성능 측정 과제

과목명 | 시스템프로그래밍

담당교수 | 최종무 교수님

학과 | 소프트웨어학과

학번 | 32191197

이름 | 김채은

Ⅰ. 프로젝트 목표

네 번째 과제를 받고 어떤 프로그램을 짜는 게 좋을지 고민하던 중 자료구조 수업에서 퀵 정렬을 배울 때 교수님이 하신 말씀이 떠올랐다. 퀵 정렬을 할 때 배열의 맨 끝에 아주 큰 수를 넣어주는 이유는 성능 향상 때문이라는 내용이었다. 시스템프로그래밍 8장에서 배운 최적화 방법과, 알고리즘 관점에서 성능을 향상시키는 방법을 모두 고려해서 성능 측정 프로그램을 짜보기로 하였다.

Ⅱ. 프로젝트 시행 과정

1. 설계

초기에 설계한 내용은 총 함수 네 개를 가지고 성능 측정을 하는 것이었다. (아직까진 비주얼 스튜디오가 손에 익어서 초기 코드는 비주얼로 작성했다.)

```
int main() {
    int arr1[9] = { 4, 6, 5, 9, 2, 7, 8, 3, 1 };
    int arr2[9] = { 4, 6, 5, 9, 2, 7, 8, 3, 1 };
    int arr3[9] = { 4, 6, 5, 9, 2, 7, 8, 3, 1 };
    int arr4[10] = { 4, 6, 5, 9, 2, 7, 8, 3, 1, MAX };
    int size3 = sizeof(arr3) / sizeof(int);
    int size4 = sizeof(arr4) / sizeof(int);

    printArr(arr1, sizeof(arr1) / sizeof(int));
    quickSort1(arr1, 0, (sizeof(arr1) / sizeof(int)) - 1);
    printArr(arr1, sizeof(arr1) / sizeof(int));

    printArr(arr2, sizeof(arr2) / sizeof(int));
    quickSort2(arr2, 0, (sizeof(arr2) / sizeof(int)) - 1);
    printArr(arr2, sizeof(arr2) / sizeof(int));

    printArr(arr3, size3);
    quickSort3(arr3, 0, size3 - 1);
    printArr(arr3, size3);

    printArr(arr4, size4);
    quickSort4(arr4, 0, size4 - 1);
    printArr(arr4, size4);

    return 0;
}
```

```
void quickSort1(int* a, int left, int right) {

    int t;

    if (left < right) {
        int i = left, j = right+1, pivot = a[left];
        do {
            do { i++; } while (a[i] < pivot && i <= right);
            do { j--; } while (a[j] > pivot);
            if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort1(a, left, j - 1);
        quickSort1(a, j + 1, right);
    }
}
```

```
void quickSort2(int* a, const int left, const int right) {

    int t;

    if (left < right) {
        int i = left, j = right + 1, pivot = a[left];
        do {
            do { i++; } while (a[i] < pivot && i <= right);
            do { j--; } while (a[j] > pivot);
            if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort2(a, left, j - 1);
        quickSort2(a, j + 1, right);
    }
}
```

quickSort1은 함수 인자로 sizeof 함수와 계산식이 들어간다. 그게 퀵 정렬 코드 안에서 사용되면서 딜레이를 발생하는 게, const를 써주면 계속 계산하는 일이 없을까 싶어서 둘을 비교하는 함수를 만들었다.

```
void quickSort3(int* a, const int left, const int right) {

    int t;

    if (left < right) {
        int i = left, j = right + 1, pivot = a[left];
        do {
            do { i++; } while (a[i] < pivot && i <= right);
            do { j--; } while (a[j] > pivot);
            if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort3(a, left, j - 1);
        quickSort3(a, j + 1, right);
    }
}
```

quickSort3는 main에서 size를 미리 계산하고 인자로 넣어줬다.

```
void quickSort4(int* a, const int left, const int right) {

    int t;

    if (left < right) {
        int i = left, j = right + 1, pivot = a[left];
        do {
            do { i++; } while (a[i] < pivot);
            do { j--; } while (a[j] > pivot);
            if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort4(a, left, j - 1);
        quickSort4(a, j + 1, right);
    }
}
```

quickSort4에서는 배열의 맨 끝에 미리 큰 수를 넣어두고 i가 right를 넘도록 증가하지 못하게 했다. i<=right를 판단하는 부분이 사라져서 여기서 큰 성능 향상을 기대했다.

