# Algorithm Report1

2015313254 노인호

October 2th 2019

## 1 Environmnet

코드는 다음과 같은 환경에서 실행되었다.

- Ubuntu 16.04 LTS 64bit
- gcc 5.4.0

만약 코드가 깨져서 잘 안보인다면 다음 url를 가진 사이트를 참고하면 된다.

- https://gist.github.com/nosy0411/af1e1914fea8acc4358a106ff7f871c5

## 2 Algo-1

### 2.1 problem

Write the INSERTION-SORT function to sort into descending order. This is pseudocode.

**Algorithm 1** INSERTION-SORT(A)

```
for j=2 to A.lenghth do
  key=A[j]
  i=j-1
  while i>0 && A[i]<key do
    A[i+1]=A[i]
    i=i-1
  end while
  A[i+1]=key
end for
```

### 2.2 source code

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

#define randomize() srand(time(NULL))
#define random(n) (rand()%(n))
#define SIZE 100
#define SAMPLE 10000
int count=0;

void InsertionSort(int* array, int n) {
        int i, j, key;
        for (i = 1; i < n; i++) {
                key = array[i];  // Choose the i-th element.
                for (j = i - 1; j >= 0; j--) {
                // If the j-th element is lower than key,
                // move to the next position.
                        if (key > array[j]){
                                array[j + 1] = array[j];
                                count++;
```

```
                        }
                        else{
                                break;
                        }
                }
                // Otherwise, move the key to the (j+1)-th element.
                array[j + 1] = key;
        }
}

void sorted_fill(int* array){
        for(int i=0;i<SIZE;i++){
                array[i]=SIZE-1-i;
        }
}

int main(){

        int A1[SIZE];
        int A2[SIZE];
        int A3[SIZE];

        randomize();
        for(int i=0;i<SIZE;i++){
                while(1){
                        int check=0;
                        A1[i]=random(SAMPLE);
                        for(int j=0;j<i;j++){
                                if(A1[j]==A1[i]){
                                        check=1;
                                        break;
                                }
                        }
                        if(check==1){
                                continue;
                        }
                        else{
                                break;
                        }
                }
        }
        sorted_fill(A2);

        for(int i=0;i<SIZE;i++){
                A3[i]=A2[SIZE-1-i];
        }

        printf("Before sort A1: ");
        for(int i=0;i<SIZE;i++){
                printf("%d ",A1[i]);
        }
        printf("\\n\\n");
        printf("After sort A1: ");
        InsertionSort(A1,SIZE);
        int A1count=count;
        count=0;

        for(int i=0;i<SIZE;i++){
                printf("%d ",A1[i]);
        }
        printf("\\n\\n");
        printf("Before sort A2: ");
        for(int i=0;i<SIZE;i++){
                printf("%d ",A2[i]);
        }
        printf("\\n\\n");
        printf("After sort A2: ");
        InsertionSort(A2,SIZE);
        int A2count=count;
        count=0;
```

```
        for (int i=0;i<SIZE;i++){
                printf("%d ",A2[i]);
        }
        printf("\\n\\n");
        printf("Before sort A3: ");
        for(int i=0;i<SIZE;i++){
                printf("%d ",A3[i]);
        }
        printf("\\n\\n");
        printf("After sort A3: ");
        InsertionSort(A3, SIZE);
        int A3count=count;
        count=0;
        for(int i=0;i<SIZE;i++){
                printf("%d ", A3[i]);
        }
        printf("\\n\\n\\n");

        printf("A1 comparison count : %d\n",A1count);
        printf("A2 comparison count : %d\n",A2count);
        printf("A3 comparison count : %d\n",A3count);
}
```

Listing 1: algo-1.c

### 2.3 result

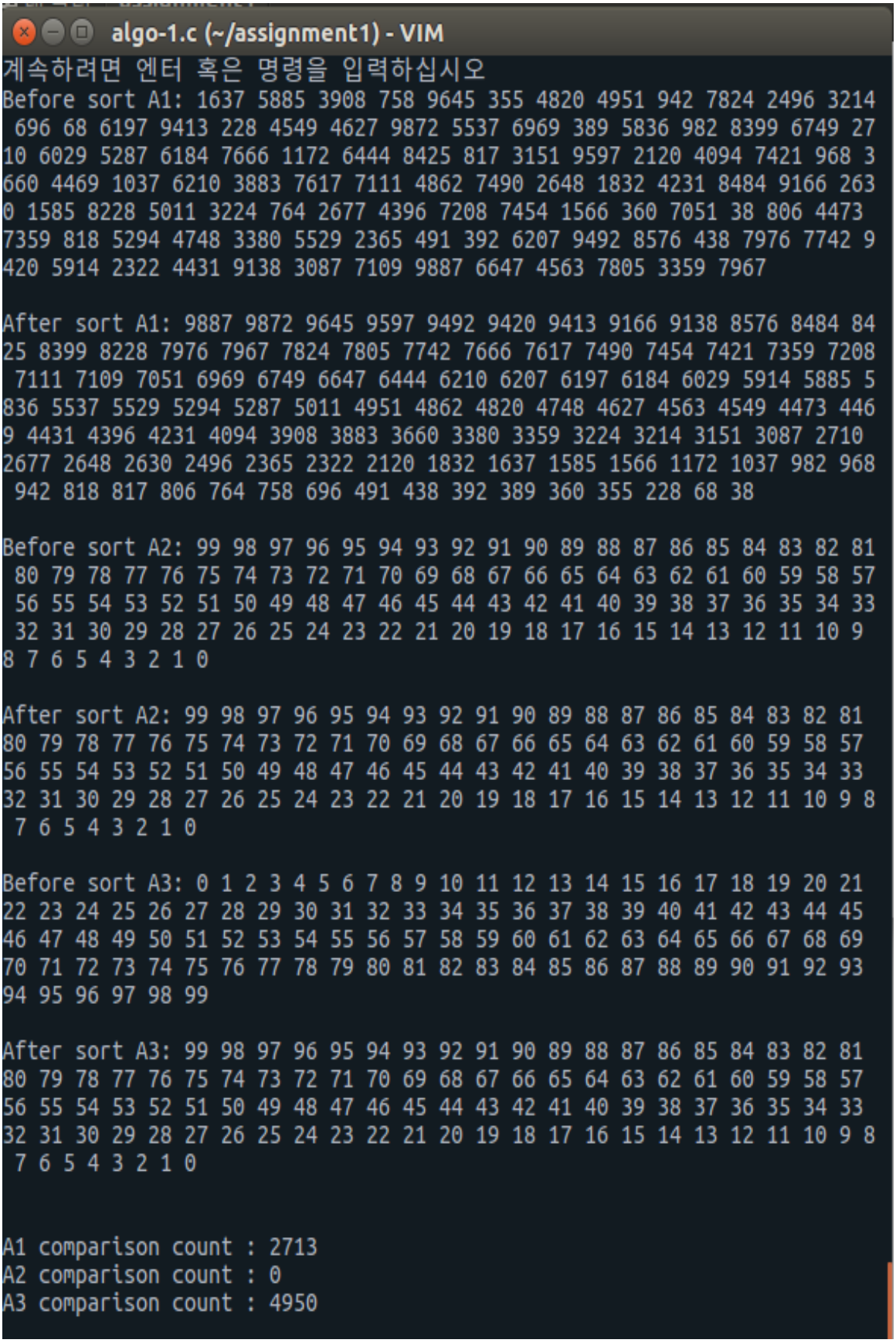



Figure 1: Executing result of algo-1.c

# 3 Algo-2

## 3.1 problem

Write the MERGE-SORT function to sort into descending order.

## 3.2 source code

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

#define randomize() srand(time(NULL))
#define random(n) (rand()%(n))
#define SIZE 100
#define SAMPLE 1000

int count=0;

void Merge(int* list, int left, int mid, int right) {
        int sorted[SIZE];
        int first = left, second = mid + 1, i = left;
        // Merge two lists by comparing elements in sequence.
        while (first <= mid \&\& second <= right) {
                if (list[first] >= list[second]){
                        sorted[i++] = list[first++];
                }
                else{
                        sorted[i++] = list[second++];
                        count++;
                }
        }
        // For remaining items, add them in sequence.
        if (first > mid) {
        for (int j = second; j <= right; j++)
                sorted[i++] = list[j];
        } else {
        for (int j = first; j <= mid; j++)
                sorted[i++] = list[j];
        }

        // Copy the sorted list to the list.
        for (int j = left; j <= right; j++)
                list[j] = sorted[j];
}

void MergeSort(int* list, int left, int right) {
        if (left < right) {
                int mid = (left + right) / 2;      // Equal partitioning
                MergeSort(list, left, mid);        // Sorting sublists
                MergeSort(list, mid + 1, right);   // Sorting sublists
                Merge(list, left, mid, right);     // Merging two sublists
                //for (int i = 0; i < SIZE; i++)
                //        printf("%d ", list[i]);
                //printf("\n");
        }
}

void sorted_fill(int* list){
        for(int i=0;i<SIZE;i++){
                list[i]=SIZE-i;
        }
}

int main(){
        int A1[SIZE];
        int A2[SIZE];
        int A3[SIZE];
```

```
        randomize();
        for(int i=0;i<SIZE;i++){
                while(1){
                        int check=0;
                        A1[i]=random(SAMPLE);
                        for(int j=0;j<i;j++){
                                if(A1[j]==A1[i]){
                                        check=1;
                                        break;
                                }
                        }
                        if(check==1){
                                continue;
                        }
                        else{
                                break;
                        }
                }
        }
        sorted_fill(A2);
        for(int i=0;i<SIZE;i++){
                A3[i]=A2[SIZE-1-i];
        }
        printf("Before sort A1: ");
        for(int i=0;i<SIZE;i++){
                printf("%d ",A1[i]);
        }
        printf("\\n\\n");
        printf("After sort A1: ");
        MergeSort(A1,0,SIZE-1);
        int A1count=count;
        count=0;
        for(int i=0;i<SIZE;i++){
                printf("%d ",A1[i]);
        }
        printf("\\n\\n");
        printf("Before sort A2: ");
        for(int i=0;i<SIZE;i++){
                printf("%d ",A2[i]);
        }
        printf("\\n\\n");
        printf("After sort A2: ");
        MergeSort(A2,0,SIZE-1);
        int A2count=count;
        count=0;
        for (int i=0;i<SIZE;i++){
                printf("%d ",A2[i]);
        }
        printf("\\n\\n");
        printf("Before sort A3: ");
        for(int i=0;i<SIZE;i++){
                printf("%d ",A3[i]);
        }
        printf("\\n\\n");
        printf("After sort A3: ");
        MergeSort(A3,0,SIZE-1);
        int A3count=count;
        count=0;

        for(int i=0;i<SIZE;i++){
                printf("%d ", A3[i]);
        }
        printf("\\n\\n\\n");
        printf("A1 comparison count : %d\n",A1count);
        printf("A2 comparison count : %d\n",A2count);
        printf("A3 comparison count : %d\n",A3count);
}
```

