; i++)**

**{**

**for(int j = 0; j < size; j++)**

**printf("%d ", arr[i][j]);**

**printf("\n");**

**}**

**}**

**int main()**

**{**

**printf("Enter an 3x3 matrix row-wise : ");**

**int \*\*arr = malloc(sizeof(int)\*N);**

**for(int i = 0; i < N; i++)**

**{**

**arr[i] = malloc(sizeof(int)\*N);**

**for(int j = 0; j < N; j++)**

**scanf("%d", &arr[i][j]);**

**}**

**printf("Given array is \n");**

**printarray(arr, N);**

**int \*\*adrr = malloc(sizeof(int \*)\*N); // cofactor matrix untransposed**

**for(int i = 0; i < N; i++)**

**adrr[i] = malloc(sizeof(int)\*N);**

**int \*\*atrr = malloc(sizeof(int \*)\*N); // cofactor matrix transposed**

**for(int i = 0; i < N; i++)**

**atrr[i] = malloc(sizeof(int)\*N);**

**float \*\*inver = malloc(sizeof(float \*)\*N); // inverse**

**for(int i = 0; i < N; i++)**

**inver[i] = malloc(sizeof(float)\*N);**

**// cofactor matrices creation**

**struct block\_adj b[3][3];**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**{**

**b[i][j].arr = arr;**

**b[i][j].adrr = adrr;**

**b[i][j].i = i;**

**b[i][j].j = j;**

**}**

**pthread\_t tid[3][3];**

**pthread\_attr\_t attr[3][3];**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**pthread\_attr\_init(&attr[i][j]);**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**{**

**pthread\_create(&tid[i][j], &attr[i][j], runner\_adj, &b[i][j]);**

**}**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**pthread\_join(tid[i][j], NULL);**

**printf("Cofactor matrix untransposed\n");**

**for(int i = 0; i < N; i++)**

**{**

**for(int j = 0; j < N; j++)**

**printf("%d ", adrr[i][j]);**

**printf("\n");**

**}**

**// Transpose calculation**

**struct block\_transpose bt[N];**

**for(int i = 0; i < N; i++)**

**{**

**bt[i].arr = adrr;**

**bt[i].trr = atrr;**

**bt[i].resp = i;**

**}**

**pthread\_t ttid[N];**

**pthread\_attr\_t atttr[N];**

**for(int i = 0; i < N; i++)**

**pthread\_attr\_init(&atttr[i]);**

**for(int i = 0; i < N; i++)**

**{**

**pthread\_create(&ttid[i], &atttr[i], runner\_transpose, &bt[i]);**

**}**

**for(int i = 0; i < N; i++)**

**{**

**pthread\_join(ttid[i], NULL);**

**}**

**printf("Cofactor matrix transposed\n");**

**for(int i = 0; i < N; i++)**

**{**

**for(int j = 0; j < N; j++)**

**printf("%d ", atrr[i][j]);**

**printf("\n");**

**}**

**int determinant = (arr[0][0]\*adrr[0][0]) + (arr[0][1]\*adrr[0][1]) + (arr[0][2]\*adrr[0][2]);**

**printf("determinant is %d\n", determinant);**

**// dividing each element by determinant to get inverse**

**struct block\_inv bi[3][3];**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**{**

**bi[i][j].arr = atrr;**

**bi[i][j].i = i;**

**bi[i][j].j = j;**

**bi[i][j].det = determinant;**

**bi[i][j].inver = inver;**

**}**

**pthread\_t tttid[3][3];**

**pthread\_attr\_t attttr[3][3];**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**pthread\_attr\_init(&attttr[i][j]);**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**{**