2. 시행착오

(1) 초기 설계 코드

초기 코드로 성능 측정을 했는데 배열 크기가 작아서 측정이 안됐다. 크기를 10으로 했으니 당

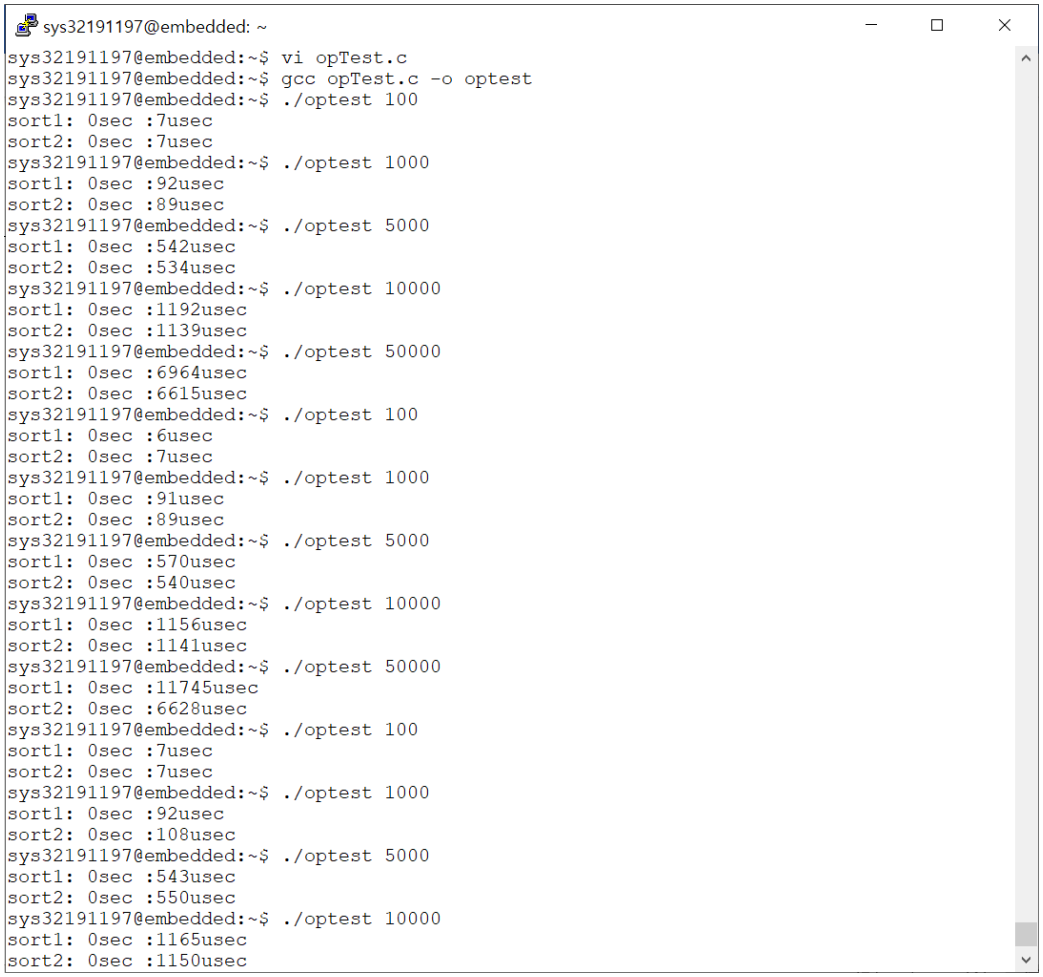
연한 일이었는데 처음엔 감이 아예 없어서 1usec가 나오는 걸 보고 당황했다.

