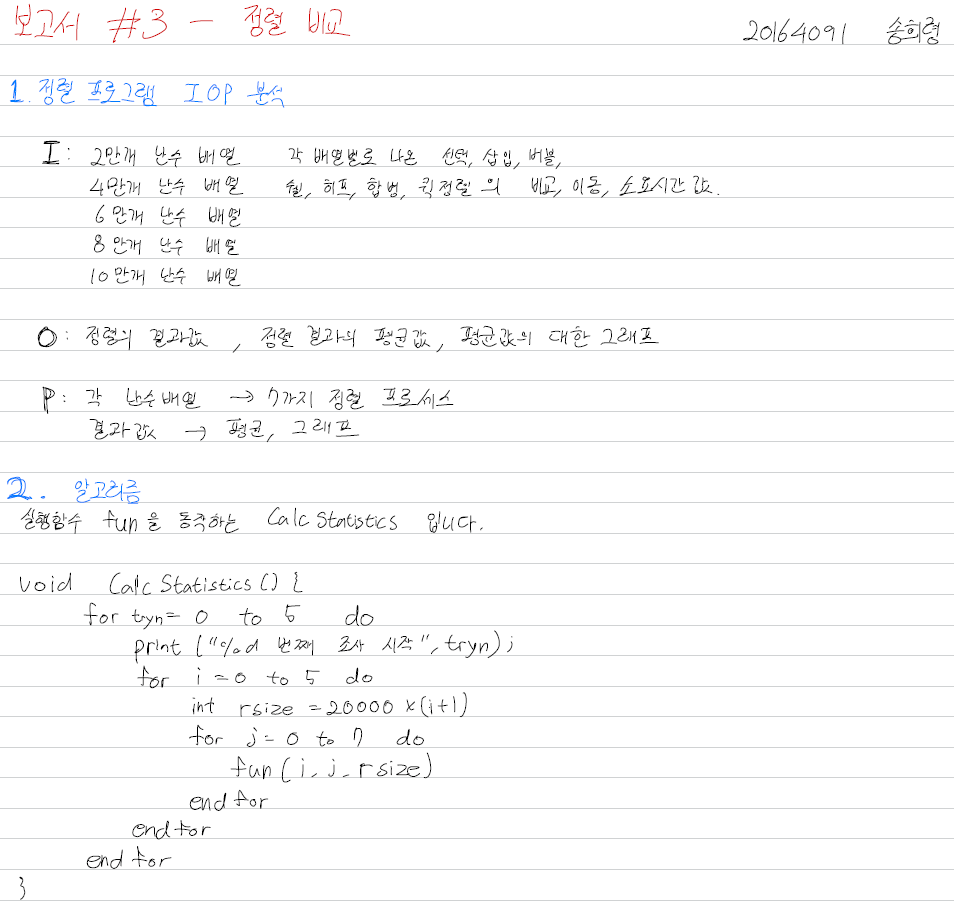
데이터구조 3

보고서 #3

(7개 정렬 분석)

20164091

송희령



3. 코드

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX 100000 //데이터 항목 최대 개수

struct avg {//데이터를 저장시킬 포맷

    double move[7][7];

    double comp[7][7];

    double clocktime[7][7];

};

struct avg avglist[5];//데이터 저장할 곳

double mvcount;//이동횟수

double cmpcount;//비교횟수

int arr1[MAX], arr2[MAX], arr3[MAX],

    arr4[MAX], arr5[MAX],arr6[MAX],arr7[MAX];//배열확인

int many = 0;

void make\_arr(int size){//크기에 맞는 배열 생성

    int x;

    srand(time(NULL));

    for(int i = 0; i < size; i++) {  //size 크기만큼 난수 생성

        x = rand() % 999 + 1;       //1부터 1000까지의 난수

        //난수 생성 후 저장

        arr1[i] = arr2[i] = arr3[i] = arr4[i] = arr5[i] =

            arr6[i] = arr7[i] = x;

    }

}

데이터를 저장할곳(struct avg)과 난수 배열을 만들어줄 코드입니다.

void swap(int\* x, int\* y){

    int temp;

    temp = \*x;

    \*x = \*y;

    \*y = temp;

}

프로그램 내의 값 교환인 swap을 생성하였습니다.

void selection\_sort(int \*arr, int size){//선택정렬

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

        int least = i;

        for (int j = i + 1; j < size; j++) {

            cmpcount++;

            if (arr[j] < arr[least]) {

                least = j;

            }

        }

        swap(&arr[least], &arr[i]);

        mvcount = mvcount + 3;  //이동 횟수

    }

}

선택정렬 입니다. 각각 비교횟수(cmpcount)와 이동횟수(mvcount)를 나타낸 것 입니다.

int insert\_sort(int\* arr, int size){//삽입정렬

    int key;

    for (int i = 1; i < size; i++) {

        int j = i - 1;

        key = arr[i];

        mvcount++;//이동횟수 증가

        cmpcount++;//비교횟수 증가

        while (j >= 0 && arr[j] > key) {

            cmpcount++;

            arr[j + 1] = arr[j];

            mvcount++;

            j = j - 1;

        }

        arr[j + 1] = key;

        mvcount++;

    }

}

삽입 정렬 입니다. 같은 방식으로 이동횟수와 비교횟수를 증가시킬곳에 배치하여 증가시켰습니다.

void bubble\_sort(int\* arr, int size)

{

    for (int i = size - 1; i > 0; i--)

    {

        for (int j = 0; j < i; j++)

        {

            cmpcount++;//비교횟수 증가

            if (arr[j] > arr[j + 1])

            {

                swap(&arr[j], &arr[j + 1]);

                mvcount = mvcount + 3;//이동횟수 증가

            }

        }

    }

}

int shell\_insert(int\* arr, int first, int last, int gap){//쉘전용 insert

    int key;

    int j;

    for (int i = first + gap; i <= last; i = i + gap) {

        j = i - gap;

        key = arr[i];

        mvcount++;

        cmpcount++;

        while (j >= first && arr[j] > key) {

            cmpcount++;

            arr[j + gap] = arr[j];

            mvcount++;

            j = j - gap;

        }

        arr[j + gap] = key;

        mvcount++;

    }

}

void shell\_sort(int\* arr, int size){//쉘정렬

    int gap;

    for (gap = size / 2; gap > 0; gap = gap / 2) {

        if (gap % 2 == 0)

            gap++;

        for (int i = 0; i < gap; i++) {

            shell\_insert(arr, i, size - 1, gap);

        }

    }

}

버블정렬과 쉘정렬, 쉘정렬에 사용할 삽입함수를 구현하였습니다.

