[CSE3081(2반)] 알고리즘 설계와 분석 과제 2 보고서

20151623 한상구

1. 5개의 알고리즘에 대한 요약

   A. Algorithm 1, qsort

      기본 stdlib.h 에서 제공하는 함수이다.

   B. Algorithm 21, qsort_orig

      교과서적인 qsort의 구현으로, 가장 왼쪽 element를 pivot으로 잡은 뒤 Recursive하게 구현하였다.

   C. Algorithm 22, qsort_median_insert

      임계값을 설정 한 뒤, 그 값 이하의 Sub-array에 대하여 insertion sort를 수행하게 하는 qsort. 임계값은 20으로 설정하였다.

   D. Algorithm 23, qsort_median_insert_iter

      Algorithm 22와 비슷하나, 큰 부분은 iterative하게, 작은 부분은 recursive하게 정렬하도록 구현하였다. 임계값은 6으로 설정하였다. Algorithm 23부터 pivot값을 잡을 때 John Tukey's median of medians 방법을 이용하기 시작했다.

   E. Algorithm 24, qsort_final

      pivot값을 John Tukey's median of median 방법을 이용하고, Recursive한 부분을 Stack을 구현하여 사용함으로서 함수호출등에 사용되는 시간을 최소화시켰다. Partition 또한 Bentley-Mcllroy 3-way partitioning을 사용하여 최적화해보았다.

2. 시간복잡도

   A. Average Case - $O(nlgn)$

   B. Worst Case - $O(n^2)$

      모든 qsort가 위와 같은 시간복잡도를 가진다. 하지만 Cut-off를 통한 상수의 감소를 통해 보다 나은 수행시간을 보일 수 있다.

3. 소스코드

A. HW2_S20151623.cpp (main)

i. Cmp function

```
int cmp(const void *a, const void *b) { return ((ELEMENT *)a)->score - ((ELEMENT *)b)->score; }
```

A가 담고있는 score가 B가 담고있는 score보다 더 크면

양수를, 같으면 0을, 작으면 음수를 반환하여준다.

ii. Main source

```c
int main()
{
    FILE *fp, *fin, *fout;
    int op, n, i;
    char in[99], out[99];               // for storing .bin files name
    ELEMENT *A;
    void (*QsortPtr[5]) (void *, size_t, size_t, _Cmpfun *) = {
        qsort,                          // From "stdlib.h"
        qsort_orig,
        qsort_median_insert,
        qsort_median_insert_iter,
        qsort_final
    };
    char algo_name[5][30] = {
        "qsort",
        "qsort_orig",
        "qsort_median_insert",
        "qsort_median_insert_iter",
        "qsort_final"
    };
    clock_t before;
    double result[11], sum=0;
    // Declaring variables
    fp = fopen("HW2_commands.txt", "r");
    fscanf(fp, "%d%d%s%s", &op, &n, in, out);
    fclose(fp);
    // Get info from file
    op -= op > 20 ? 20 : 1;             // makes 21, 22, 23, 24 as 1, 2, 3, 4 respectively
    A = (ELEMENT *)malloc(sizeof(ELEMENT) * n);
    // Initialize A ( array ) and op ( given function number )
    for(i = 0; i < 10; i++)
    {
        fin = fopen(in, "rb");
        fread(A, sizeof(ELEMENT), n, fin);
        fclose(fin);
        // Get array's elements
        before = clock();
        QsortPtr[op](A, n, sizeof(ELEMENT), cmp);
        result[i] = (double)(clock() - before)/CLOCKS_PER_SEC;
        // Sort!
        sum += result[i];
    }
    fout = fopen(out, "wb");
    fwrite(A, sizeof(ELEMENT), n, fout);
    fclose(fout);
    // Write elements of sorted array
    free(A);
    // Free which was dynamically allocated
    printf("Input size : %10d\nWith algorithm %d, '%s'\n", n, op?op+20:1, algo_name[op]);
    puts("Sorted 10 times");
    for(i = 0; i < 10; i++) printf("%d : %4.3fms\n", i+1, result[i] * 1000);
    printf("Average : %4.3fms\n", sum * 100);
    return 0;
}
```