배열 크기를 늘리기 위해 동적 할당과 srand() 함수를 이용해서 랜덤한 숫자 배열을 만들었다. 그리고 똑같이 수행해보니 계속 오류가 떠서, 알아보니 동적 할당은 sizeof() 사용이 안 되기 때문이었다. 구글링을 해보며 _msize, malloc_usable_size() 시도해봤으나 실패했다. 원인을 찾지 못하고 계획했던 quickSort 2, 3을 버리게 됐다.

(2) 두번째 시도

sys32191197@embedded: ~

```
int main(int argc, char *argv[]) {

        int *arr1;
        int *arr2;
        int len = 0;
        int i;
        struct timeval stime, etime, gap;

        if(argc == 2){
                len = atoi(argv[1]);                    // 배열 크기
                arr1 = (int*)malloc(sizeof(int)*len);   // 동적할당
                for(i=0; i<len; i++)                    // 1부터 len까지 배열에 입력
                        arr1[i] = i+1;
                shuffle(arr1, len);                     // 섞어줌
                // printArr(arr1, len);

                gettimeofday(&stime, NULL);
                quickSort1(arr1, 0, len-1);
                gettimeofday(&etime, NULL);

                gap.tv_sec = etime.tv_sec - stime.tv_sec;
                gap.tv_usec = etime.tv_usec - stime.tv_usec;
                if (gap.tv_usec < 0) {
                        gap.tv_sec = gap.tv_sec-1;
                        gap.tv_usec = gap.tv_usec + 1000000;
                }
                // printArr(arr1, len);
                printf("sort1: %ldsec :%ldusec\n", gap.tv_sec, gap.tv_usec);

                arr2 = (int*)malloc(sizeof(int)*(len+1));
                for(i=0; i<len+1; i++)
                        arr2[i] = i+1;
                shuffle(arr2, len+1);
                arr2[len] = MAX;                        // 배열 끝에 아주 큰 값을 넣어준다
                // printArr(arr2, len+1);

                gettimeofday(&stime, NULL);
                quickSort2(arr2, 0, len);
                gettimeofday(&etime, NULL);

                gap.tv_sec = etime.tv_sec - stime.tv_sec;
                gap.tv_usec = etime.tv_usec - stime.tv_usec;
                if (gap.tv_usec < 0) {
                                                                        54,1          15%
```



성능이 들쑥날쑥해서 배열을 두 번 다 랜덤하게 발생시키니 더 순차적으로 섞이냐, 더 랜덤하게 섞이냐에 따라 성능 다르게 나타났다. 그래서 하나의 배열을 랜덤하게 섞고, 그 배열을 나머지 하나에 복사했다.

(3) 세 번째 시도

```
int main(int argc, char *argv[]) {

        int *arr1;
        int *arr2;
        int len = 0;
        int i;
        struct timeval stime, etime, gap;

        if(argc == 2){
                len = atoi(argv[1]);                    // 배열 크기
                arr1 = (int*)malloc(sizeof(int)*len);   // 동적할당
                for(i=0; i<len; i++)                    // 1부터 len까지 배열에 입력
                        arr1[i] = i+1;
                shuffle(arr1, len);                     // 섞어줌
                printArr(arr1, len);

                arr2 = (int*)malloc(sizeof(int)*(len+1));
                for(i=0; i<len+1; i++)
                        arr2[i] = arr1[i];
                arr2[len] = MAX;                        // 배열 끝에 아주 큰 값을 넣어준다
                printArr(arr2, len+1);
                // 제대로 복사됐고, MAX도 제대로 넣어졌는지 확인
```

```
sys32191197@embedded:~$ gcc opTest.c -o optest
sys32191197@embedded:~$ ./optest 10
9 6 2 7 4 3 10 8 1 5
9 6 2 7 4 3 10 8 1 5 99999
sort1: 0sec :1usec
sort2: 0sec :0usec
```

배열이 제대로 복사됐고, 빠른 정렬을 위한 MAX가 제대로 넣어졌는지 확인한 뒤 printArr() 부분은 다시 주석으로 처리했다.