//히프정렬

typedef struct heaptype {//히프 정의

    int heap[MAX];

    int heap\_size;

}heaptype;

heaptype\* create()//히프 생성

{

    return (heaptype\*)malloc(sizeof(heaptype));

}

void init(heaptype\* h){//히프 초기화

    h->heap\_size = 0;

}

void insert\_min\_heap(heaptype\* h, int item){//히프 삽입 함수

    int i;

    i = ++(h->heap\_size);

    while ((i != 1) && (item < h->heap[i / 2])) {

        h->heap[i] = h->heap[i / 2];

        i /= 2;

    }

    h->heap[i] = item;

}

int delete\_min\_heap(heaptype\* h){//히프 삭제 함수

    int parent = 1, child = 2;

    int item, temp;

    item = h->heap[1];

    temp = h->heap[(h->heap\_size)--];

    while (child <= h->heap\_size) {

        cmpcount++;

        if ((child < h->heap\_size) && h->heap[child] > h->heap[child + 1]) {

            child++;

        }

        cmpcount++;

        if (temp < h->heap[child]) {

            break;

        }

        h->heap[parent] = h->heap[child];

        mvcount++;

        parent = child;

        child \*= 2;

    }

    h->heap[parent] = temp;

    mvcount++;

    return item;

}

힙 정렬에 사용할 생성,초기화,삽입,삭제 함수입니다.

void heap\_sort(int arr[], int size){//히프 정렬

    int i;

    heaptype h;//히프 h를 이용하여 정렬

    init(&h);

    for (int i = 0; i < size; i++) {

        insert\_min\_heap(&h, arr[i]);

    }

    for (int i = 0; i < size; i++) {

        arr[i] = delete\_min\_heap(&h);

    }

}

위의 함수를 이용하여 히프정렬을 구현하였습니다.

void merge(int\* arr, int low, int mid, int high){//병합

    int b1, b2, e1, e2, indx;

    b1 = low;

    e1 = mid;

    b2 = mid + 1;

    e2 = high;

    indx = low;

    int sorted[MAX] = { 0 };

    while (b1 <= e1 && b2 <= e2) {

        cmpcount++;

        if (arr[b1] < arr[b2]) {

            sorted[indx++] = arr[b1++];

            mvcount++;

        }

        else{

            sorted[indx++] = arr[b2++];

            mvcount++;

        }

    }

    if (b1 > e1) {

        for (int i = b2; i <= e2; i++) {

            sorted[indx++] = arr[i];

            mvcount++;

        }

    }

    else if (b2 > e2) {

        for (int i = b1; i <= e1; i++) {

            sorted[indx++] = arr[i];

            mvcount++;

        }

    }

    for (int i = low; i <= high; i++) {

        arr[i] = sorted[i];

        mvcount++;

    }

}

void merge\_sort(int\* arr, int left, int right){//합병 정렬

    int mid;

    if (left < right) {

        mid = (left + right) / 2;

        merge\_sort(arr, left, mid);

        merge\_sort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

합병 정렬을 위한 합병 함수와 합병정렬을 구현하였습니다.

//퀵정렬

int partition(int\* arr, int left, int right)

{

    int pivot, low, high;

    low = left, high = right + 1;

    pivot = arr[left];

    do {

        do {

            low++;

            cmpcount++;

        } while (low <= right && arr[low] < pivot);

        do {

            high--;

            cmpcount++;

        } while (high >= left && arr[high] > pivot);

        if (low < high)

        {

            swap(&arr[low], &arr[high]);

            mvcount = mvcount + 3;

        }

        cmpcount++;

    } while (low < high);

    cmpcount++;

    swap(&arr[left], &arr[high]);

    mvcount = mvcount + 3;

    return high;

}

void quick\_sort(int\* arr, int left, int right)

{

    if (left < right)

    {

        int q = partition(arr, left, right);

        quick\_sort(arr, left, q - 1);

        quick\_sort(arr, q + 1, right);

    }

}

퀵정렬 함수 입니다.

int fun(int x, int y, int s){

    clock\_t start, end;

    double elapsed;

    make\_arr(s);

    mvcount = 0, cmpcount = 0;

    start = clock();

    if(y == 0) {//선택 정렬 실행

        selection\_sort(arr1,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 1) {//삽입 정렬 실행

        insert\_sort(arr2,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 2) {//버블 정렬 실행

        bubble\_sort(arr3,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 3) {//쉘 정렬 실행

        shell\_sort(arr4,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 4) {//히프 정렬 실행

        heap\_sort(arr5,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 5) {//합병 정렬 실행

        merge\_sort(arr6,0,s-1);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 6) {   //퀵  정렬 실행

        quick\_sort(arr7,0,s-1);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    end = clock();

    //실행시간 계산, 저장

    elapsed = (double)((end - start) \* 1000) / CLOCKS\_PER\_SEC;

    avglist[many].clocktime[x][y] = elapsed;

}

각 함수들을 실행시켜줄 fun입니다. y값에 맞게 삽입정렬부터 퀵정렬까지 실행하도록 합니다. 실행 후 결과값은 avglist에 저장합니다.

void calcStatistics(){//연산 호출

    for(int tryn = 0; tryn < 5; tryn++) {

        printf("★%d번째 조사 시작★\n\n", tryn+1);

        for(int i = 0; i < 5; i++) {

            int rsize = 20000 \* (i+1);

            for(int j = 0; j < 7; j++)

                fun(i, j, rsize);

        }

        many++;

        printf("\n★%d번째 조사 종료★\n\n", tryn+1);

    }

}

연산 호출 함수인 calcStatistics입니다.

