

# 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.length 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++;
      }
    }
    array[j + 1] = key;
  }
}
```

```

        }
        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;

```

```

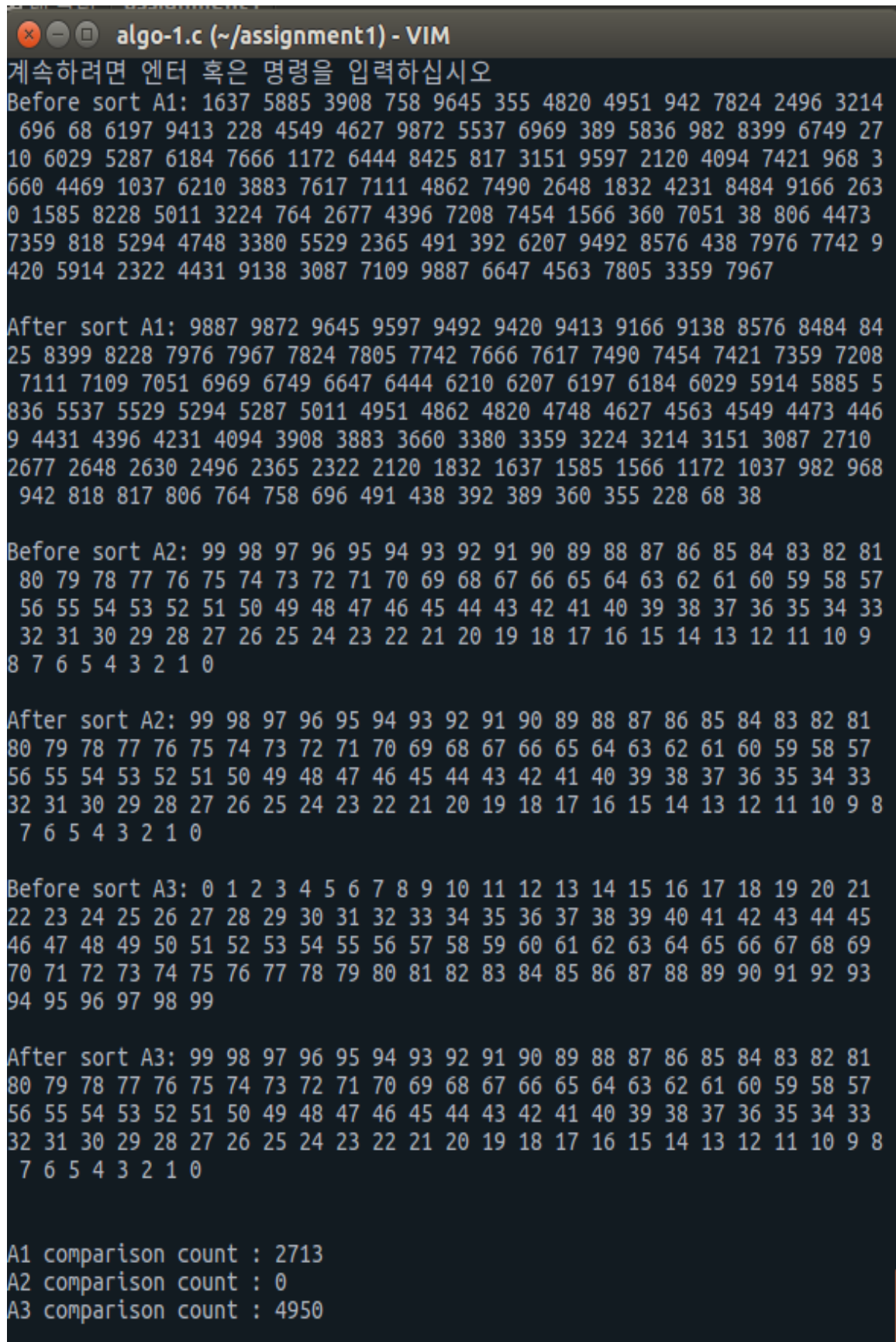
    for (int i=0;i<SIZE;i++){
        printf("%d_",A2[i]);
    }
    printf("\\n\\n");
    printf("Before_sort_A3:_");
    for(int i=0;i<SIZE;i++){
        printf("%d_",A3[i]);
    }
    printf("\\n\\n");
    printf("After_sort_A3:_");
    InsertionSort(A3, SIZE);
    int A3count=count;
    count=0;
    for(int i=0;i<SIZE;i++){
        printf("%d_", A3[i]);
    }
    printf("\\n\\n\\n\\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



```
algo-1.c (~/.assignment1) - VIM
계속하려면 엔터 혹은 명령을 입력하십시오
Before sort A1: 1637 5885 3908 758 9645 355 4820 4951 942 7824 2496 3214
696 68 6197 9413 228 4549 4627 9872 5537 6969 389 5836 982 8399 6749 27
10 6029 5287 6184 7666 1172 6444 8425 817 3151 9597 2120 4094 7421 968 3
660 4469 1037 6210 3883 7617 7111 4862 7490 2648 1832 4231 8484 9166 263
0 1585 8228 5011 3224 764 2677 4396 7208 7454 1566 360 7051 38 806 4473
7359 818 5294 4748 3380 5529 2365 491 392 6207 9492 8576 438 7976 7742 9
420 5914 2322 4431 9138 3087 7109 9887 6647 4563 7805 3359 7967

After sort A1: 9887 9872 9645 9597 9492 9420 9413 9166 9138 8576 8484 84
25 8399 8228 7976 7967 7824 7805 7742 7666 7617 7490 7454 7421 7359 7208
7111 7109 7051 6969 6749 6647 6444 6210 6207 6197 6184 6029 5914 5885 5
836 5537 5529 5294 5287 5011 4951 4862 4820 4748 4627 4563 4549 4473 446
9 4431 4396 4231 4094 3908 3883 3660 3380 3359 3224 3214 3151 3087 2710
2677 2648 2630 2496 2365 2322 2120 1832 1637 1585 1566 1172 1037 982 968
942 818 817 806 764 758 696 491 438 392 389 360 355 228 68 38

Before sort A2: 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81
80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57
56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9
8 7 6 5 4 3 2 1 0

After sort A2: 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81
80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57
56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8
7 6 5 4 3 2 1 0

Before sort A3: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99

After sort A3: 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81
80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57
56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8
7 6 5 4 3 2 1 0

A1 comparison count : 2713
A2 comparison count : 0
A3 comparison count : 4950
```