(4) 네번째시도

여기까지 잘 왔다고 생각하고 마무리를 지으려고 했는데, 구상했던 내용들이 아까워서 고민하던중 '굳이 동적할당을 써야하나?'라는 생각이 들었다. 그래서 랜덤 배열을 만들고 복사하는 아이디어는 유지하면서 프로그램을 다시 엎고 초기에 구상했던 형태로 만들었다.

```
int main(int argc, char *argv[]) {

        int len=0;

        if(argc==2){
                len = atoi(argv[1]);
                int arr1[len];
                int arr2[len];
                int arr3[len];
                int arr4[len];
                int arr5[len+1];
                int i;
                struct timeval stime, etime, gap;

                for(i=0; i<len+1; i++)
                        arr1[i] = i+1;

                shuffle(arr1, len);             // 섞어줌
                // printArr(arr1, len);

                for(i=0; i<len+1; i++){
                        arr2[i] = arr1[i];
                        arr3[i] = arr1[i];
                        arr4[i] = arr1[i];
                        arr5[i] = arr1[i];
                }
                arr5[len] = MAX;
                printArr(arr1, len);
                printArr(arr2, len);
                printArr(arr3, len);
                printArr(arr4, len);
                printArr(arr5, len+1);

                // 제대로 복사됐고, MAX도 제대로 넣어졌는지 확인
```

```
sys32191197@embedded:~$ ./optest2 10
6 3 1 7 2 4 5 9 10 8
6 3 1 7 2 4 5 9 10 8
6 3 1 7 2 4 5 9 10 8
6 3 1 7 2 4 5 9 10 8
6 3 1 7 2 4 5 9 10 8 999999
sort1: 0sec :1usec
sort2: 0sec :0usec
sort3: 0sec :1usec
sort4: 0sec :1usec
sort5: 0sec :1usec
```

printArr()은 배열이 잘 복사됐는지 테스트하고 다시 주석 처리했다.