void cmpSortByTable(){

    for(int tryn = 0; tryn < 5; tryn++) {

        printf("★%d번 조사 결과 출력 시작★\n\n",tryn+1);

        for(int i = 0; i < 5; i++) {

            int rsize = 20000 \* (i+1);//초기배열사이즈 설정

            printf("-----크기 = %d--------------------------------------------\n",rsize);

            printf("\t       소요시간|         비교회수|        이동회수|\n");

            for(int j = 0; j < 7; j++) {

                switch(j+1) {

                case 1:

                    printf("선택 정렬\t");

                    break;

                case 2:

                    printf("삽입 정렬\t");

                    break;

                case 3:

                    printf("버블 정렬\t");

                    break;

                case 4:

                    printf("쉘   정렬\t");

                    break;

                case 5:

                    printf("히프 정렬\t");

                    break;

                case 6:

                    printf("합병 정렬\t");

                    break;

                case 7:

                    printf("퀵   정렬\t");

                    break;

                }

                printf("%15.1f | %15.1f | %15.1f|\n", avglist[tryn].clocktime[i][j],

                       avglist[tryn].comp[i][j],avglist[tryn].move[i][j]);

            }

        }

        printf("\n★%d번째 조사 결과 출력 종료★\n\n", tryn+1);

    }

    printf("\n★평균출력★\n");

    for(int i = 0; i < 5; i++) {

        printf("★%d 사이즈의 평균★\n",20000\*(i+1));

        for(int j = 0; j < 7; j++) {

            double mavg,cavg,tavg;

            mavg = cavg = tavg = 0;

            for(int k = 0; k < 5; k++) {

                mavg += avglist[k].move[i][j];

                cavg += avglist[k].comp[i][j];

                tavg += avglist[k].clocktime[i][j];

            }

            switch(j+1) {

            case 1:

                printf("선택 정렬 평균\n");

                break;

            case 2:

                printf("삽입 정렬 평균\n");

                break;

            case 3:

                printf("버블 정렬 평균\n");

                break;

            case 4:

                printf("쉘   정렬 평균\n");

                break;

            case 5:

                printf("히프 정렬 평균\n");

                break;

            case 6:

                printf("합병 정렬 평균\n");

                break;

            case 7:

                printf("퀵   정렬 평균\n");

                break;

            }

            printf("        소요시간|         비교회수|        이동회수|\n");

            printf("%15.1f | %15.1f | %15.1f|\n", tavg/5, cavg/5, mavg/5);

        }

    }

}

이후 정렬 결과들의 출력과 평균을 구한 출력을 해줄 cmpSortByTable입니다.

int main(void){

    calcStatistics();

    cmpSortByTable();

    //cmpSortByGraph(stat)

    //그래프는 직접 그리겠습니다.

}

이 모든 함수를 실행할 main함수 부분 입니다.

연산을 위해 calcStatistics를 호출하고

출력을 위해 cmpSortByTable을 호출합니다

그래프는 python을 이용하여 출력해 보겠습니다.

from matplotlib import pyplot as plt

SelectTime = [835.8,3262.2,7406.4,13317.4,20710.2]

SelectCompare =[199990000.0,799980000.0,1799970000.0,3199960000.0,4999950000.0]

SelectMove = [59997.0,1199970.0,179997.0,239997.0,299997.0]

InsertTime = [410.8,1644.2,3720.2,6634.6,10328.6]

InsertCompare =[100192419.6,400055475.0,901263721.8,1598681399.4,2499912332.4]

InsertMove = [100212418.6,400095474.0,901323720.8,1598761398.4,2500012331.4]

BubbleTime = [1028.8,4592.2,10370.4,18501.8,29098.0]

BubbleCompare =[199990000.0,799980000.0,1799970000.0,3199960000.0,4999950000.0]

BubbleMove = [300914286.6,1200747451.8,2695793026.8,4799868249.0,7494994849.8]

ShellTime = [3.6,7.4,12.2,16.0,20.6]

ShellCompare =[512549.4,1170807.8,1844605.0,2538958.4,3302836.0]

ShellMove = [772545.4,1730802.8,2684604.0,3738952.4,4802832.0]

HeapTime = [3.4,7.0,11.0,14.6,2.2]

HeapCompare =[473226.8,1026499.2,1608747.2,2212920.8,12030.4]

HeapMove = [253601.4,547350.2,855505.0,1174545.6,105900.2]

MergeTime = [154.8,311.8,479.2,621.4,786.4]

MergeCompare =[260853.2,561747.0,877104.2,1203351.6,1536063.0]

MergeMove = [574464.0,1228928.0,1908928.0,2617856.0,3337856.0]

QuickTime = [2.0,3.8,6.2,8.2,11.4]

QuickCompare =[431237.2,910293.6,1429641.0,1938485.0,2477889.8]

QuickMove = [238263.6,532609.2,846099.0,1178224.2,1523955.0]

plt.subplot(1,3,1)

plt.plot(SelectTime,label='Select',marker='o')

plt.plot(InsertTime,label='Insert',marker='o')

plt.plot(BubbleTime,label='Bubble',marker='o')

plt.plot(ShellTime,label='Shell',marker='o')

plt.plot(HeapTime,label='Heap',marker='o')

plt.plot(MergeTime,label='Merge',marker='o')

plt.plot(QuickTime,label='Quick',marker='o')

plt.title('Time')

plt.legend();

plt.subplot(1,3,2)

plt.plot(SelectCompare,label='Select',marker='o')

plt.plot(InsertCompare,label='Insert',marker='o')

plt.plot(BubbleCompare,label='Bubble',marker='o')

plt.plot(ShellCompare,label='Shell',marker='o')

plt.plot(HeapCompare,label='Heap',marker='o')

plt.plot(MergeCompare,label='Merge',marker='o')

plt.plot(QuickCompare,label='Quick',marker='o')

plt.title('Compare')

plt.legend();

plt.subplot(1,3,3)

plt.plot(SelectMove,label='Select',marker='o')

plt.plot(InsertMove,label='Insert',marker='o')

plt.plot(BubbleMove,label='Bubble',marker='o')

plt.plot(ShellMove,label='Shell',marker='o')

plt.plot(HeapMove,label='Heap',marker='o')

plt.plot(MergeMove,label='Merge',marker='o')

plt.plot(QuickMove,label='Quick',marker='o')

plt.title('Move')

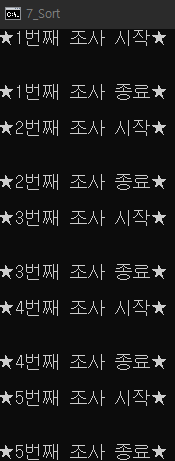
plt.legend();

plt.show()