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];
```

```

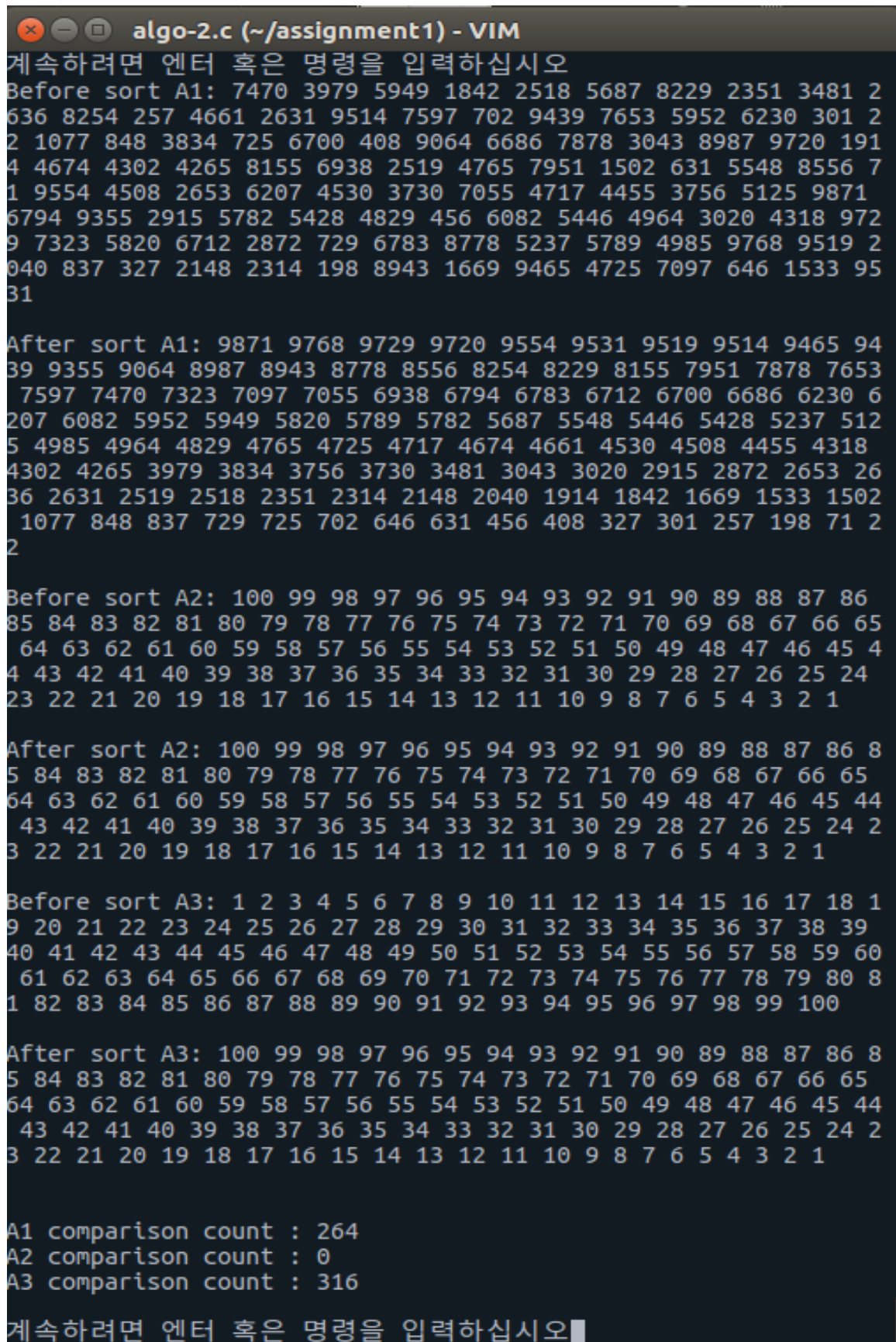
randomize ();
for (int i=0;i<SIZE;i++){
    while(1){
        int check=0;
        A1[i]=random(SAMPLE);
        for (int j=0;j<i;j++){
            if (A1[j]==A1[i]){
                check=1;
                break;
            }
        }
        if (check==1){
            continue;
        }
        else{
            break;
        }
    }
}
sorted_fill(A2);
for (int i=0;i<SIZE;i++){
    A3[i]=A2[SIZE-1-i];
}
printf("Before_sort_A1: \n");
for (int i=0;i<SIZE;i++){
    printf("%d\n",A1[i]);
}
printf("\n\n");
printf("After_sort_A1: \n");
MergeSort(A1,0,SIZE-1);
int A1count=count;
count=0;
for (int i=0;i<SIZE;i++){
    printf("%d\n",A1[i]);
}
printf("\n\n");
printf("Before_sort_A2: \n");
for (int i=0;i<SIZE;i++){
    printf("%d\n",A2[i]);
}
printf("\n\n");
printf("After_sort_A2: \n");
MergeSort(A2,0,SIZE-1);
int A2count=count;
count=0;
for (int i=0;i<SIZE;i++){
    printf("%d\n",A2[i]);
}
printf("\n\n");
printf("Before_sort_A3: \n");
for (int i=0;i<SIZE;i++){
    printf("%d\n",A3[i]);
}
printf("\n\n");
printf("After_sort_A3: \n");
MergeSort(A3,0,SIZE-1);
int A3count=count;
count=0;