Listing 2: algo-2.c

### 3.3 result

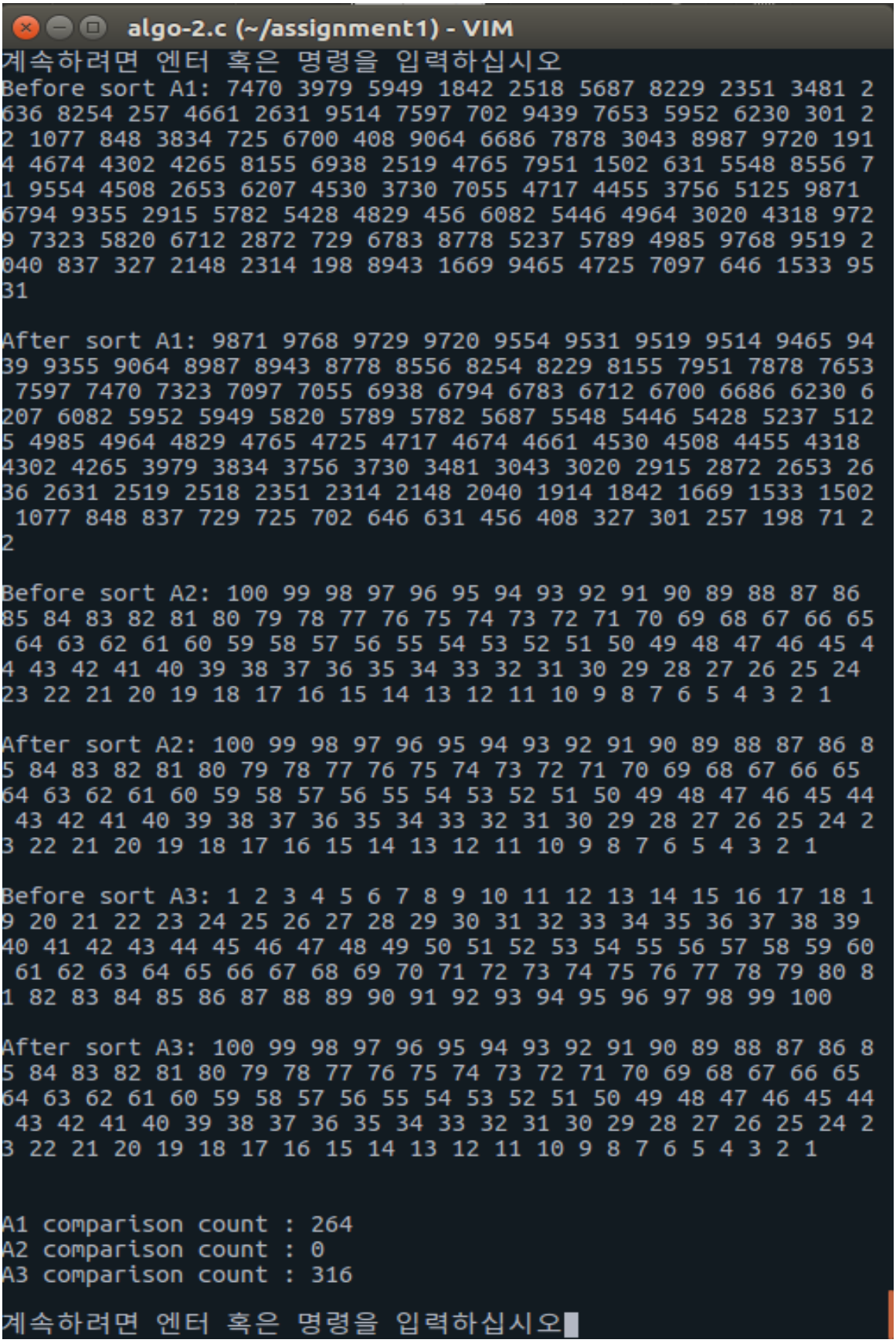



Figure 2: Executing result of algo-2.c

# 4 Algo-3

## 4.1 problem

Write functions which perform according to the following descriptions. The input to each function is a linked list of integers.