**pthread\_create(&tttid[i][j], &attttr[i][j], runner\_inv, &bi[i][j]);**

**}**

**for(int i = 0; i < 3; i++)**

**for(int j = 0; j < 3; j++)**

**pthread\_join(tttid[i][j], NULL);**

**printf("Inverse\n");**

**for(int i = 0; i < N; i++)**

**{**

**for(int j = 0; j < N; j++)**

**printf("%.2f ", inver[i][j]);**

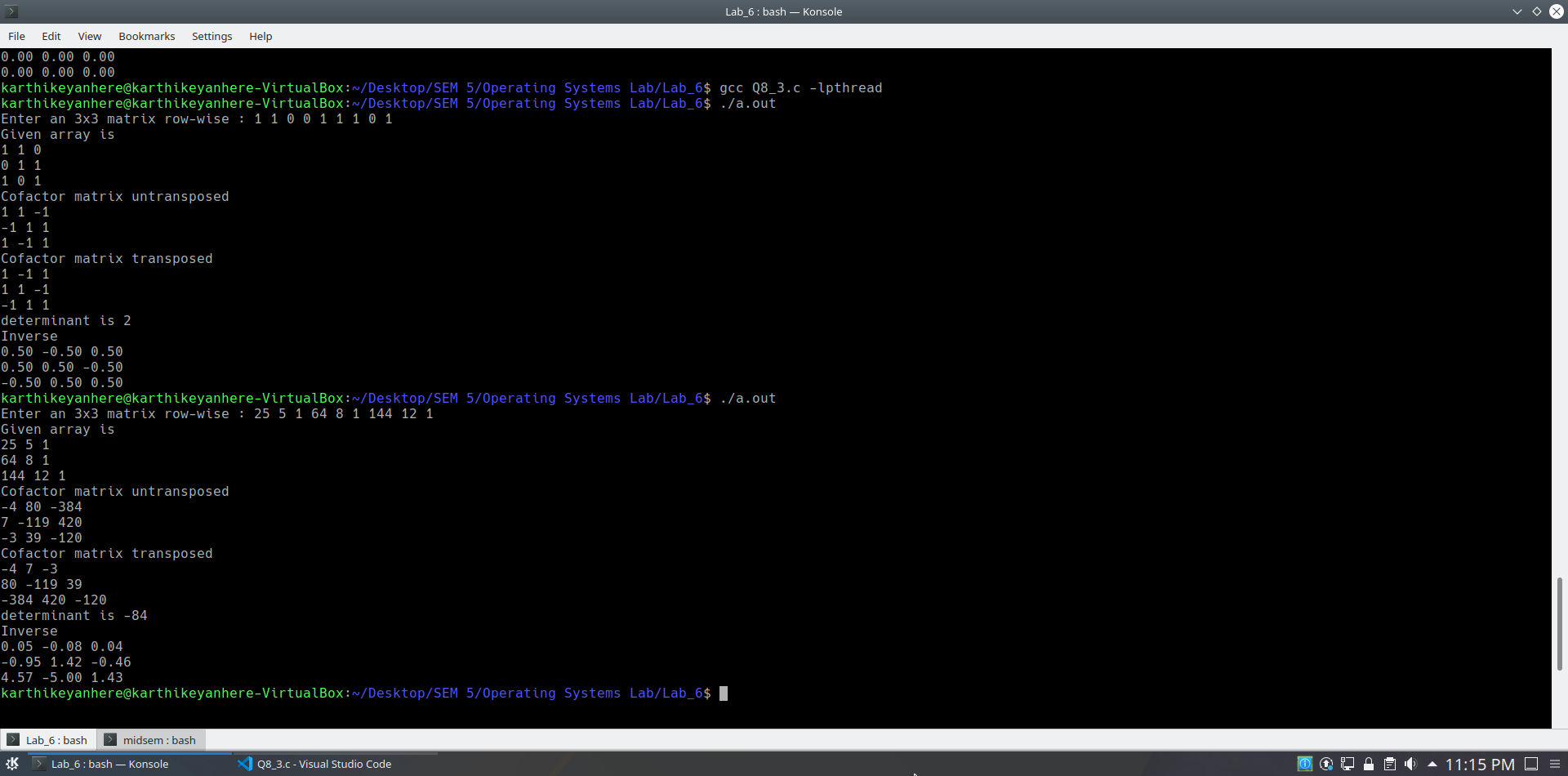
**printf("\n");**

**}**

**return 0;**

**}**

**OUTPUT**

****

**(9) Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a multithreaded fashion to contribute a faster version of fib series generation.**