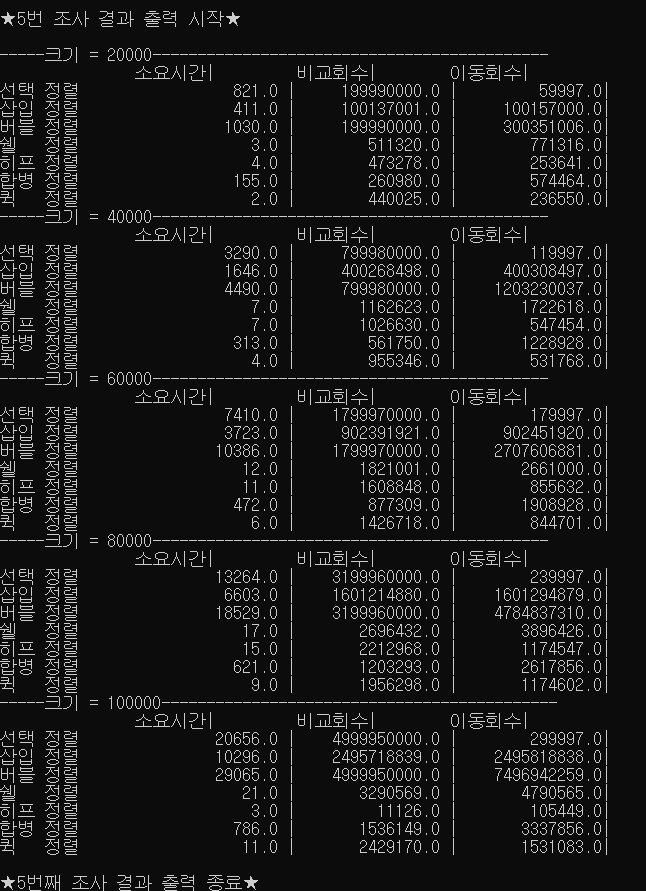
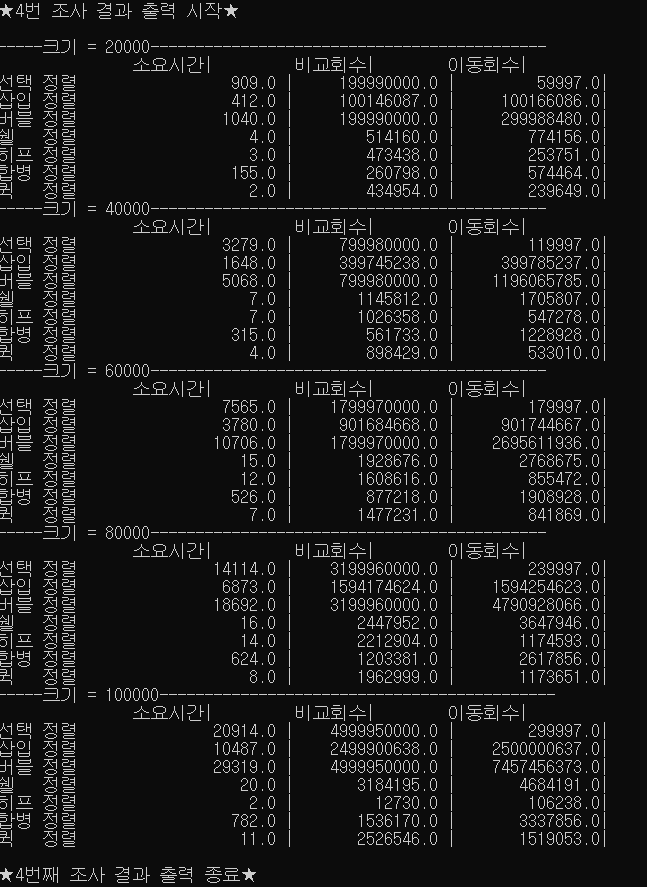
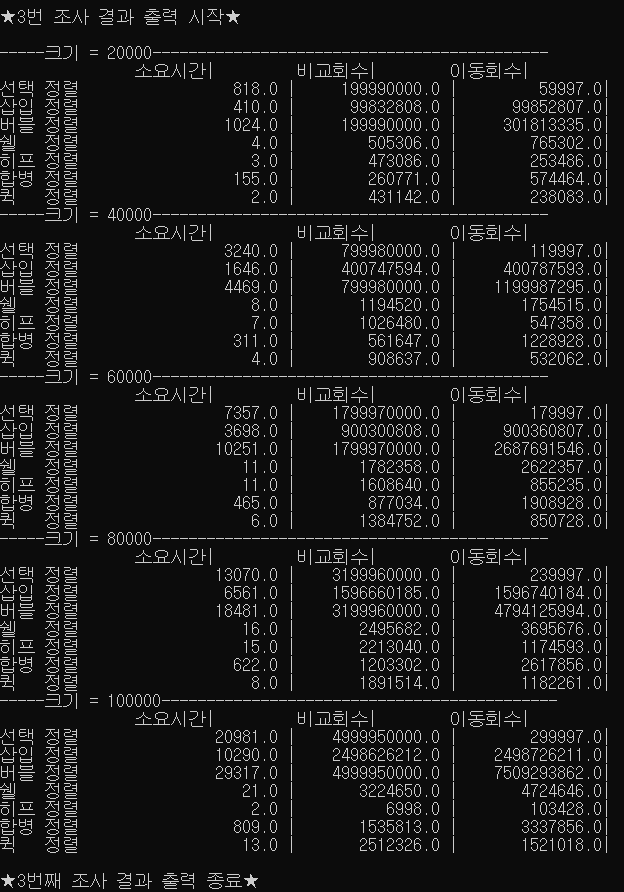
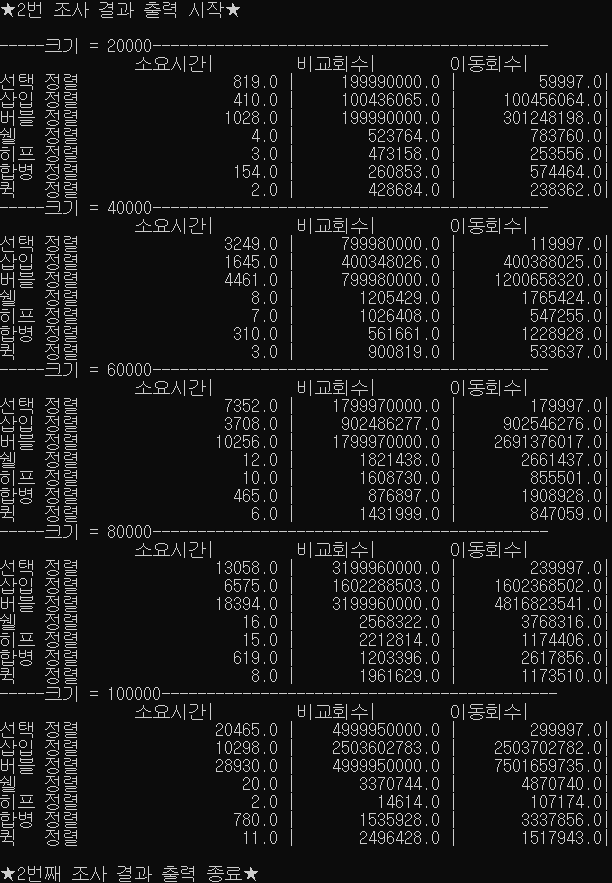
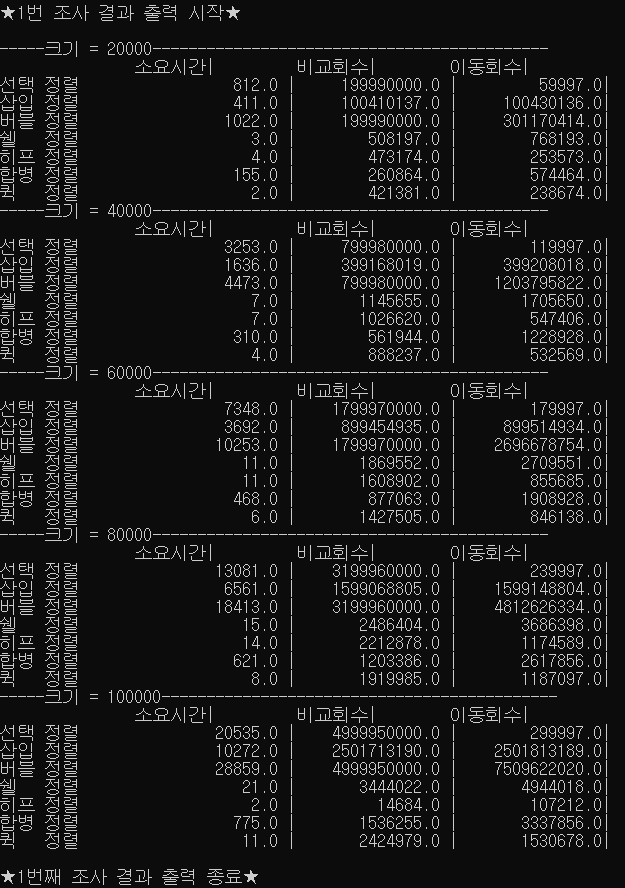
파이썬의 matplotlib를 이용하여 그래프를 출력하였습니다.