## 4.2 source code

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

#define randomize() srand(time(NULL))
#define random(n) (rand()%(n))
#define SIZE 100
#define SAMPLE 20

typedef enum { false, true } bool;
typedef int Data;

typedef struct _Node {
        Data item;
        struct _Node* next;
} Node;

typedef struct {
        Node* head;
        int len;
} LinkedList;

// Make a list empty.
void InitList(LinkedList* plist) {
        // Create a dummy node;
        plist->head = (Node*)malloc(sizeof(Node));
        plist->head->next = NULL;
        plist->len = 0;
}

// Check whether the list is empty.
bool IsEmpty(LinkedList* plist) {
        return plist->len == 0;
}

// Insert an item at the k-th position.
void InsertMiddle(LinkedList* plist, int pos, Data item) {
        Node *cur, *newNode;
        if (pos < 0 || pos > plist->len)
                exit(1);

        // Create a new node.
        newNode = (Node*)malloc(sizeof(Node));
        newNode->item = item;
        newNode->next = NULL;

        // Move the cur pointer to the (k-1)-th position.
        cur = plist->head;
        for (int i = 0; i < pos; i++)
        cur = cur->next;

        // Insert the new node to the k-th position.
        newNode->next = cur->next;
        cur->next = newNode;
        plist->len++;
}

// Insert an item at the last position.
void insert_last(LinkedList* plist, Data item) {
        InsertMiddle(plist, plist->len, item);
```

```
}


// Remove an item at the k-th position.
void delete_value(LinkedList* plist, int value) {
        Node *cur, *temp, *prev;

        if(IsEmpty(plist)) exit(1);

        cur = plist->head;
        prev = plist->head;
        cur = cur->next;
        prev->next=cur;

        while(cur!=NULL){
                int check=0;
                while(cur!=NULL){
                        if (cur->item==value){
                                check=1;
                                break;
                        }
                        prev=cur;
                        cur=cur->next;
                        prev->next=cur;
                }
                if(check==1){
                        temp = cur;
                        cur=cur->next;
                        prev->next=cur;
                        plist->len--;
                        free(temp);
                }
        }
}

// Print each item in a list in sequence.
void PrintList(LinkedList* plist) {
        int check=0;
        int count=0;
        for (Node* cur = plist->head->next; cur != NULL; cur = cur->next){

                if(count<(plist->len)/2.0){
                        printf("%d ", cur->item);
                        count++;
                }
                else{
                        if(check==1){
                                printf("%d ",cur->item);
                        }
                        else{
                                printf("\\n%d ",cur->item);
                                check=1;
                        }
                }
        }
}

int main(){
        LinkedList list;
        int A[SIZE];

        InitList(\&list);
        randomize();
        for(int i=0;i<SIZE;i++){
                A[i]=random(SAMPLE);
                insert_last(\&list, A[i]);
        }
        int flag;
        printf("Select delete(0) or Print(1) and if you want to stop, press -1\\n");
        scanf("%d",&flag);
```