for (int i=0;i<SIZE;i++){
    printf("%d\n", 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



```
algo-2.c (~/.assignment1) - VIM
계속하려면 엔터 혹은 명령을 입력하십시오
Before sort A1: 7470 3979 5949 1842 2518 5687 8229 2351 3481 2
636 8254 257 4661 2631 9514 7597 702 9439 7653 5952 6230 301 2
2 1077 848 3834 725 6700 408 9064 6686 7878 3043 8987 9720 191
4 4674 4302 4265 8155 6938 2519 4765 7951 1502 631 5548 8556 7
1 9554 4508 2653 6207 4530 3730 7055 4717 4455 3756 5125 9871
6794 9355 2915 5782 5428 4829 456 6082 5446 4964 3020 4318 972
9 7323 5820 6712 2872 729 6783 8778 5237 5789 4985 9768 9519 2
040 837 327 2148 2314 198 8943 1669 9465 4725 7097 646 1533 95
31

After sort A1: 9871 9768 9729 9720 9554 9531 9519 9514 9465 94
39 9355 9064 8987 8943 8778 8556 8254 8229 8155 7951 7878 7653
7597 7470 7323 7097 7055 6938 6794 6783 6712 6700 6686 6230 6
207 6082 5952 5949 5820 5789 5782 5687 5548 5446 5428 5237 512
5 4985 4964 4829 4765 4725 4717 4674 4661 4530 4508 4455 4318
4302 4265 3979 3834 3756 3730 3481 3043 3020 2915 2872 2653 26
36 2631 2519 2518 2351 2314 2148 2040 1914 1842 1669 1533 1502
1077 848 837 729 725 702 646 631 456 408 327 301 257 198 71 2
2

Before sort A2: 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86
85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65
64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 4
4 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

After sort A2: 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 8
5 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65
64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44
43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 2
3 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Before sort A3: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 1
9 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 8
1 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

After sort A3: 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 8
5 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65
64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44
43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 2
3 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

A1 comparison count : 264
A2 comparison count : 0
A3 comparison count : 316
계속하려면 엔터 혹은 명령을 입력하십시오
```

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

```
inhosecond@innovation: ~/assignment1
~/assignment1
> ./algo-3
Select delete(0) or Print(1) and if you want to stop, press -1
1
Print Linked List : 10 16 1 11 19 1 3 5 0 11 13 5 0 16 5 13 2 12 17
3 12 5 13 9 8 18 2 6 6 4 19 16 0 12 0 11 13 3 16 6 7 1 3 19 17 8 1
2 12 0 10
15 5 7 1 6 15 11 8 13 17 4 12 5 17 16 5 8 2 1 17 8 8 10 11 7 8 12 1
2 12 12 2 19 9 9 0 15 16 3 16 9 0 12 13 18 9 9 3 10 3 4

Select delete(0) or Print(1) and if you want to stop, press -1
0
Randomly selected value :6
value=6 is erased

Select delete(0) or Print(1) and if you want to stop, press -1
1
Print Linked List : 10 16 1 11 19 1 3 5 0 11 13 5 0 16 5 13 2 12 17
3 12 5 13 9 8 18 2 4 19 16 0 12 0 11 13 3 16 7 1 3 19 17 8 12 12 0
10 15
5 7 1 15 11 8 13 17 4 12 5 17 16 5 8 2 1 17 8 8 10 11 7 8 12 12 12
12 2 19 9 9 0 15 16 3 16 9 0 12 13 18 9 9 3 10 3 4

Select delete(0) or Print(1) and if you want to stop, press -1
0
Randomly selected value :12
value=12 is erased

Select delete(0) or Print(1) and if you want to stop, press -1
1
Print Linked List : 10 16 1 11 19 1 3 5 0 11 13 5 0 16 5 13 2 17 3
5 13 9 8 18 2 4 19 16 0 0 11 13 3 16 7 1 3 19 17 8 0 10 15
5 7 1 15 11 8 13 17 4 5 17 16 5 8 2 1 17 8 8 10 11 7 8 2 19 9 9 0 1
5 16 3 16 9 0 13 18 9 9 3 10 3 4

Select delete(0) or Print(1) and if you want to stop, press -1
0
Randomly selected value :16
value=16 is erased

Select delete(0) or Print(1) and if you want to stop, press -1
1
Print Linked List : 10 1 11 19 1 3 5 0 11 13 5 0 5 13 2 17 3 5 13 9
8 18 2 4 19 0 0 11 13 3 7 1 3 19 17 8 0 10 15
5 7 1 15 11 8 13 17 4 5 17 5 8 2 1 17 8 8 10 11 7 8 2 19 9 9 0 15 3
9 0 13 18 9 9 3 10 3 4

Select delete(0) or Print(1) and if you want to stop, press -1
-1
~/assignment1
```

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

```

inhosecond@innovation: ~/assignment1
By using recursion Matrix C1 is :
796 741
933 907

509 475
77 455

344 50
915 196

140 971
353 57

268 782
969 244

885 395
999 569

572 161
509 171

954 790
921 149

1464280 966450
1404566 1041116

2367780 1208934
2224311 1013243

714961 469789
666073 829934

1382241 419609
1394838 760312

1464280 966450 2367780 1208934
1404566 1041116 2224311 1013243
714961 469789 1382241 419609
666073 829934 1394838 760312

Partial matrix completion