B. My_quick_sorts.h (header)

Default

```
#define MAX_STACK_SIZE 4096
#define MAX_BUF_SIZE 512

typedef struct{
    unsigned int score;
    float data[3];
    char comments[16];
} ELEMENT;

typedef struct{
    size_t l, r;
} stack;

typedef int _Cmpfun(const void *, const void *);

void qsort_orig(void *, size_t, size_t, _Cmpfun *);
void qsort_median_insert(void *, size_t, size_t, _Cmpfun *);
void qsort_median_insert_iter(void *, size_t, size_t, _Cmpfun *);
void qsort_final(void *, size_t, size_t, _Cmpfun *);
void insertion_sort(void *, size_t, size_t, _Cmpfun *);
size_t partition(void *, size_t, size_t, _Cmpfun, size_t);
void swap(char *, char *, size_t);
size_t median(size_t, size_t);
size_t median3(size_t, size_t, size_t);
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"my_quick_sorts.h" 24L, 682C
```

모두 my_quick_sorts.cpp에 구현되어있다.

Med3라는 함수가 빠져있는데, median3와 동일하나

3개의 포인터를 받아와 그 값들을 cmp 함수로 비교한 뒤

중간값을 갖는 포인터를 반환해주는 함수이다.

C. My_quick_sorts.cpp

i. Algorithm 21

```
void qsort_orig(void *A, size_t n, size_t size, _Cmpfun *cmp)
{
    char *S = (char *)A;
    size_t i, pivot;
    if( n > 1 )
    {
        pivot = partition(S, n, size, cmp, 0);
        // partition part ends here

        qsort_orig(S, pivot, size, cmp);
        qsort_orig(S+(pivot+1)*size, n-pivot-1, size, cmp);
        // recursion part
    }
    return;
}
```

보는 바와 같이, 주어진 원소가 1개가 되기 전까지

Recursive하게 작동한다.

ii. Algorithm 22

```
void qsort_median_insert(void *A, size_t n, size_t size, _Cmpfun *cmp)
{
    char *S;
    size_t pivot;
    size_t p, fir, sec, thi, len, l, r;
    S = (char *)A;
    if(n < 20)
    {
        insertion_sort(S, n, size, cmp);
        return;
    }
    // Do insertion sort when given array's size is small enough
    pivot = median(0, n);
    // set pivot as median of randomly chosen 3 indices

    pivot = partition(S, n, size, cmp, pivot);
    // partition part ends here

    if(pivot) qsort_median_insert(S, pivot, size, cmp);
    if(n > pivot+1) qsort_median_insert(S+(pivot+1)*size, n-pivot-1, size, cmp);
    // recursion part
    return;
}
```

Algorithm21과 비슷하나, Cut-off를 통한 효과향상을

기대할 수 있다. 임계값은 20으로 설정하였다.

iii. Algorithm 23

```
void qsort_median_insert_iter(void *A, size_t n, size_t size, _Cmpfun *cmp)
{
    char *S = (char *)A;
    char *fir, *sec, *thi, *P, *L, *R;
    size_t pivot, j = n-1, p, len;
    while(j > 0)
    {
        if(j < 7)
        {
            insertion_sort(S, j+1, size, cmp);
            return;
        }
        // Do insertion sort when given array's size is small enough
        if(j > 6)
        {
            L = (char *)A;
            R = (char *)A + j*size;
            len = j+1;
            p = len/8;
            fir = med3(L, L+p*size, L+2*p*size, cmp);
            sec = med3(L+(len/2-p)*size, L+len/2*size, L+(len/2+p)*size, cmp);
            thi = med3(R-2*p*size, R-p*size, R, cmp);
            P = med3(fir, sec, thi, cmp);
            pivot = (size_t)(P-(char *)A)/size;
        }
        pivot = partition(S, j+1, size, cmp, pivot);
        // partition part ends here

        if(pivot > j-pivot)
        {
            if(pivot < j) qsort_median_insert_iter(S+(pivot+1)*size, j-pivot, size, cmp);
            j = pivot-1;
            // When pivot is at right side
        } else
        {
            if(pivot) qsort_median_insert_iter(S, pivot, size, cmp);
            S += (pivot+1)*size;
            j -= pivot+1;
            // When pivot is at middle or left side
        }
    }
    return;
}
```

마지막을 살펴보면 작은 부분에 대하여 Recursive를,

큰 부분은 iterative하게 진행하는 것을 볼 수 있다.

중간 pivot을 설정하는 부분에서, 총 9부분의 원소에

대해 중간값을 설정함을 볼 수 있다.