```
        while(flag!=-1){
                if(flag==0){
                        int value=A[random(SIZE)];
                        printf("Randomly selected value :%d\n",value);
                        delete_value(\&list,value);
                        printf("value=%d is erased\n",value);
                }
                else if(flag==1){
                        printf("Print Linked List : ");
                        PrintList(\&list);
                else{
                        printf("please press correct number\\n");
                }
                printf("\\n\\n");
                printf("Select delete(0) or Print(1) and if you want to stop, press -1\\n");
                scanf("%d",&flag);
        }
}
```

Listing 3: algo-3.c

### 4.3 result

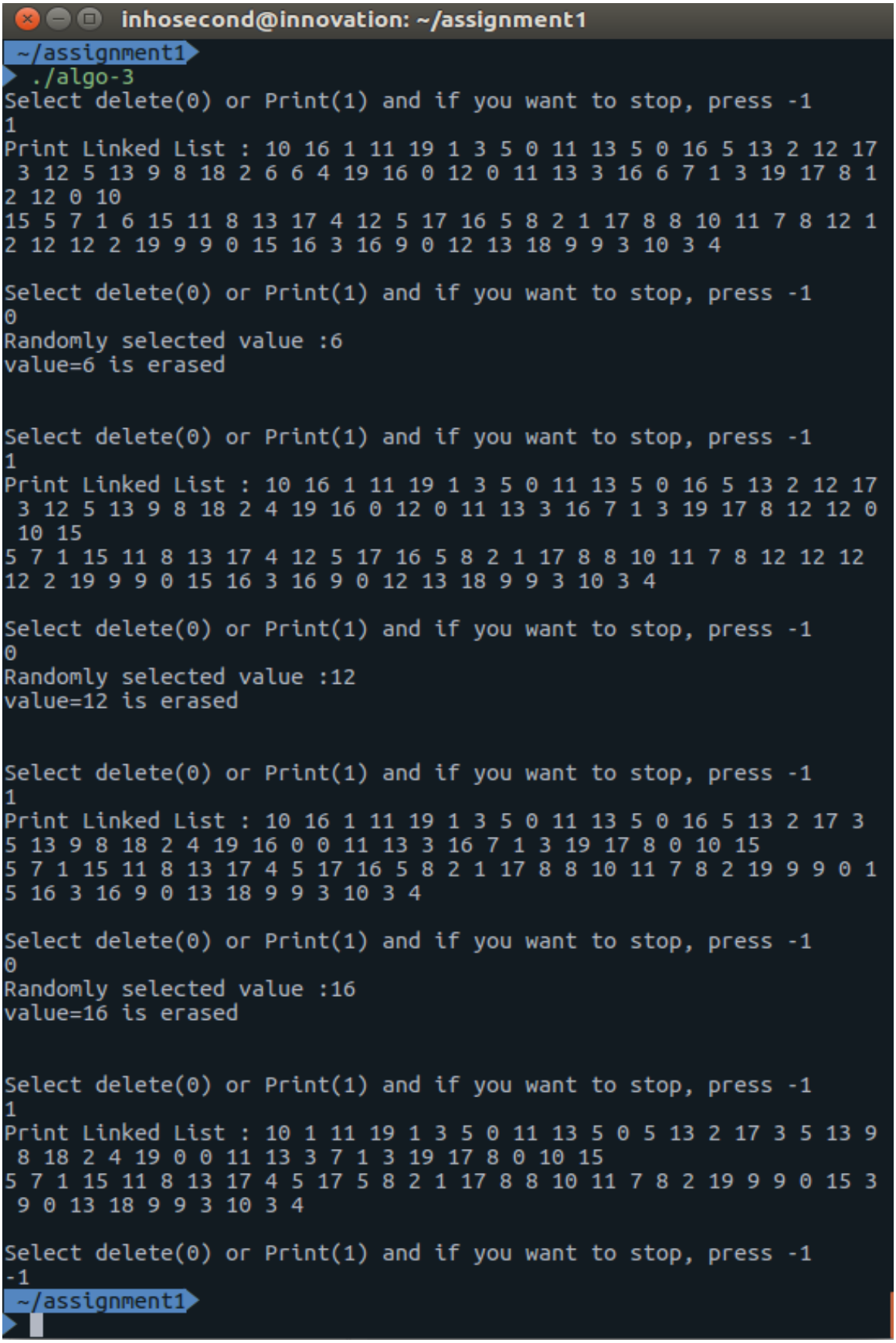



Figure 3: Executing result of algo-3.c

# 5 Algo-4

## 5.1 problem

Program the divide and conquer matrix multiplication using

(1) standard algorithm

(2) recursion

(3) strassen's method.

## 5.2 source code

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

#define randomize() srand(time(NULL))
#define random(n) (rand()%(n))
#define t1 4
#define t2 8
#define SAMPLE 1000

int count_addition=0;
int count_multiply=0;
int count_subtraction=0;

void standard_algorithm(int** A,int** B, int** C, int dim){
        int i,j,k;
        int n=dim;
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        count_addition-=1;
                        for(k=0;k<n;k++){
                                C[i][j]+=A[i][k]*B[k][j];
                                count_addition+=1;
                                count_multiply+=1;
                        }
                }
        }
}

void display(int** X,int dim){
        int i,j;
        int n=dim;
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        printf("%d ",X[i][j]);
                }
                printf("\\n");
        }
        printf("\\n");
}

void add(int **A, int **B, int **C, int dim){
        int i,j;
        int n=dim;
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        C[i][j]=A[i][j]+B[i][j];
                }
        }
}
void sub(int **A, int **B, int **C, int dim){
        int i,j;
        int n=dim;
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        C[i][j]=A[i][j]-B[i][j];
```

```
                }
        }
}

void recursion_algorithm(int** A,int** B, int** C, int haf,int dim){
        int n=dim;
        int k=haf;
        int i,j;

        int **p1=(int**)malloc(sizeof(int*)*k);
        int **p2=(int**)malloc(sizeof(int*)*k);
        int **p3=(int**)malloc(sizeof(int*)*k);
        int **p4=(int**)malloc(sizeof(int*)*k);
        int **p5=(int**)malloc(sizeof(int*)*k);
        int **p6=(int**)malloc(sizeof(int*)*k);
        int **p7=(int**)malloc(sizeof(int*)*k);
        int **p8=(int**)malloc(sizeof(int*)*k);


        int **A11=(int**)malloc(sizeof(int*)*k);
        int **A12=(int**)malloc(sizeof(int*)*k);
        int **A21=(int**)malloc(sizeof(int*)*k);
        int **A22=(int**)malloc(sizeof(int*)*k);
        int **B11=(int**)malloc(sizeof(int*)*k);
        int **B12=(int**)malloc(sizeof(int*)*k);
        int **B21=(int**)malloc(sizeof(int*)*k);
        int **B22=(int**)malloc(sizeof(int*)*k);
        int **C11=(int**)malloc(sizeof(int*)*k);
        int **C12=(int**)malloc(sizeof(int*)*k);
        int **C21=(int**)malloc(sizeof(int*)*k);
        int **C22=(int**)malloc(sizeof(int*)*k);

        for(i=0;i<k;i++){
                *(p1+i)=(int*)malloc(sizeof(int)*k);
                *(p2+i)=(int*)malloc(sizeof(int)*k);
                *(p3+i)=(int*)malloc(sizeof(int)*k);
                *(p4+i)=(int*)malloc(sizeof(int)*k);
                *(p5+i)=(int*)malloc(sizeof(int)*k);
                *(p6+i)=(int*)malloc(sizeof(int)*k);
                *(p7+i)=(int*)malloc(sizeof(int)*k);
                *(p8+i)=(int*)malloc(sizeof(int)*k);
                *(A11+i)=(int*)malloc(sizeof(int)*k);
                *(A12+i)=(int*)malloc(sizeof(int)*k);
                *(A21+i)=(int*)malloc(sizeof(int)*k);
                *(A22+i)=(int*)malloc(sizeof(int)*k);
                *(B11+i)=(int*)malloc(sizeof(int)*k);
                *(B12+i)=(int*)malloc(sizeof(int)*k);
                *(B21+i)=(int*)malloc(sizeof(int)*k);
                *(B22+i)=(int*)malloc(sizeof(int)*k);
                *(C11+i)=(int*)malloc(sizeof(int)*k);
                *(C12+i)=(int*)malloc(sizeof(int)*k);
                *(C21+i)=(int*)malloc(sizeof(int)*k);
                *(C22+i)=(int*)malloc(sizeof(int)*k);
        }
        if(k==1){
                C[0][0]=A[0][0]*B[0][0]+A[0][1]*B[1][0];
                C[0][1]=A[0][0]*B[0][1]+A[0][1]*B[1][1];
                C[1][0]=A[1][0]*B[0][0]+A[1][1]*B[1][0];
                C[1][1]=A[1][0]*B[0][1]+A[1][1]*B[1][1];
                count_addition+=4;
                count_multiply+=8;
                return;
        }

        for(i=0;i<k;i++){
                for(j=0;j<k;j++){
                        A11[i][j]=A[i][j];
                        A12[i][j]=A[i][j+k];
                        A21[i][j]=A[i+k][j];
                        A22[i][j]=A[i+k][j+k];
```

```
                B11[i][j]=B[i][j];
                B12[i][j]=B[i][j+k];
                B21[i][j]=B[i+k][j];
                B22[i][j]=B[i+k][j+k];
        }
}
recursion_algorithm(A11,B11,p1,k/2,n);
recursion_algorithm(A12,B21,p2,k/2,n);
recursion_algorithm(A11,B12,p3,k/2,n);
recursion_algorithm(A12,B22,p4,k/2,n);
recursion_algorithm(A21,B11,p5,k/2,n);
recursion_algorithm(A22,B21,p6,k/2,n);
recursion_algorithm(A21,B12,p7,k/2,n);
recursion_algorithm(A22,B22,p8,k/2,n);

add(p1,p2,C11,k);
add(p3,p4,C12,k);
add(p5,p6,C21,k);
add(p7,p8,C22,k);
count_addition+=4;

display(A11,k);
display(A12,k);
display(A21,k);
display(A22,k);

display(B11,k);
display(B12,k);
display(B21,k);
display(B22,k);

display(C11,k);
display(C12,k);
display(C21,k);
display(C22,k);

for(i=0;i<k;i++){
        for(j=0;j<k;j++){