해당하는 평균값을 넣은 후 plot함수를 통하여 도표로 표현하였습니다.

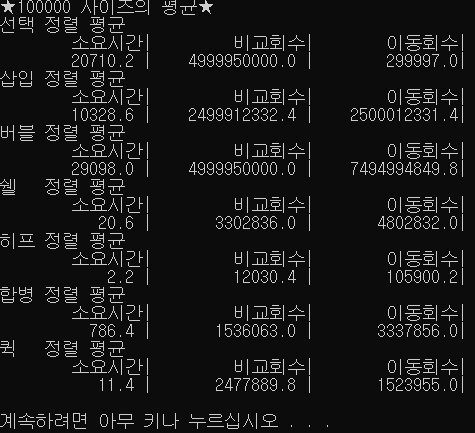
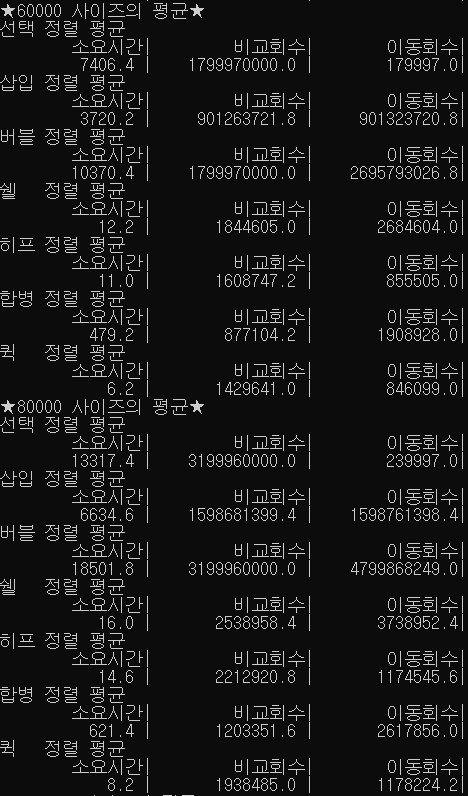
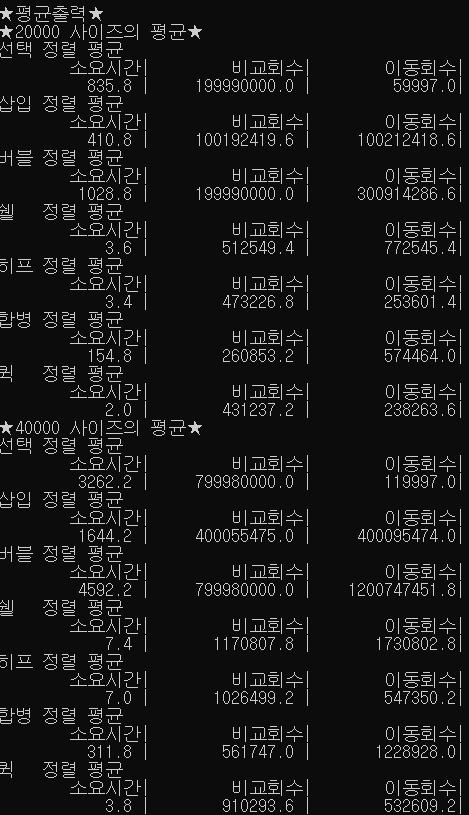
4. 실행결과