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**#include <time.h>**

**long long fib[30];**

**void \*runner(void \*param);**

**long long fibonacci(int n);**

**int main()**

**{**

**int n;**

**printf("Enter the number of fibonacci numbers to be generated : ");**

**scanf("%d", &n);**

**pthread\_t tid;**

**pthread\_attr\_t attr;**

**pthread\_attr\_init(&attr);**

**pthread\_create(&tid, &attr, runner, &n);**

**pthread\_join(tid, NULL);**

**printf("Fibonacci numbers : ");**

**for(int i = 0; i < n; i++)**

**printf("%lld ", fib[i]);**

**printf("\n");**

**return 0;**

**}**

**void \*runner(void \*param)**

**{**

**int \*i = param;**

**if(\*i < 2)**

**fib[\*i] = \*i;**

**else**

**{**

**int x = \*i - 1;**

**int y = \*i - 2;**

**pthread\_t tid1, tid2;**

**pthread\_attr\_t attr1, attr2;**

**pthread\_attr\_init(&attr1);**

**pthread\_attr\_init(&attr2);**

**pthread\_create(&tid1, &attr1, runner, &x);**

**pthread\_create(&tid2, &attr2, runner, &y);**

**pthread\_join(tid1, NULL);**

**pthread\_join(tid2, NULL);**

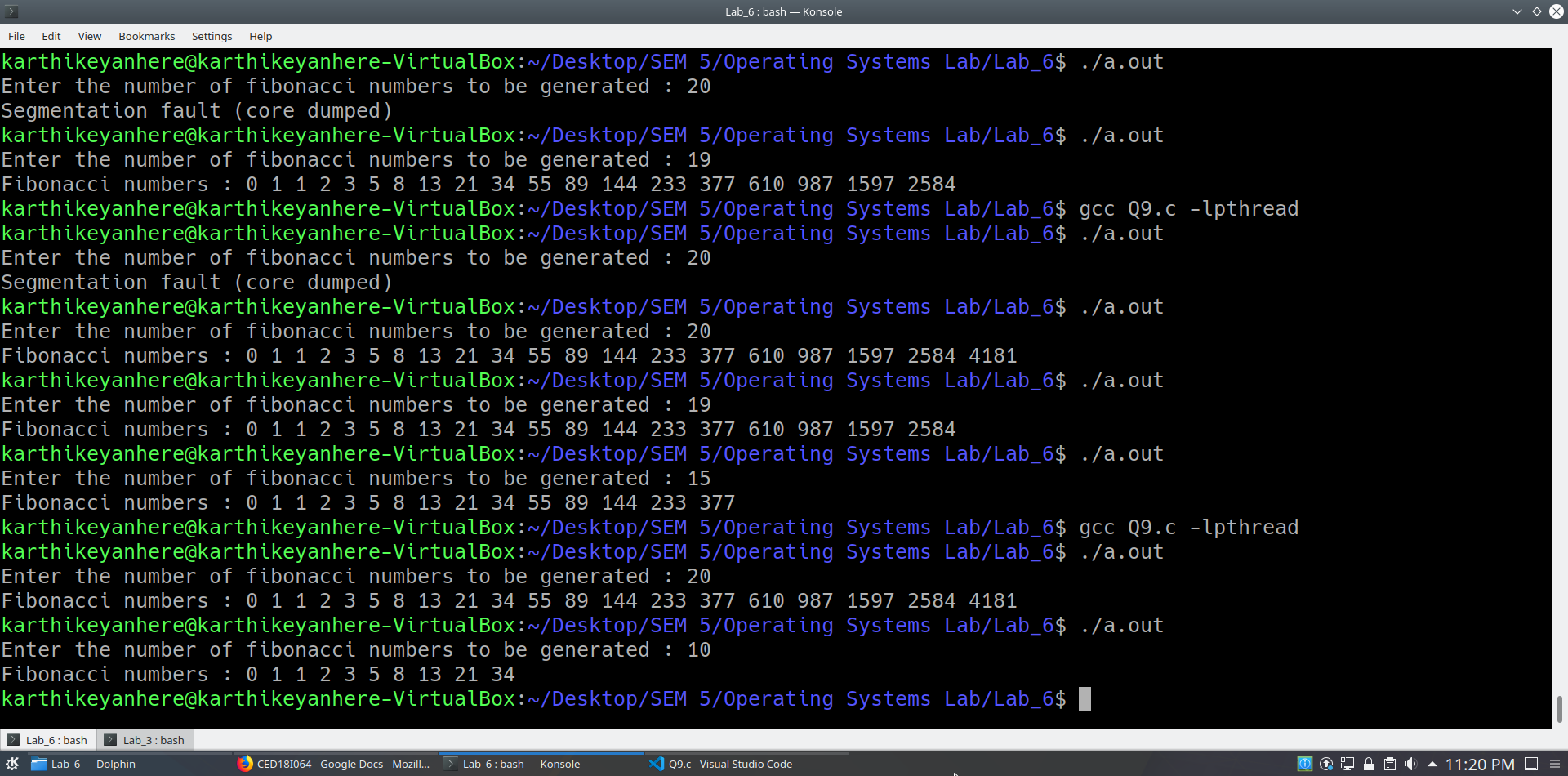
**fib[\*i] = fib[x] + fib[y];**

**}**

**pthread\_exit(0);**

**}**

**OUTPUT**

****

**(10) Longest common subsequence generation problem using threads.**

**LOGIC**

*Dynamic programming table is filled in a diagonal fashion, thus parallelism*

**CODE**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**#include <math.h>**

**#include <time.h>**

**int L[1000][1000];**

**char Y[] = "AGGTAB";**

**char X[] = "GXTXAYB";**

**int min(int a, int b);**

**int max(int a, int b);**

**struct block**

**{**

**int x;**

**int y;**

**};**

**/\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/**

**//int lcs( char \*X, char \*Y, int m, int n )**

**void \*runner(void \*param)**

**{**

**/\* Following