                C[i][j]=C11[i][j];
                C[i][j+k]=C12[i][j];
                C[i+k][j]=C21[i][j];
                C[i+k][j+k]=C22[i][j];
        }
}

for(i=0;i<k;i++){
        free(*(p1+i));
        free(*(p2+i));
        free(*(p3+i));
        free(*(p4+i));
        free(*(p5+i));
        free(*(p6+i));
        free(*(p7+i));
        free(*(p8+i));
        free(*(A11+i));
        free(*(A12+i));
        free(*(A21+i));
        free(*(A22+i));
        free(*(B11+i));
        free(*(B12+i));
        free(*(B21+i));
        free(*(B22+i));
        free(*(C11+i));
        free(*(C12+i));
        free(*(C21+i));
        free(*(C22+i));
}
free(p1);
free(p2);
free(p3);
```

```
        free(p4);
        free(p5);
        free(p6);
        free(p7);
        free(p8);
        free(A11);
        free(A12);
        free(A21);
        free(A22);
        free(B11);
        free(B12);
        free(B21);
        free(B22);
        free(C11);
        free(C12);
        free(C21);
        free(C22);
}

void strassen(int** A, int** B, int** C,int haf, int dim){
        int n=dim;
        int k=haf;
        int i,j;
        int P1,P2,P3,P4,P5,P6,P7;
        if(k==1){
                P1=A[0][0]*(B[0][1]-B[1][1]);
                P2=(A[0][0]+A[0][1])*B[1][1];
                P3=(A[1][0]+A[1][1])*B[0][0];
                P4=A[1][1]*(B[1][0]-B[0][0]);
                P5=(A[0][0]+A[1][1])*(B[0][0]+B[1][1]);
                P6=(A[0][1]-A[1][1])*(B[1][0]+B[1][1]);
                P7=(A[0][0]-A[1][0])*(B[0][0]+B[0][1]);
                C[0][0]=P5+P4-P2+P6;
                C[0][1]=P1+P2;
                C[1][0]=P3+P4;
                C[1][1]=P5+P1-P3-P7;
                count_multiply+=7;
                count_subtraction+=7;
                count_addition+=11;
                return;
        }
        int **p1=(int**)malloc(sizeof(int*)*k);
        int **p2=(int**)malloc(sizeof(int*)*k);
        int **p3=(int**)malloc(sizeof(int*)*k);
        int **p4=(int**)malloc(sizeof(int*)*k);
        int **p5=(int**)malloc(sizeof(int*)*k);
        int **p6=(int**)malloc(sizeof(int*)*k);
        int **p7=(int**)malloc(sizeof(int*)*k);


        int **A11=(int**)malloc(sizeof(int*)*k);
        int **A12=(int**)malloc(sizeof(int*)*k);
        int **A21=(int**)malloc(sizeof(int*)*k);
        int **A22=(int**)malloc(sizeof(int*)*k);
        int **B11=(int**)malloc(sizeof(int*)*k);
        int **B12=(int**)malloc(sizeof(int*)*k);
        int **B21=(int**)malloc(sizeof(int*)*k);
        int **B22=(int**)malloc(sizeof(int*)*k);
        int **C11=(int**)malloc(sizeof(int*)*k);
        int **C12=(int**)malloc(sizeof(int*)*k);
        int **C21=(int**)malloc(sizeof(int*)*k);
        int **C22=(int**)malloc(sizeof(int*)*k);


        int **tmp1=(int**)malloc(sizeof(int*)*k);
        int **tmp2=(int**)malloc(sizeof(int*)*k);


        for(i=0;i<k;i++){
                *(p1+i)=(int*)malloc(sizeof(int)*k);
```

```
            *(p2+i)=(int*)malloc(sizeof(int)*k);
            *(p3+i)=(int*)malloc(sizeof(int)*k);
            *(p4+i)=(int*)malloc(sizeof(int)*k);
            *(p5+i)=(int*)malloc(sizeof(int)*k);
            *(p6+i)=(int*)malloc(sizeof(int)*k);
            *(p7+i)=(int*)malloc(sizeof(int)*k);
            *(A11+i)=(int*)malloc(sizeof(int)*k);
            *(A12+i)=(int*)malloc(sizeof(int)*k);
            *(A21+i)=(int*)malloc(sizeof(int)*k);
            *(A22+i)=(int*)malloc(sizeof(int)*k);
            *(B11+i)=(int*)malloc(sizeof(int)*k);
            *(B12+i)=(int*)malloc(sizeof(int)*k);
            *(B21+i)=(int*)malloc(sizeof(int)*k);
            *(B22+i)=(int*)malloc(sizeof(int)*k);
            *(C11+i)=(int*)malloc(sizeof(int)*k);
            *(C12+i)=(int*)malloc(sizeof(int)*k);
            *(C21+i)=(int*)malloc(sizeof(int)*k);
            *(C22+i)=(int*)malloc(sizeof(int)*k);
            *(tmp1+i)=(int*)malloc(sizeof(int)*k);
            *(tmp2+i)=(int*)malloc(sizeof(int)*k);
    }

    for(i=0;i<k;i++){
            for(j=0;j<k;j++){
                    A11[i][j]=A[i][j];
                    A12[i][j]=A[i][j+k];
                    A21[i][j]=A[i+k][j];
                    A22[i][j]=A[i+k][j+k];
                    B11[i][j]=B[i][j];
                    B12[i][j]=B[i][j+k];
                    B21[i][j]=B[i+k][j];
                    B22[i][j]=B[i+k][j+k];
            }
    }

    sub(B12,B22,tmp1,k);
    strassen(A11,tmp1,p1,k/2,n);      //p1
    add(A11,A12,tmp1,k);
    strassen(tmp1,B22,p2,k/2,n);      //p2
    add(A21,A22,tmp1,k);
    strassen(tmp1,B11,p3,k/2,n);      //p3
    sub(B21,B11,tmp1,k);
    strassen(A22,tmp1,p4,k/2,n);      //p4
    add(A11,A22,tmp1,k);
    add(B11,B22,tmp2,k);
    strassen(tmp1,tmp2,p5,k/2,n);     //p5
    sub(A12,A22,tmp1,k);
    add(B21,B22,tmp2,k);
    strassen(tmp1,tmp2,p6,k/2,n);     //p6
    sub(A11,A21,tmp1,k);
    add(B11,B12,tmp2,k);
    strassen(tmp1,tmp2,p7,k/2,n);     //p7

    add(p5,p4,tmp1,k);
    sub(tmp1,p2,tmp2,k);
    add(tmp2,p6,C11,k);         //C11
    add(p1,p2,C12,k);           //C12
    add(p3,p4,C21,k);           //C21
    add(p5,p1,tmp1,k);
    sub(tmp1,p3,tmp2,k);
    sub(tmp2,p7,C22,k);         //C22

    display(A11,k);
    display(A12,k);
    display(A21,k);
    display(A22,k);

    display(B11,k);
    display(B12,k);
    display(B21,k);
```

```
        display(B22,k);

        display(C11,k);
        display(C12,k);
        display(C21,k);
        display(C22,k);

        count_addition+=11;
        count_subtraction+=7;
        for(i=0;i<k;i++){
                for(j=0;j<k;j++){
                        C[i][j]=C11[i][j];
                        C[i][j+k]=C12[i][j];
                        C[i+k][j]=C21[i][j];
                        C[i+k][j+k]=C22[i][j];
                }
        }

        for(i=0;i<k;i++){
                free(*(p1+i));
                free(*(p2+i));
                free(*(p3+i));
                free(*(p4+i));
                free(*(p5+i));
                free(*(p6+i));