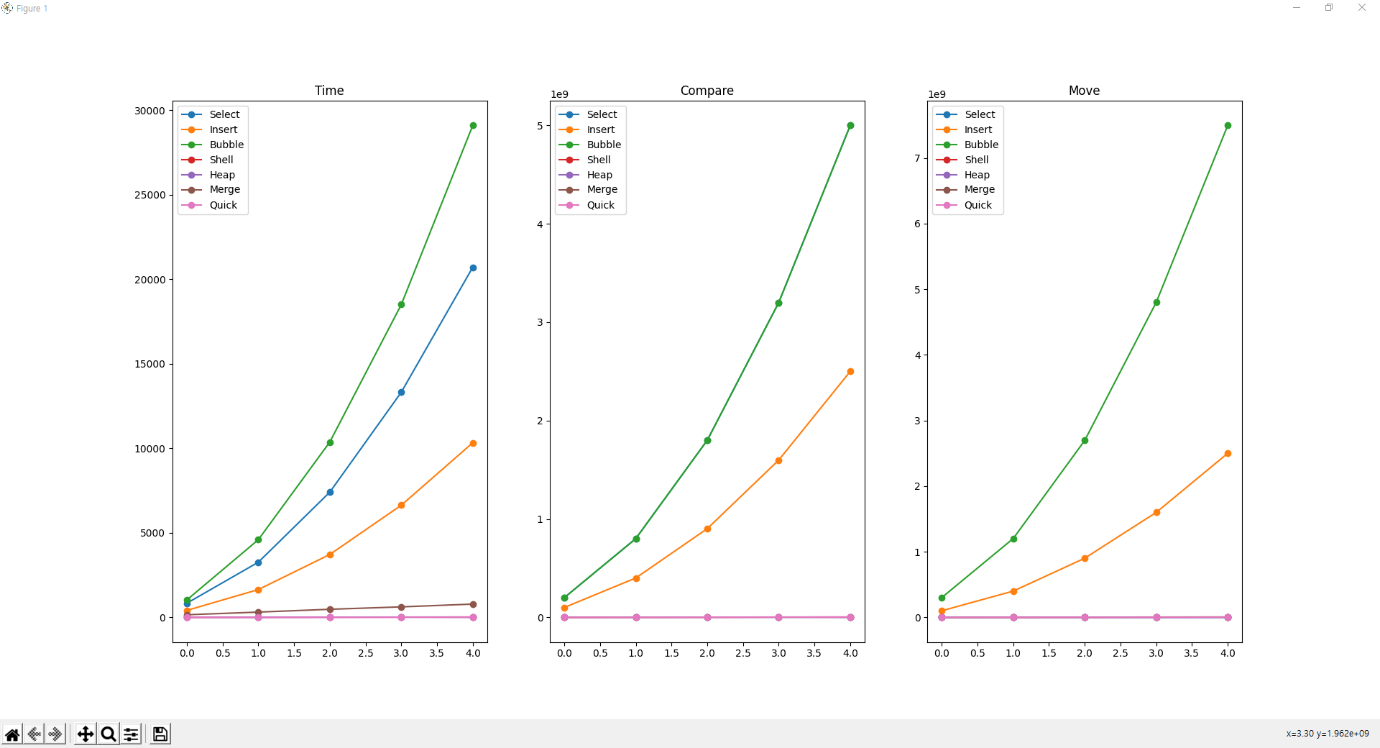
조사 시작 후 조사 종료를 표현하도록 하여 진행상황을 나타내도록 하였습니다.



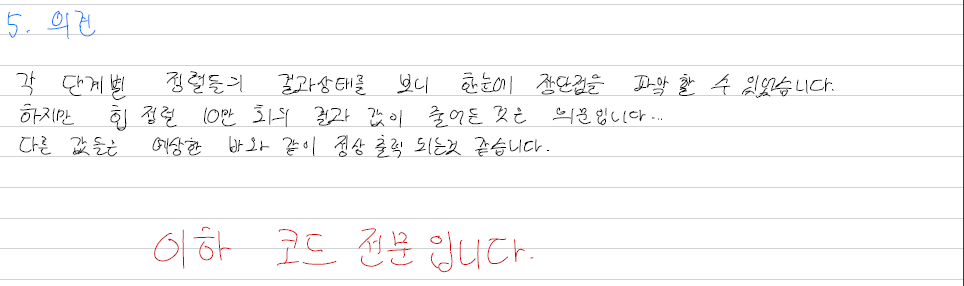
각 단계별 조사 결과입니다.



난수의 개수에 따라 평균을 구하였습니다.



파이썬의 matplotlib을 이용하여 그래프를 출력하였습니다.



#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX 100000 //데이터 항목 최대 개수

struct avg {//데이터를 저장시킬 포맷

    double move[7][7];

    double comp[7][7];

    double clocktime[7][7];

};

struct avg avglist[5];//데이터 저장할 곳

double mvcount;//이동횟수

double cmpcount;//비교횟수

int arr1[MAX], arr2[MAX], arr3[MAX],

    arr4[MAX], arr5[MAX],arr6[MAX],arr7[MAX];//배열확인

int many = 0;

void make\_arr(int size){//크기에 맞는 배열 생성

    int x;

    srand(time(NULL));

    for(int i = 0; i < size; i++) {  //size 크기만큼 난수 생성

        x = rand() % 999 + 1;       //1부터 1000까지의 난수

        //난수 생성 후 저장

        arr1[i] = arr2[i] = arr3[i] = arr4[i] = arr5[i] =

            arr6[i] = arr7[i] = x;

    }

}

void swap(int\* x, int\* y){

    int temp;

    temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void selection\_sort(int \*arr, int size){//선택정렬

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

        int least = i;

        for (int j = i + 1; j < size; j++) {

            cmpcount++;

            if (arr[j] < arr[least]) {

                least = j;

            }

        }

        swap(&arr[least], &arr[i]);

        mvcount = mvcount + 3;  //이동 횟수

    }

}

int insert\_sort(int\* arr, int size){//삽입정렬

    int key;

    for (int i = 1; i < size; i++) {

        int j = i - 1;

        key = arr[i];

        mvcount++;//이동횟수 증가

        cmpcount++;//비교횟수 증가

        while (j >= 0 && arr[j] > key) {

            cmpcount++;

            arr[j + 1] = arr[j];

            mvcount++;

            j = j - 1;