Ⅲ. 결과

1. 소스코드

sys32191197@embedded: ~

```c
/* optimization test2 by. chae-eun. 2020-12-05 */

#include<stdio.h>
#include<unistd.h>
#include<sys/time.h>

#define SWAP(a, b, temp) {temp=a; a=b; b=temp;}
#define MAX 999999

void quickSort1(int* a, int left, int right);
void quickSort2(int* a, const int left, const int right);
void quickSort3(int* a, const int left, const int right);
void shuffle(int *a, int n);
void printArr(int* a, const int n);

int main(int argc, char *argv[]) {

        int len=0;

        if(argc==2){
                len = atoi(argv[1]);
                int arr1[len];
                int arr2[len];
                int arr3[len];
                int arr4[len];
                int arr5[len+1];
                int i;
                struct timeval stime, etime, gap;

                for(i=0; i<len+1; i++)
                        arr1[i] = i+1;

                shuffle(arr1, len);             // 섞 어 줌

                for(i=0; i<len+1; i++){
                        arr2[i] = arr1[i];
                        arr3[i] = arr1[i];
                        arr4[i] = arr1[i];
                        arr5[i] = arr1[i];
                }
                arr5[len] = MAX;
                /*printArr(arr1, len);
                printArr(arr2, len);
```

```
printArr(arr3, len);
printArr(arr4, len);
printArr(arr5, len+1);
*/
// 제대로 복사됐고, MAX도 제대로 넣어졌는지 확인

gettimeofday(&stime, NULL);
quickSort1(arr1, 0, (sizeof(arr1)/sizeof(int))-1);
gettimeofday(&etime, NULL);

gap.tv_sec = etime.tv_sec - stime.tv_sec;
gap.tv_usec = etime.tv_usec - stime.tv_usec;
if (gap.tv_usec < 0) {
        gap.tv_sec = gap.tv_sec-1;
        gap.tv_usec = gap.tv_usec + 1000000;
}
printf("sort1: %ldsec :%ldusec\n", gap.tv_sec, gap.tv_usec);

gettimeofday(&stime, NULL);
quickSort2(arr2, 0, (sizeof(arr1)/sizeof(int))-1);
gettimeofday(&etime, NULL);

gap.tv_sec = etime.tv_sec - stime.tv_sec;
gap.tv_usec = etime.tv_usec - stime.tv_usec;
if (gap.tv_usec < 0) {
        gap.tv_sec = gap.tv_sec-1;
        gap.tv_usec = gap.tv_usec + 1000000;
}
printf("sort2: %ldsec :%ldusec\n", gap.tv_sec, gap.tv_usec);



gettimeofday(&stime, NULL);
quickSort1(arr3, 0, len-1);
gettimeofday(&etime, NULL);

gap.tv_sec = etime.tv_sec - stime.tv_sec;
gap.tv_usec = etime.tv_usec - stime.tv_usec;
if (gap.tv_usec < 0) {
        gap.tv_sec = gap.tv_sec-1;
        gap.tv_usec = gap.tv_usec + 1000000;
}
printf("sort3: %ldsec :%ldusec\n", gap.tv_sec, gap.tv_usec);

gettimeofday(&stime, NULL);
quickSort2(arr4, 0, len-1);
gettimeofday(&etime, NULL);

gap.tv_sec = etime.tv_sec - stime.tv_sec;
gap.tv_usec = etime.tv_usec - stime.tv_usec;
if (gap.tv_usec < 0) {
        gap.tv_sec = gap.tv_sec-1;
        gap.tv_usec = gap.tv_usec + 1000000;
}
printf("sort4: %ldsec :%ldusec\n", gap.tv_sec, gap.tv_usec);


gettimeofday(&stime, NULL);
quickSort3(arr5, 0, len);
gettimeofday(&etime, NULL);

gap.tv_sec = etime.tv_sec - stime.tv_sec;
gap.tv_usec = etime.tv_usec - stime.tv_usec;
if (gap.tv_usec < 0) {
        gap.tv_sec = gap.tv_sec-1;
        gap.tv_usec = gap.tv_usec + 1000000;
}
```

```
                printf("sort5: %ldsec :%ldusec\n", gap.tv_sec, gap.tv_usec);



        }

        return 0;
}
void shuffle(int *a, int n){ // 배 열  랜 덤 하 게  섞 어 준 다

        srand(time(NULL));
        int t, i;
        int rn;

        for(i=0; i<n-1; i++){
                rn = rand()%(n-i)+i;
                SWAP(a[i], a[rn], t);
        }
}
void quickSort1(int* a, int left, int right) {

    int t;

    if (left < right) {
        int i = left, j = right+1, pivot = a[left];
        do {
            do { i++; } while (a[i] < pivot && i <= right);
            do { j--; } while (a[j] > pivot);
            if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort1(a, left, j - 1);
        quickSort1(a, j + 1, right);
    }
}
void quickSort2(int* a, const int left, const int right) {

    int t;

    if (left < right) {
        int i = left, j = right + 1, pivot = a[left];
        do {
                do { i++; } while (a[i] < pivot && i<=right);
                do { j--; } while (a[j] > pivot);
                if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort2(a, left, j - 1);
        quickSort2(a, j + 1, right);
    }
}


void quickSort3(int* a, const int left, const int right) {

    int t;

    if (left < right) {
        int i = left, j = right + 1, pivot = a[left];
        do {
                do { i++; } while (a[i] < pivot);
```

```
    if (left < right) {
        int i = left, j = right + 1, pivot = a[left];
        do {
                do { i++; } while (a[i] < pivot && i<=right);
                do { j--; } while (a[j] > pivot);
                if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort2(a, left, j - 1);
        quickSort2(a, j + 1, right);
    }
}


void quickSort3(int* a, const int left, const int right) {

    int t;

    if (left < right) {
        int i = left, j = right + 1, pivot = a[left];
        do {
                do { i++; } while (a[i] < pivot);
                // 배열 마지막에 99,999가 있기 때문에 i가 right 밖으로 넘어가지
않음
                do { j--; } while (a[j] > pivot);
                if (i < j) SWAP(a[i], a[j], t);
        } while (i < j);
        SWAP(a[left], a[j], t);
        quickSort2(a, left, j - 1);
        quickSort2(a, j + 1, right);
    }
}


void printArr(int* a, const int n) {    // 배열 출력
        int i;
        for (i = 0; i < n; i++){
                        printf("%d ", a[i]);
        }
        printf("\n");
}

                                                                194,0-1       Bot
```