steps build L[m+1][n+1] in bottom up fashion. Note**

**that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] \*/**

**struct block \*b = param;**

**int i = b->x;**

**int j = b->y;**

**/\*for (int i=0; i <= m; i++)**

**{**

**for (int j=0; j <= n; j++)**

**{**

**if (i == 0 || j == 0)**

**L[i][j] = 0; \*/**

**if (X[i-1] == Y[j-1])**

**{**

**L[i][j] = L[i-1][j-1] + 1;**

**}**

**else**

**L[i][j] = max(L[i-1][j], L[i][j-1]);**

**/\* }**

**}**

**//L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1]**

**return L[m][n]; \*/**

**pthread\_exit(0);**

**}**

**/\* Utility function to get max of 2 integers \*/**

**int min(int a, int b)**

**{**

**return (a > b)? b : a;**

**}**

**int max(int a, int b)**

**{**

**return (a > b)? a : b;**

**}**

**/\* Driver program to test above function \*/**

**int main()**

**{**

**printf("Enter main string(larger) : ");**

**scanf("%s",X);**

**printf("Enter sub string(smaller) : ");**

**scanf("%s",Y);**

**int m = strlen(X);**

**int n = strlen(Y);**

**int mi = min(m,n);**

**int ma = max(m,n);**

**for(int i = 0; i <= ma; i++)**

**{**

**L[0][i] = 0;**

**L[i][0] = 0;**

**}**

**for(int k = 1; k <= mi; k++) // L[1,k]...**

**{**

**int tx = 1;**

**int ty = k;**

**int strand\_size = 0;**

**while(ty!=0 && tx!=(ma+1))**

**{**

**strand\_size++;**

**tx++;**

**ty--;**

**}**

**int strand\_x[strand\_size];**

**int strand\_y[strand\_size];**

**tx = 1; ty = k;**

**int temp = 0;**

**while(ty!=0 && tx!=(ma+1))**

**{**

**strand\_x[temp] = tx;**

**strand\_y[temp] = ty;**

**temp++;**

**tx++;**

**ty--;**

**}**

**// for(int i = 0; i < strand\_size; i++)**

**// printf("(%d,%d),", strand\_x[i], strand\_y[i]);**

**// printf("\n");**

**int NUM\_THREADS = strand\_size;**

**struct block b[NUM\_THREADS];**

**for(int