        }

        arr[j + 1] = key;

        mvcount++;

    }

}

void bubble\_sort(int\* arr, int size)

{

    for (int i = size - 1; i > 0; i--)

    {

        for (int j = 0; j < i; j++)

        {

            cmpcount++;//비교횟수 증가

            if (arr[j] > arr[j + 1])

            {

                swap(&arr[j], &arr[j + 1]);

                mvcount = mvcount + 3;//이동횟수 증가

            }

        }

    }

}

int shell\_insert(int\* arr, int first, int last, int gap){//쉘전용 insert

    int key;

    int j;

    for (int i = first + gap; i <= last; i = i + gap) {

        j = i - gap;

        key = arr[i];

        mvcount++;

        cmpcount++;

        while (j >= first && arr[j] > key) {

            cmpcount++;

            arr[j + gap] = arr[j];

            mvcount++;

            j = j - gap;

        }

        arr[j + gap] = key;

        mvcount++;

    }

}

void shell\_sort(int\* arr, int size){//쉘정렬

    int gap;

    for (gap = size / 2; gap > 0; gap = gap / 2) {

        if (gap % 2 == 0)

            gap++;

        for (int i = 0; i < gap; i++) {

            shell\_insert(arr, i, size - 1, gap);

        }

    }

}

//히프정렬

typedef struct heaptype {//히프 정의

    int heap[MAX];

    int heap\_size;

}heaptype;

heaptype\* create()//히프 생성

{

    return (heaptype\*)malloc(sizeof(heaptype));

}

void init(heaptype\* h){//히프 초기화

    h->heap\_size = 0;

}

void insert\_min\_heap(heaptype\* h, int item){//히프 삽입 함수

    int i;

    i = ++(h->heap\_size);

    while ((i != 1) && (item < h->heap[i / 2])) {

        h->heap[i] = h->heap[i / 2];

        i /= 2;

    }

    h->heap[i] = item;

}

int delete\_min\_heap(heaptype\* h){//히프 삭제 함수

    int parent = 1, child = 2;

    int item, temp;

    item = h->heap[1];

    temp = h->heap[(h->heap\_size)--];

    while (child <= h->heap\_size) {

        cmpcount++;

        if ((child < h->heap\_size) && h->heap[child] > h->heap[child + 1]) {

            child++;

        }

        cmpcount++;

        if (temp < h->heap[child]) {

            break;

        }

        h->heap[parent] = h->heap[child];

        mvcount++;

        parent = child;

        child \*= 2;

    }

    h->heap[parent] = temp;

    mvcount++;

    return item;

}

void heap\_sort(int arr[], int size){//히프 정렬

    int i;

    heaptype h;//히프 h를 이용하여 정렬

    init(&h);

    for (int i = 0; i < size; i++) {

        insert\_min\_heap(&h, arr[i]);

    }

    for (int i = 0; i < size; i++) {

        arr[i] = delete\_min\_heap(&h);

    }

}

//합병정렬

void merge(int\* arr, int low, int mid, int high){//병합

    int b1, b2, e1, e2, indx;

    b1 = low;

    e1 = mid;

    b2 = mid + 1;

    e2 = high;

    indx = low;

    int sorted[MAX] = { 0 };

    while (b1 <= e1 && b2 <= e2) {

        cmpcount++;

        if (arr[b1] < arr[b2]) {

            sorted[indx++] = arr[b1++];

            mvcount++;

        }

        else{

            sorted[indx++] = arr[b2++];

            mvcount++;

        }

    }

    if (b1 > e1) {

        for (int i = b2; i <= e2; i++) {

            sorted[indx++] = arr[i];

            mvcount++;

        }

    }

    else if (b2 > e2) {

        for (int i = b1; i <= e1; i++) {

            sorted[indx++] = arr[i];

            mvcount++;

        }

    }

    for (int i = low; i <= high; i++) {

        arr[i] = sorted[i];

        mvcount++;

    }

}

void merge\_sort(int\* arr, int left, int right){//합병 정렬

    int mid;

    if (left < right) {

        mid = (left + right) / 2;

        merge\_sort(arr, left, mid);

        merge\_sort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

//퀵정렬

int partition(int\* arr, int left, int right)

{

    int pivot, low, high;

    low = left, high = right + 1;

    pivot = arr[left];

    do {

        do {

            low++;

            cmpcount++;

        } while (low <= right && arr[low] < pivot);

        do {

            high--;

            cmpcount++;

        } while (high >= left && arr[high] > pivot);

        if (low < high)

        {

            swap(&arr[low], &arr[high]);

            mvcount = mvcount + 3;

        }

        cmpcount++;

    } while (low < high);

    cmpcount++;

    swap(&arr[left], &arr[high]);

    mvcount = mvcount + 3;

    return high;

}

void quick\_sort(int\* arr, int left, int right)

{

    if (left < right)

    {

        int q = partition(arr, left, right);

        quick\_sort(arr, left, q - 1);

        quick\_sort(arr, q + 1, right);

    }

}

int fun(int x, int y, int s){

    clock\_t start, end;

    double elapsed;

    make\_arr(s);

    mvcount = 0, cmpcount = 0;

    start = clock();

    if(y == 0) {//선택 정렬 실행

        selection\_sort(arr1,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 1) {//삽입 정렬 실행

        insert\_sort(arr2,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 2) {//버블 정렬 실행

        bubble\_sort(arr3,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 3) {//쉘 정렬 실행

        shell\_sort(arr4,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 4) {//히프 정렬 실행

        heap\_sort(arr5,s);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 5) {//합병 정렬 실행

        merge\_sort(arr6,0,s-1);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    if(y == 6) {   //퀵  정렬 실행

        quick\_sort(arr7,0,s-1);

        avglist[many].move[x][y] = mvcount;

        avglist[many].comp[x][y] = cmpcount;

    }

    end = clock();

    //실행시간 계산, 저장

    elapsed = (double)((end - start) \* 1000) / CLOCKS\_PER\_SEC;

    avglist[many].clocktime[x][y] = elapsed;

}

void calcStatistics(){//연산 호출

    for(int tryn = 0; tryn < 5; tryn++) {

        printf("★%d번째 조사 시작★\n\n", tryn+1);

        for(int i = 0; i < 5; i++) {

            int rsize = 20000 \* (i+1);

            for(int j = 0; j < 7; j++)

                fun(i, j, rsize);