```

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

```

Recursion Matrix C1 :
1464280 966450 2367780 1208934
1404566 1041116 2224311 1013243
714961 469789 1382241 419609
666073 829934 1394838 760312

addition : 36, multiplication : 64

```

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

```

inhosecond@innovation: ~/assignment1
By using recursion Matrix C2 is :
189 931
296 561

35 446
434 226

432 935
85 514

289 140
340 222

150 50
128 436

758 321
532 350

531 467
455 520

733 369
92 381

369033 663631
449492 579594

705241 569360
861734 537618

401639 637023
360092 502574

1049593 625903
607522 417227

728 565
41 873

704 78
695 247

438 878
867 480

```

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

```

inhosecond@innovation: ~/assignment1
422 746 282 885
705 494 957 725
95 562 727 161

735 900 121 259
199 147 636 159
306 76 572 605
190 742 333 262

500 608 0 542
361 812 521 51
793 529 899 952
975 486 752 104

1246792 1513266 1581331 1294103
912954 980919 1901714 1172233
1199425 1856241 2204089 1372720
1400861 1740512 1583548 1276259

1795459 2303869 1676504 2098427
1900475 2027887 2113336 1719994
2663980 2627445 2318124 2015898
2158325 2209292 1904937 2006022

889880 1256069 1545327 1127527
1130823 1270615 1955546 1334842
1117975 1455072 1815143 1477704
1725461 2026748 1634646 1403315

1764926 1751031 1731739 1924636
2000774 2101539 2352484 1577168
2103244 2188963 2414230 2213136
2197970 2296928 2583317 1798301

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

Partial matrix completion

```

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

```

Recursion Matrix C2 :
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 : 292, multiplication : 512

```

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

(3) strassen's method.

```

inhosecond@innovation: ~/assignment1
By using strassen algorithm Matrix C1 is :
796 741
933 907

509 475
77 455

344 50
915 196

140 971
353 57

268 782
969 244

885 395
999 569

572 161
509 171

954 790
921 149

1464280 966450
1404566 1041116

2367780 1208934
2224311 1013243

714961 469789
666073 829934

1382241 419609
1394838 760312

1464280 966450 2367780 1208934
1404566 1041116 2224311 1013243
714961 469789 1382241 419609
666073 829934 1394838 760312

Partial matrix completion

```

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

```

Strassen Matrix C1 :
1464280 966450 2367780 1208934
1404566 1041116 2224311 1013243
714961 469789 1382241 419609
666073 829934 1394838 760312

addition : 88, multiplication : 49, subtraction : 56

```

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

```

inhosecond@innovation: ~/assignment1
By using strassen algorithm Matrix C2 is :
189 931
296 561

35 446
434 226

432 935
85 514

289 140
340 222

205 -451
61 -66

372 -142
-239 834

-88 -35
-880 76

58 -227
-25 57

-300024 -114014
-142171 -168536

-161321 767093
-4445 340206

-3037 -256017
-176501 -67287

-49499 660823
-77056 352080

917 1496
337 1434

739 524
1129 473

870 1813
952 994

718 1034

```

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

```

inhosecond@innovation: ~/assignment1
705 157 372 400
422 746 282 885
705 494 957 725
95 562 727 161

735 900 121 259
199 147 636 159
306 76 572 605
190 742 333 262

500 608 0 542
361 812 521 51
793 529 899 952
975 486 752 104

1246792 1513266 1581331 1294103
912954 980919 1901714 1172233
1199425 1856241 2204089 1372720
1400861 1740512 1583548 1276259

1795459 2303869 1676504 2098427
1900475 2027887 2113336 1719994
2663980 2627445 2318124 2015898
2158325 2209292 1904937 2006022

889880 1256069 1545327 1127527
1130823 1270615 1955546 1334842
1117975 1455072 1815143 1477704
1725461 2026748 1634646 1403315

1764926 1751031 1731739 1924636
2000774 2101539 2352484 1577168
2103244 2188963 2414230 2213136
2197970 2296928 2583317 1798301

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

Partial matrix completion

```

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

```

Strassen Matrix C2 :
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 : 627, multiplication : 343, subtraction : 399
~/assignment1

```

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