i = 0; i < NUM\_THREADS; i++)**

**{**

**b[i].x = strand\_x[i];**

**b[i].y = strand\_y[i];**

**}**

**pthread\_t tid[NUM\_THREADS];**

**pthread\_attr\_t attr[NUM\_THREADS];**

**for(int i = 0; i < NUM\_THREADS; i++)**

**pthread\_attr\_init(&attr[i]);**

**for(int i = 0; i < NUM\_THREADS; i++)**

**{**

**pthread\_create(&tid[i], &attr[i], runner, &b[i]);**

**}**

**for(int i = 0; i < NUM\_THREADS; i++)**

**pthread\_join(tid[i], NULL);**

**}**

**for(int k = 2; k <= ma; k++) // L[k,mi]...**

**{**

**int tx = k;**

**int ty = mi;**

**int strand\_size = 0;**

**while(ty!=0 && tx!=(ma+1))**

**{**

**strand\_size++;**

**tx++;**

**ty--;**

**}**

**int strand\_x[strand\_size];**

**int strand\_y[strand\_size];**

**tx = k; ty = mi;**

**int temp = 0;**

**while(ty!=0 && tx!=(ma+1))**

**{**

**strand\_x[temp] = tx;**

**strand\_y[temp] = ty;**

**temp++;**

**tx++;**

**ty--;**

**}**

**int NUM\_THREADS = strand\_size;**

**struct block b[NUM\_THREADS];**

**for(int i = 0; i < NUM\_THREADS; i++)**

**{**

**b[i].x = strand\_x[i];**

**b[i].y = strand\_y[i];**

**}**

**pthread\_t tid[NUM\_THREADS];**

**pthread\_attr\_t attr[NUM\_THREADS];**

**for(int i = 0; i < NUM\_THREADS; i++)**

**pthread\_attr\_init(&attr[i]);**

**for(int i = 0; i < NUM\_THREADS; i++)**

**{**

**pthread\_create(&tid[i], &attr[i], runner, &b[i]);**

**}**

**for(int i = 0; i < NUM\_THREADS; i++)**

**pthread\_join(tid[i], NULL);**

**}**

**//printf("Length of LCS is %d", lcs( X, Y, m, n ) );**

**printf("X:%s\n",X);**

**printf("Y:%s\n",Y);**

**printf("LCS is %d\n",L[m][n]);**

**for(int i = 0; i <= ma; i++)**

**{**

**for(int j = 0; j <= mi; j++)**

**{**

**printf("%d ",L[i][j]);**

**}**

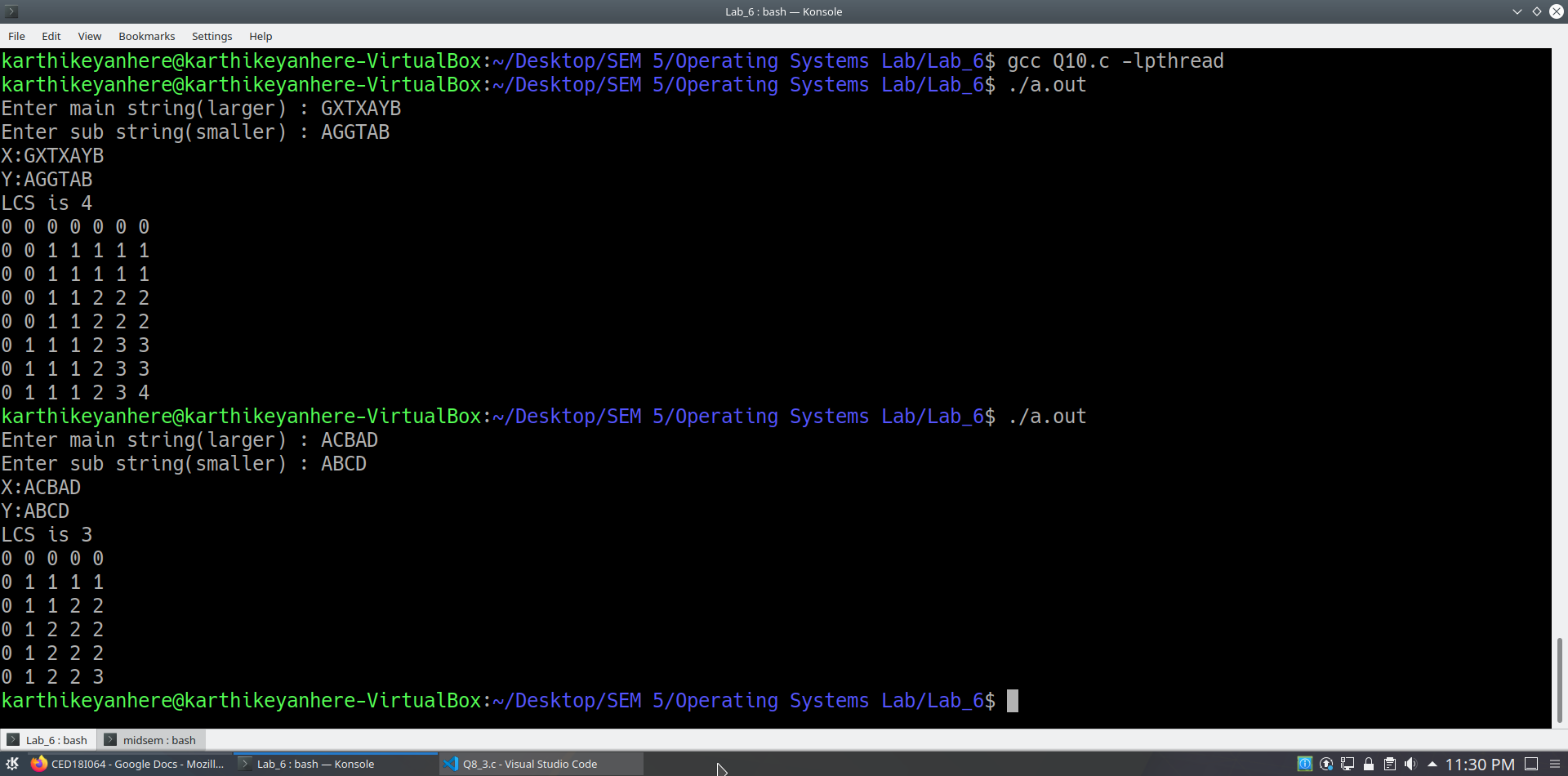
**printf("\n");**

**}**

**return 0;**

**}**

**OUTPUT**

****