        }

        many++;

        printf("\n★%d번째 조사 종료★\n\n", tryn+1);

    }

}

void cmpSortByTable(){

    for(int tryn = 0; tryn < 5; tryn++) {

        printf("★%d번 조사 결과 출력 시작★\n\n",tryn+1);

        for(int i = 0; i < 5; i++) {

            int rsize = 20000 \* (i+1);//초기배열사이즈 설정

            printf("-----크기 = %d--------------------------------------------\n",rsize);

            printf("\t       소요시간|         비교회수|        이동회수|\n");

            for(int j = 0; j < 7; j++) {

                switch(j+1) {

                case 1:

                    printf("선택 정렬\t");

                    break;

                case 2:

                    printf("삽입 정렬\t");

                    break;

                case 3:

                    printf("버블 정렬\t");

                    break;

                case 4:

                    printf("쉘   정렬\t");

                    break;

                case 5:

                    printf("히프 정렬\t");

                    break;

                case 6:

                    printf("합병 정렬\t");

                    break;

                case 7:

                    printf("퀵   정렬\t");

                    break;

                }

                printf("%15.1f | %15.1f | %15.1f|\n", avglist[tryn].clocktime[i][j],

                       avglist[tryn].comp[i][j],avglist[tryn].move[i][j]);

            }

        }

        printf("\n★%d번째 조사 결과 출력 종료★\n\n", tryn+1);

    }

    printf("\n★평균출력★\n");

    for(int i = 0; i < 5; i++) {

        printf("★%d 사이즈의 평균★\n",20000\*(i+1));

        for(int j = 0; j < 7; j++) {

            double mavg,cavg,tavg;

            mavg = cavg = tavg = 0;

            for(int k = 0; k < 5; k++) {

                mavg += avglist[k].move[i][j];

                cavg += avglist[k].comp[i][j];

                tavg += avglist[k].clocktime[i][j];

            }

            switch(j+1) {

            case 1:

                printf("선택 정렬 평균\n");

                break;

            case 2:

                printf("삽입 정렬 평균\n");

                break;

            case 3:

                printf("버블 정렬 평균\n");

                break;

            case 4:

                printf("쉘   정렬 평균\n");

                break;

            case 5:

                printf("히프 정렬 평균\n");

                break;

            case 6:

                printf("합병 정렬 평균\n");

                break;

            case 7:

                printf("퀵   정렬 평균\n");

                break;

            }

            printf("        소요시간|         비교회수|        이동회수|\n");

            printf("%15.1f | %15.1f | %15.1f|\n", tavg/5, cavg/5, mavg/5);

        }

    }

}

int main(void){

    calcStatistics();

    cmpSortByTable();

    //cmpSortByGraph(stat)

    //그래프는 직접 그리겠습니다.

}

여기까지 7\_SORT.c 입니다.

이후 파이썬 코드입니다.

from matplotlib import pyplot as plt

SelectTime = [835.8,3262.2,7406.4,13317.4,20710.2]

SelectCompare =[199990000.0,799980000.0,1799970000.0,3199960000.0,4999950000.0]

SelectMove = [59997.0,1199970.0,179997.0,239997.0,299997.0]

InsertTime = [410.8,1644.2,3720.2,6634.6,10328.6]

InsertCompare =[100192419.6,400055475.0,901263721.8,1598681399.4,2499912332.4]

InsertMove = [100212418.6,400095474.0,901323720.8,1598761398.4,2500012331.4]

BubbleTime = [1028.8,4592.2,10370.4,18501.8,29098.0]

BubbleCompare =[199990000.0,799980000.0,1799970000.0,3199960000.0,4999950000.0]

BubbleMove = [300914286.6,1200747451.8,2695793026.8,4799868249.0,7494994849.8]

ShellTime = [3.6,7.4,12.2,16.0,20.6]

ShellCompare =[512549.4,1170807.8,1844605.0,2538958.4,3302836.0]

ShellMove = [772545.4,1730802.8,2684604.0,3738952.4,4802832.0]

HeapTime = [3.4,7.0,11.0,14.6,2.2]

HeapCompare =[473226.8,1026499.2,1608747.2,2212920.8,12030.4]

HeapMove = [253601.4,547350.2,855505.0,1174545.6,105900.2]

MergeTime = [154.8,311.8,479.2,621.4,786.4]

MergeCompare =[260853.2,561747.0,877104.2,1203351.6,1536063.0]

MergeMove = [574464.0,1228928.0,1908928.0,2617856.0,3337856.0]

QuickTime = [2.0,3.8,6.2,8.2,11.4]

QuickCompare =[431237.2,910293.6,1429641.0,1938485.0,2477889.8]

QuickMove = [238263.6,532609.2,846099.0,1178224.2,1523955.0]

plt.subplot(1,3,1)

plt.plot(SelectTime,label='Select',marker='o')

plt.plot(InsertTime,label='Insert',marker='o')

plt.plot(BubbleTime,label='Bubble',marker='o')

plt.plot(ShellTime,label='Shell',marker='o')

plt.plot(HeapTime,label='Heap',marker='o')

plt.plot(MergeTime,label='Merge',marker='o')

plt.plot(QuickTime,label='Quick',marker='o')

plt.title('Time')

plt.legend();

plt.subplot(1,3,2)

plt.plot(SelectCompare,label='Select',marker='o')

plt.plot(InsertCompare,label='Insert',marker='o')

plt.plot(BubbleCompare,label='Bubble',marker='o')

plt.plot(ShellCompare,label='Shell',marker='o')

plt.plot(HeapCompare,label='Heap',marker='o')

plt.plot(MergeCompare,label='Merge',marker='o')

plt.plot(QuickCompare,label='Quick',marker='o')

plt.title('Compare')

plt.legend();

plt.subplot(1,3,3)

plt.plot(SelectMove,label='Select',marker='o')

plt.plot(InsertMove,label='Insert',marker='o')

plt.plot(BubbleMove,label='Bubble',marker='o')

plt.plot(ShellMove,label='Shell',marker='o')

plt.plot(HeapMove,label='Heap',marker='o')

plt.plot(MergeMove,label='Merge',marker='o')

plt.plot(QuickMove,label='Quick',marker='o')

plt.title('Move')

plt.legend();

plt.show()