                free(*(p7+i));
                free(*(A11+i));
                free(*(A12+i));
                free(*(A21+i));
                free(*(A22+i));
                free(*(B11+i));
                free(*(B12+i));
                free(*(B21+i));
                free(*(B22+i));
                free(*(C11+i));
                free(*(C12+i));
                free(*(C21+i));
                free(*(C22+i));
                free(*(tmp1+i));
                free(*(tmp2+i));
        }

        free(p1);
        free(p2);
        free(p3);
        free(p4);
        free(p5);
        free(p6);
        free(p7);
        free(A11);
        free(A12);
        free(A21);
        free(A22);
        free(B11);
        free(B12);
        free(B21);
        free(B22);
        free(C11);
        free(C12);
        free(C21);
        free(C22);
        free(tmp1);
        free(tmp2);
}

void init(int** X,int dim){
        int i,j;
        int n=dim;
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
```

```
                        X[i][j]=0;
                }
        }
}

int main(){

        int **A1 = (int**)malloc(sizeof(int *)*t1);
        int **B1 = (int**)malloc(sizeof(int *)*t1);
        int **C1 = (int**)malloc(sizeof(int *)*t1);
        int **A2 = (int**)malloc(sizeof(int *)*t2);
        int **B2 = (int**)malloc(sizeof(int *)*t2);
        int **C2 = (int**)malloc(sizeof(int *)*t2);
        int i,j;

        randomize();
        for(i=0;i<t1;i++){
                *(A1+i)=(int*)malloc(sizeof(int)*t1);
                *(B1+i)=(int*)malloc(sizeof(int)*t1);
                *(C1+i)=(int*)malloc(sizeof(int)*t1);
                for(j=0;j<t1;j++){
                        A1[i][j]=random(SAMPLE);
                        B1[i][j]=random(SAMPLE);
                        C1[i][j]=0;
                }
        }

        for(i=0;i<t2;i++){
                *(A2+i)=(int*)malloc(sizeof(int)*t2);
                *(B2+i)=(int*)malloc(sizeof(int)*t2);
                *(C2+i)=(int*)malloc(sizeof(int)*t2);
                for(j=0;j<t2;j++){
                        A2[i][j]=random(SAMPLE);
                        B2[i][j]=random(SAMPLE);
                        C2[i][j]=0;
                }
        }

        printf("Matrix A1 is :\n");
        display(A1,t1);
        printf("\n");

        printf("Matrix B1 is :\n");
        display(B1,t1);
        printf("\n");
        printf("\nBy using standard algorithm Matrix C1 is :\n");
        standard_algorithm(A1,B1,C1,t1);
        int count_standard_addition_C1=count_addition;
        int count_standard_multiply_C1=count_multiply;
        count_addition=0;
        count_multiply=0;
        display(C1,t1);
        printf("\n");

        printf("addition : %d, multiplication : %d\n",
        count_standard_addition_C1, count_standard_multiply_C1);

        printf("\nBy using recursion Matrix C1 is :\n");
        init(C1,t1);
        recursion_algorithm(A1,B1,C1,t1/2,t1);
        display(C1,t1);
        printf("\nPartial matrix completion\n");
        printf("\nRecursion Matrix C1 :\n");
        int count_recursion_addition_C1=count_addition;
        int count_recursion_multiply_C1=count_multiply;
        count_addition=0;
        count_multiply=0;
        display(C1,t1);
        printf("\n");
```

```
printf("addition : %d, multiplication : %d\n",
count_recursion_addition_C1, count_recursion_multiply_C1);
printf("\n");
printf("\nBy using strassen algorithm Matrix C1 is :\n");
init(C1,t1);
strassen(A1,B1,C1,t1/2,t1);
display(C1,t1);
printf("\nPartial matrix completion\n");
printf("\nStrassen Matrix C1 :\n");

int count_strassen_addition_C1=count_addition;
int count_strassen_multiply_C1=count_multiply;
int count_strassen_subtraction_C1=count_subtraction;
count_addition=0;
count_multiply=0;
count_subtraction=0;
display(C1,t1);
printf("\n");

printf("addition : %d, multiplication : %d, subtraction : %d\n",
count_strassen_addition_C1, count_strassen_multiply_C1,count_strassen_subtraction_C1);

printf("\n");
printf("Matrix A2 is :\n");
display(A2,t2);
printf("\n");

printf("Matrix B2 is :\n");
display(B2,t2);
printf("\n");
printf("\nBy using standard algorithm Matrix C2 is :\n");
standard_algorithm(A2,B2,C2,t2);
int count_standard_addition_C2=count_addition;
int count_standard_multiply_C2=count_multiply;
count_addition=0;
count_multiply=0;
display(C2,t2);
printf("\n");

printf("addition : %d, multiplication : %d\n",
count_standard_addition_C2, count_standard_multiply_C2);

printf("\nBy using recursion Matrix C2 is :\n");
init(C2,t2);
recursion_algorithm(A2,B2,C2,t2/2,t2);
display(C2,t2);
printf("\nPartial matrix completion\n");
printf("\nRecursion Matrix C2 :\n");
int count_recursion_addition_C2=count_addition;
int count_recursion_multiply_C2=count_multiply;
count_addition=0;
count_multiply=0;
display(C2,t2);
printf("\n");

printf("addition : %d, multiplication : %d\n",
count_recursion_addition_C2, count_recursion_multiply_C2);

printf("\nBy using strassen algorithm Matrix C2 is :\n");
init(C2,t2);
strassen(A2,B2,C2,t2/2,t2);
display(C2,t2);
printf("\nPartial matrix completion\n");
printf("\nStrassen Matrix C2 :\n");

int count_strassen_addition_C2=count_addition;
int count_strassen_multiply_C2=count_multiply;
int count_strassen_subtraction_C2=count_subtraction;
count_addition=0;
count_multiply=0;
```

```
        count_subtraction=0;
        display(C2,t2);
        printf("\n");

        printf("addition : %d, multiplication : %d, subtraction : %d\n",
        count_strassen_addition_C2, count_strassen_multiply_C2, count_strassen_subtraction_C2);
        for(i=0;i<t1;i++){
                free(*(A1+i));
                free(*(B1+i));
                free(*(C1+i));
        }
        for(i=0;i<t2;i++){
                free(*(A2+i));
                free(*(B2+i));
                free(*(C2+i));
        }

        free(A1);
        free(B1);
        free(C1);
        free(A2);
        free(B2);
        free(C2);
}
```