2. 실행화면

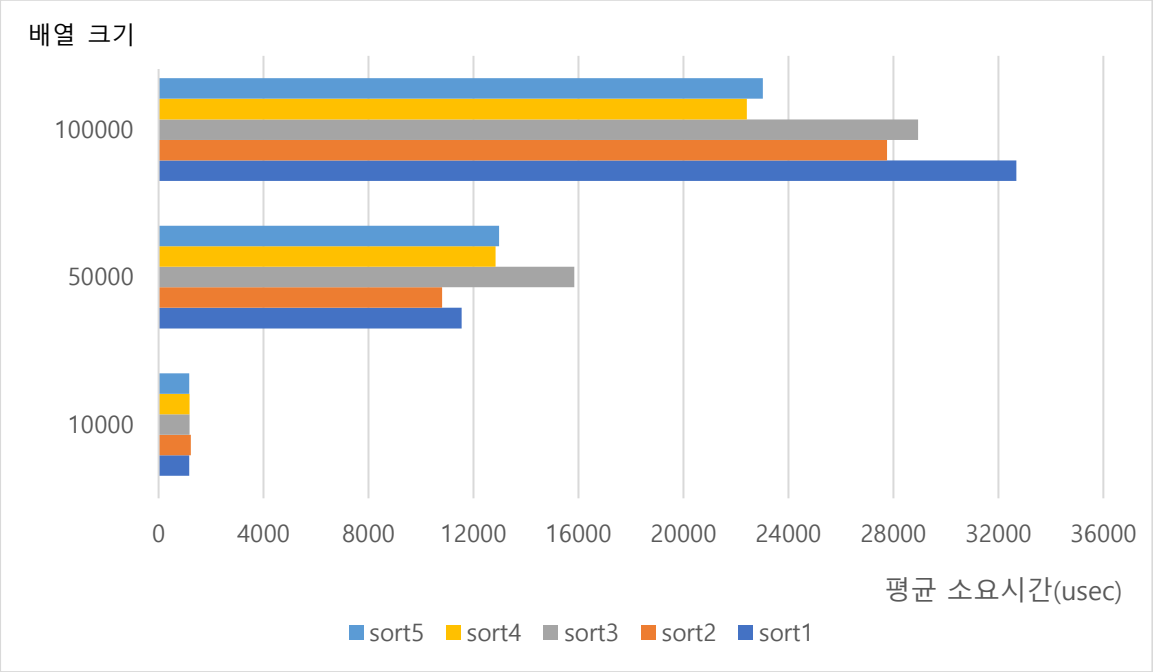
```
sys32191197@embedded: ~
sys32191197@embedded:~$ vi opTest2.c
sys32191197@embedded:~$ gcc opTest2.c -o optest
sys32191197@embedded:~$ ./optest 100
sort1: 0sec :7usec
sort2: 0sec :7usec
sort3: 0sec :7usec
sort4: 0sec :6usec
sort5: 0sec :7usec
sys32191197@embedded:~$ ./optest 1000
sort1: 0sec :91usec
sort2: 0sec :91usec
sort3: 0sec :91usec
sort4: 0sec :111usec
sort5: 0sec :91usec
```

```
sys32191197@embedded:~$ ./optest 10000
sort1: 0sec :1180usec
sort2: 0sec :1349usec
sort3: 0sec :1205usec
sort4: 0sec :1202usec
sort5: 0sec :1190usec
sys32191197@embedded:~$ ./optest 10000
sort1: 0sec :1189usec
sort2: 0sec :1168usec
sort3: 0sec :1174usec
sort4: 0sec :1189usec
sort5: 0sec :1173usec
sys32191197@embedded:~$ ./optest 10000
sort1: 0sec :1152usec
sort2: 0sec :1188usec
sort3: 0sec :1153usec
sort4: 0sec :1158usec
sort5: 0sec :1157usec
sys32191197@embedded:~$ ./optest 50000
sort1: 0sec :21119usec
sort2: 0sec :6876usec
sort3: 0sec :18999usec
sort4: 0sec :19029usec
sort5: 0sec :6828usec
sys32191197@embedded:~$ ./optest 50000
sort1: 0sec :6735usec
sort2: 0sec :10791usec
sort3: 0sec :18840usec
sort4: 0sec :6807usec
sort5: 0sec :19160usec
```

```
sort5: 0sec :6828usec
sys32191197@embedded:~$ ./optest 50000
sort1: 0sec :6735usec
sort2: 0sec :10791usec
sort3: 0sec :18840usec
sort4: 0sec :6807usec
sort5: 0sec :19160usec
sys32191197@embedded:~$ ./optest 50000
sort1: 0sec :11822usec
sort2: 0sec :14769usec
sort3: 0sec :6696usec
sort4: 0sec :18726usec
sort5: 0sec :6752usec
sys32191197@embedded:~$ ./optest 5000
sort1: 0sec :544usec
sort2: 0sec :545usec
sort3: 0sec :544usec
sort4: 0sec :570usec
sort5: 0sec :546usec
sys32191197@embedded:~$ ./optest 100000
sort1: 0sec :42668usec
sort2: 0sec :34289usec
sort3: 0sec :21532usec
sort4: 0sec :30240usec
sort5: 0sec :26291usec
sys32191197@embedded:~$ ./optest 100000
sort1: 0sec :22661usec
sort2: 0sec :26455usec
sort3: 0sec :27032usec
sort4: 0sec :22644usec
sort5: 0sec :20276usec
```

```
sys32191197@embedded:~$ ./optest 100000
sort1: 0sec :32710usec
sort2: 0sec :22542usec
sort3: 0sec :38269usec
sort4: 0sec :14349usec
sort5: 0sec :22500usec
sys32191197@embedded:~$
```

3. 측정 결과



- sort1 – 배열 크기를 sizeof 함수 이용해서 int형 인자에 넣어주었다.
- sort2 – 배열 크기를 sizeof 함수 이용해서 const int형 인자에 넣어주었다.
- sort3 – 배열 크기를 미리 len이라는 변수에 계산해넣고 int형 인자로 넣어주었다.
- sort4 – 배열 크기를 len에 계산해넣고 const int형 인자로 넣어주었다.
- sort5 – 배열 크기를 len에 계산해넣고 const int형 인자로 넣어주고, 정렬 할 때 계산해야할 조건을 하나 빼주었다.

IV. Discussion

배열 크기가 작을 때는 큰 차이가 보이지 않다가 배열 크기를 키우니 경향성이 보이기 시작했다. 우선 const를 이용하면 상수 처리가 되기 때문에 한 번 계산을 하면 더는 안 하지 않을까 하는 마음으로 반신반의하며 실험에 넣었는데 sort1, 2, 3, 4를 비교했을 때 결과가 가장 뚜렷하게 보여서 놀라웠다. 오히려 기대했던 결과는 quickSort3()에서 알고리즘을 바꾸어줬을 때 비교 조건문을 훨씬 덜 체크하기 때문에 크게 성능이 향상할 것이라고 생각했는데, sort4와 다른 점이 없었다. 프로그램이 평면적이지 않고, 항상 내가 예측하고 원하는 대로 흘러가지 않는다는 것을 느끼게 되었다. 수업 시간에 교수님도 이런 면에 대해서 말씀하신 적이 있었는데, 막상 여러 번 시행했을

때마다 결과가 다르게 발생하니 당황스러웠다. 그래도 수업 시간에 배운 내용과는 조금 다른 실험적인 코드를 만들어보고, 직접 테스트해서 이정도 경향성이 나온 것에 의의를 두기로 했다.