Listing 4: algo-4.c

### 5.3 result

(1) standard algorithm

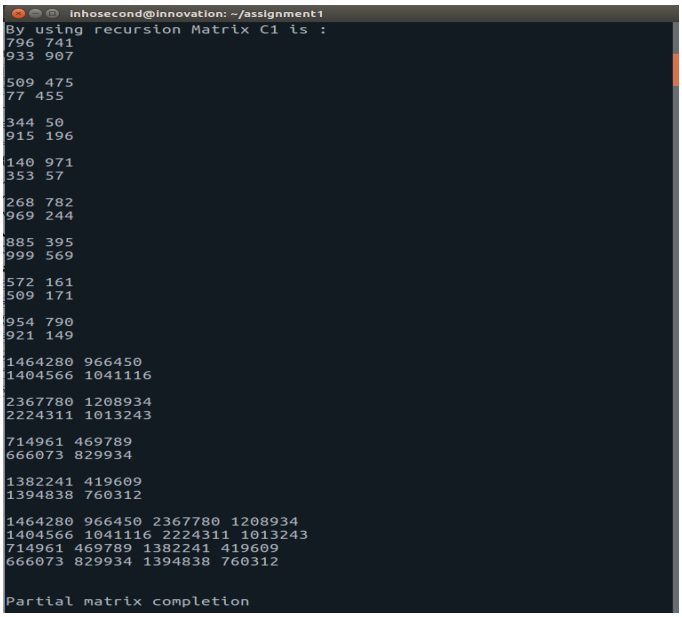
```
~/assignment1
 ./algo-4
Matrix A1 is :
796 741 509 475
933 907 77 455
344 50 140 971
915 196 353 57


Matrix B1 is :
268 782 885 395
969 244 999 569
572 161 954 790
509 171 921 149



By using standard algorithm Matrix C1 is :
1464280 966450 2367780 1208934
1404566 1041116 2224311 1013243
714961 469789 1382241 419609
666073 829934 1394838 760312


addition : 48, multiplication : 64
```

(a) standard algorithm 4*4 matrix in algo-4.c

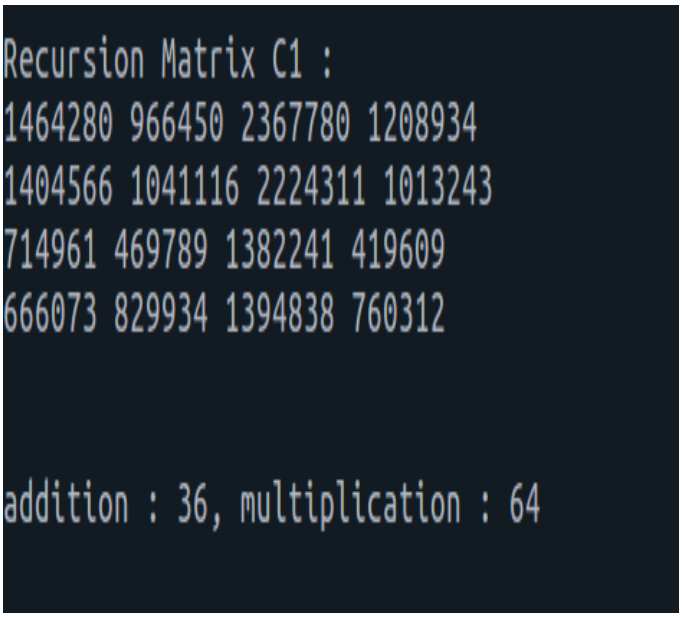
```
inhosecond@innovation: ~/assignment1

Matrix A2 is :
189 931 35 446 728 565 704 78
296 561 434 226 41 873 695 247
432 935 289 140 438 878 429 894
85 514 340 222 867 480 729 447
258 980 468 269 178 105 457 333
807 414 292 881 83 721 440 325
256 951 427 687 72 213 793 424
98 100 827 901 694 427 496 537


Matrix B2 is :
150 50 758 321 705 157 372 400
128 436 532 350 422 746 282 885
531 467 733 369 705 494 957 725
455 520 92 381 95 562 727 161
735 900 121 259 500 608 0 542
199 147 636 159 361 812 521 51
306 76 572 605 793 529 899 952
190 742 333 262 975 486 752 104



By using standard algorithm Matrix C2 is :
1246792 1513266 1581331 1294103 1795459 2303869 1676504 2098427
912954 980919 1901714 1172233 1900475 2027887 2113336 1719994
1199425 1856241 2204089 1372720 2663980 2627445 2318124 2015898
1400861 1740512 1583548 1276259 2158325 2209292 1904937 2006022
889880 1256069 1545327 1127527 1764926 1751031 1731739 1924636
1130823 1270615 1955546 1334842 2000774 2101539 2352484 1577168
1117975 1455072 1815143 1477704 2103244 2188963 2414230 2213136
1725461 2026748 1634646 1403315 2197970 2296928 2583317 1798301


addition : 448, multiplication : 512
```