처음 과제를 받았을 때, 물론 프로그램을 직접 만들 수도 있지만 시험 기간이 다가오면서 시간도 많이 없고, 동기들한테 물어보니 수업에서 배운 combine으로 과제를 하겠다는 친구들이 많아서 역시 효율적인 면을 선택해야 하나 고민을 했다(물론 combine으로 더 새로운 아이디어를 만들어낼 학우도 있다고 생각한다). 다행인 건 어떤 프로그램을 만들까에 대한 고민 시간이 자료구조 수업을 들으면서 많이 단축되었다는 것이다. 덕분에 퀵 정렬을 다시 제대로 공부할 수 있는 기회도 되었다. 이제와 돌이켜보면 combine을 이용해서 프로젝트를 했다면, 수업의 연장 선상이라는 생각이 들어 다른 과제들처럼 꾸역꾸역했을 것 같다. 2학년 2학기에 들어와서는 갑자기 어려워진 내용에, 실제 실습보다는 이론이 더 많았기 때문에 주어진 문제를 해결하고 프로젝트 만드는 데에서 오는 뿌듯함을 느끼기가 쉽지 않았고 솔직히 지루함의 연속이었다. 이번 과제는 측정 시간이 눈으로 보이고 실행할 때마다 결과가 다르게 나와서, 재밌기도 했고 '내가 지금 뭔가 스스로 작업하고 있다'라는 기분을 느끼게 해주었다.

비록 시간은 오래 걸리고 많이 막혔지만 결국 이렇게 해낼 때 뿌듯함은 정말 큰 것 같다. 매일 이렇게 많이 어렵지 않아서 몇 시간만에 해내고 뿌듯함을 주는 과제만 하면 좋겠다. 사실 얼마 전 컴퓨터구조 수업에서 과제로 cache simulator를 만드는 게 있었는데 trace 확장자나, cache 구조를 처음 접해서 생소하다 보니 이해를 하나도 못하겠고, 그러나보니 스트레스를 많이 받은 상태에서 이번 과제를 시작했다. 자료구조를 공부하면서도 퀵 정렬 부분에서 많이 막혀서 걱정이 앞섰다. 그래도 과제 내용을 이해하고 시작한다는 데에 스스로 위안을 삼으며 아이디어를 정리해 나가기 시작했다. 처음 동적할당 문제에 부딪혔을 때는 계획했던 것을 실패하고 다 버려야 한다는 사실에 절망적이었다. 제대로 해내는 것 없이 마무리돼야 한다는 생각에 답답했지만 결국 스스로 성능을 향상시키는 방법을 생각해보고, 수업 시간에 배운 gettimeofday 함수로 성능을 직접 측정해봤다는 데에 의의를 두고 위안을 삼기로 했다. 그러던 중 돌파구(사실 매우 간단한 문제였지만)를 찾았을 때 얼마나 신이 났는지 모른다. 그렇게 한 번 머리가 열리면 프로그램을 만들고 테스트하고, 오류가 있는 부분을 찾아 하나하나 수정하는 작업이 너무 재밌고 뿌듯해서 시간 가는 줄도 모른다. 저녁쯔음 시작했던 게 결국 새벽 세 시에서야 끝났다. 머리를 CPU처럼 끊임없이 팽팽 돌리다보니 몸은 완전히 방전 상태가 됐지만, 오랜만에 어떤 프로그래밍 작업에 꽂혀서 열정을 쏟아부었더니 번아웃에서 조금 빠져나온 기분이 든다. 지치고 어려워서 다 포기하고 싶다가도 프로그래밍, 그리고 컴퓨터에 대한 공부를 놓지 못하는 이유는 여기에 있다고 생각한다.