(b) standard algorithm 8*8 matrix in algo-4.c

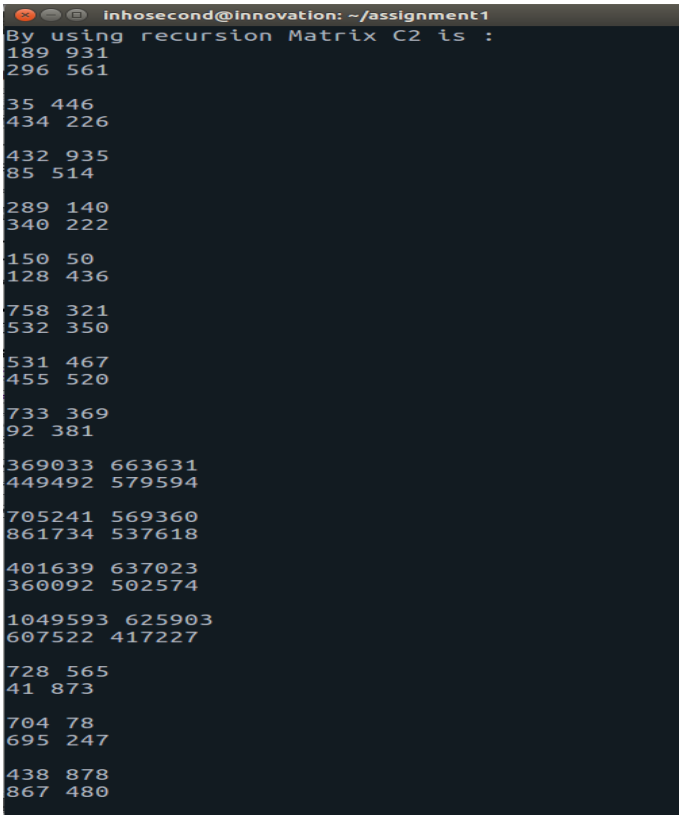
(2) recursion



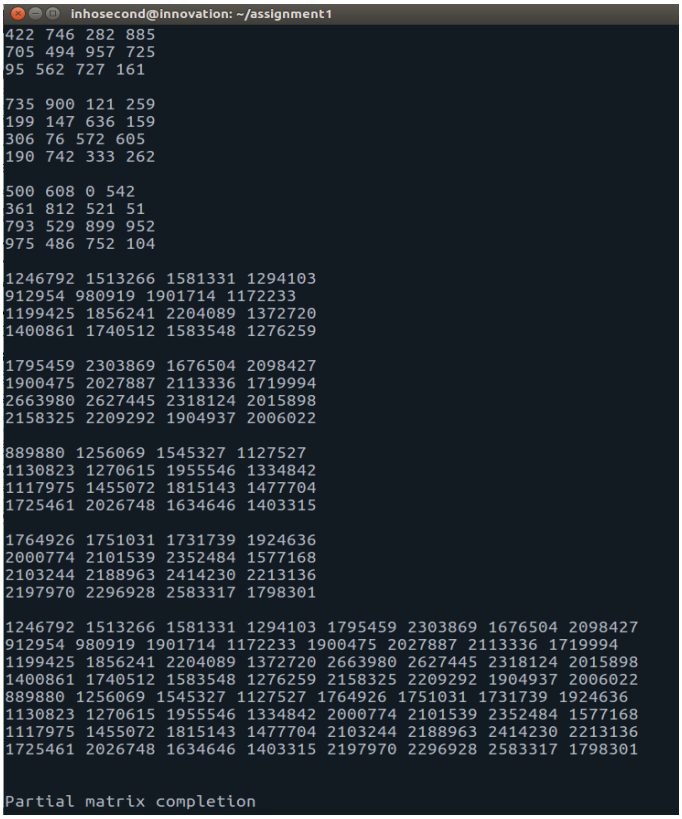

(a) recursion algorithm 4*4 partial matrix in algo-4.c



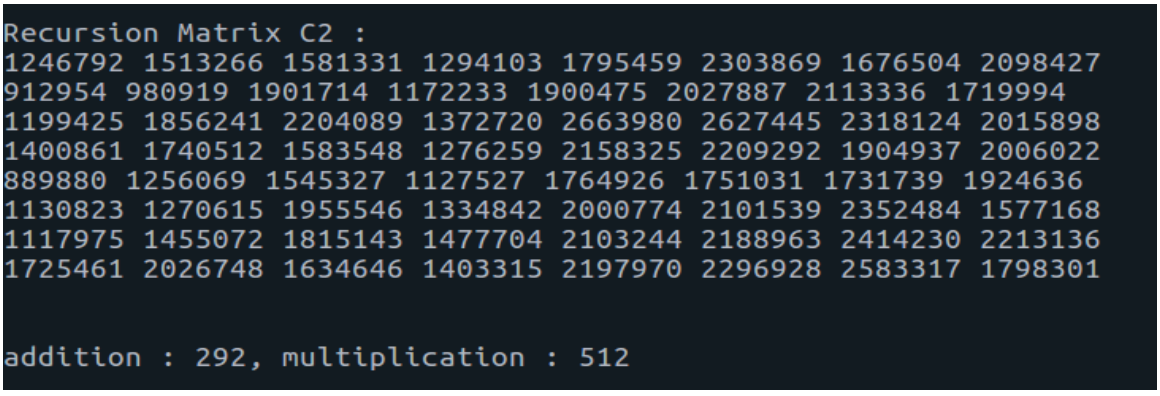

(b) recursion algorithm 4*4 matrix in algo-4.c




(c) recursion algorithm 8*8 partial matrix (1) in algo-4.c

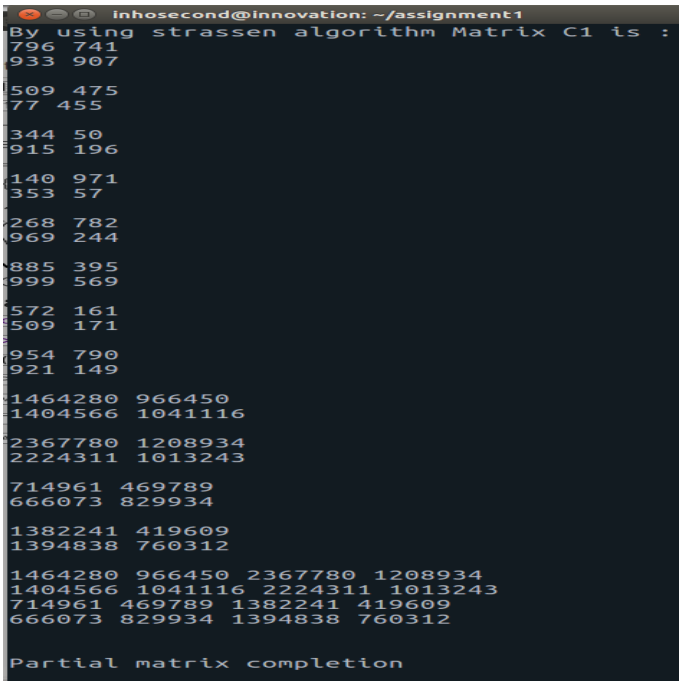



(d) recursion algorithm 8*8 partial matrix (2) in algo-4.c

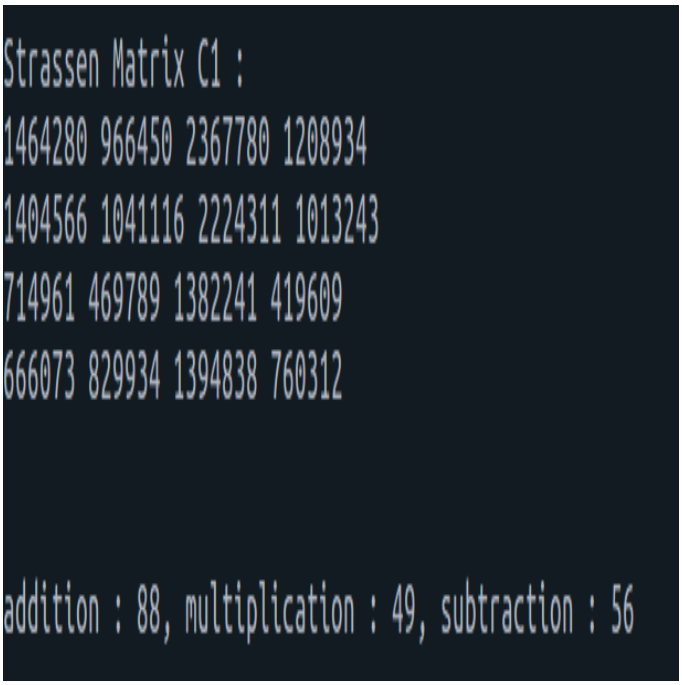



(e) recursion algorithm 8*8 matrix in algo-4.c

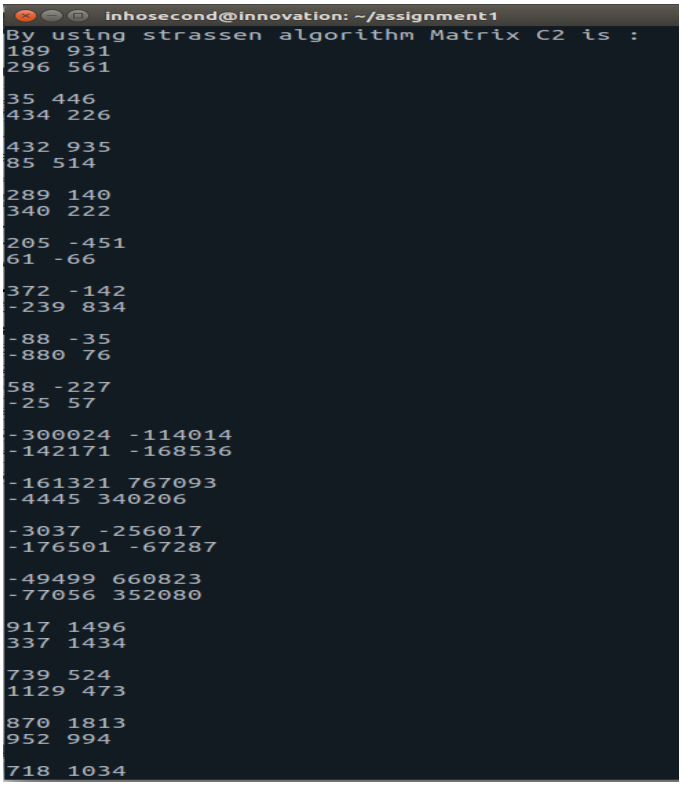
(3) strassen's method.



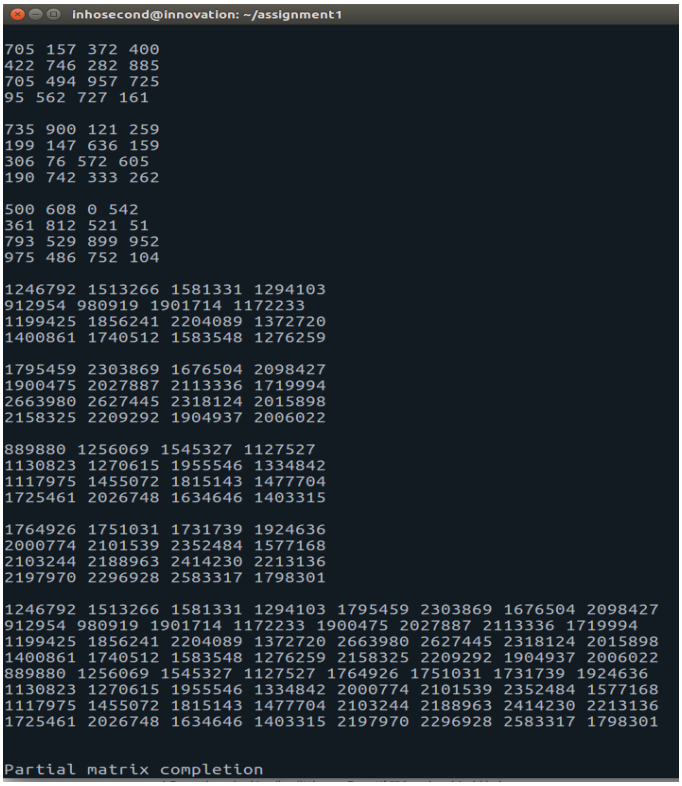

(a) strassen algorithm 4*4 partial matrix in algo-4.c



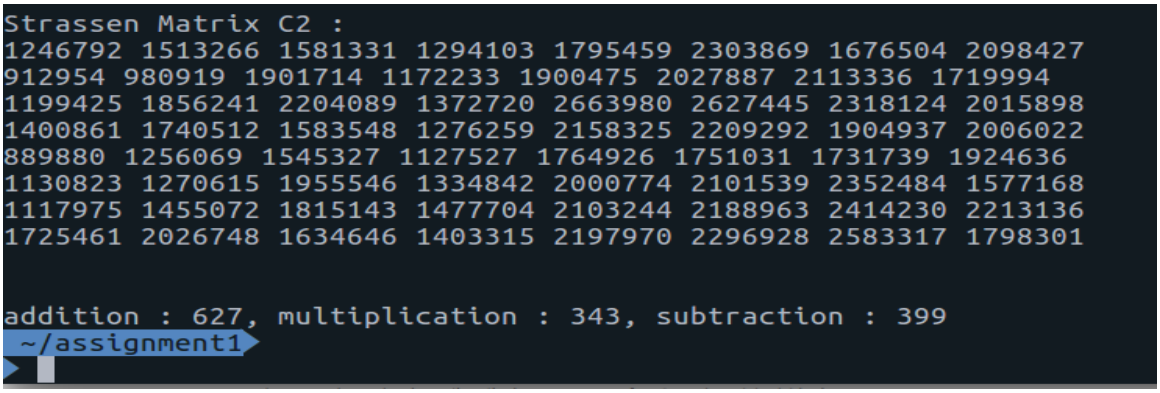

(b) strassen algorithm 4*4 matrix in algo-4.c




(c) strassen algorithm 8*8 partial matrix (1) in algo-4.c




(d) strassen algorithm 8*8 partial matrix (2) in algo-4.c




(e) strassen algorithm 8*8 matrix in algo-4.c