(John Tukey's median of medians)

```
void qsort_final(void *A, size_t n, size_t size, _Cmpfun *cmp)
{
    char *S = (char *)A;
    char *t = (char *)malloc(sizeof(char) * size);
    char *fir, *sec, *thi, *pivot, *L, *R;
    size_t top = 0, l, r, len, p;
    size_t I, J, P, Q, k;
    stack s[MAX_STACK_SIZE];
    s[++top].l = 0; s[top].r = n-1;
    while(top)
    {
        l = s[top].l; r = s[top--].r;
        if(l >= r) continue;
        len = r-l+1;
        if(len <= 8)
        {
            insertion_sort(S+l*size, len, size, cmp);
            continue;
        }
        L = (char *)A + l*size;
        R = (char *)A + r*size;
        p = len/8;
        fir = med3(L, L+p*size, L+2*p*size, cmp);
        sec = med3(L+(len/2-p)*size, L+len/2*size, L+(len/2+p)*size, cmp);
        thi = med3(R-2*p*size, R-p*size, R, cmp);
        pivot = med3(fir, sec, thi, cmp);
        swap(pivot, S+l*size, size);
        I = l; J = r+1; P = l; Q = r+1;
        while(1)
        {
            memcpy(t, S+l*size, size);
            while((*cmp)(S+(++I)*size, t) < 0)
                if(I == r)
                    break;
            while((*cmp)(t, S+(--J)*size) < 0)
                if(J == l)
                    break;
            if(I >= J)
                break;
            swap(S+I*size, S+J*size, size);
            if((*cmp)(t, S+I*size) == 0)
                swap(S+(++P)*size, S+I*size, size);
            if((*cmp)(t, S+J*size) == 0)
                swap(S+(--Q)*size, S+J*size, size);
        }
        swap(S+l*size, S+J*size, size);

        I = J+1; J = J-1;
        for(k = l+1; k <= P; ++k)
            swap(S+k*size, S+(J--)*size, size);
        for(k = r; k >= Q; --k)
            swap(S+k*size, S+(I++)*size, size);
        s[++top].l = l; s[top].r = J;
        s[++top].l = I; s[top].r = r;
    }
    free(t);
    return;
}
```

iv. Algorithm 24

Bentley-McIlroy 3-way partitioning를 적용했음을 볼

수 있다. 또한 Stack을 이용하여 Recursive에서 소요되는

시간을 최소화 시켰음을 확인할 수 있다.

v. 기타 median, median3, insertion, partition, swap function

```
char* med3(char *a, char *b, char *c, _Cmpfun *cmp)
{
    return cmp(a, b) < 0 ?
        (cmp(b, c) < 0 ? b : (cmp(a, c) < 0 ? c : a))
        :(cmp(b, c) > 0 ? b : (cmp(a, c) < 0 ? a : c));
}

size_t median3(size_t a, size_t b, size_t c)
{ return a>b?a>c?b>c?b:c:a:a<c?b<c?b:c:a; }

size_t median(size_t j, size_t k)
{
    size_t a, b, c;
    a = rand() % (k-j) + j;
    b = rand() % (k-j) + j;
    c = rand() % (k-j) + j;
    return median3(a, b, c);
}
```

```
size_t partition(void *A, size_t n, size_t size, _Cmpfun *cmp, size_t pivot)
{
    char *S = (char *)A;
    char *p = (char *)malloc(sizeof(char) * size);
    size_t i=0, k=0;
    memcpy(p, S+pivot*size, size);
    swap(S+pivot*size, S+(n-1)*size, size);
    for(i = 0; i < (n-1)*size; i+=size)
    {
        if((*cmp)(S+i, p) <= 0)
        {
            swap(S+k*size, S+i, size);
            k++;
        }
    }
    swap(S+(n-1)*size, S+k*size, size);
    free(p);
    return k;
}
```

```
void swap(char *x, char *y, size_t size)
{
    char buf[MAX_BUF_SIZE];
    size_t m, ms;
    for(ms = size; ms > 0; ms-=m, x+=m, y+=m)
    {
        m = (ms < sizeof(buf)) ? ms : sizeof(buf);
        memcpy(buf, x, m);
        memcpy(x, y, m);
        memcpy(y, buf, m);
    }
}
```

```
void insertion_sort(void *A, size_t n, size_t size, _Cmpfun *cmp)
{
    size_t i, j;
    char *S = (char *)A;
    char *T = (char *)malloc(sizeof(char) * size);
    i = n-1;
    while(i-- > 0)
    {
        memcpy(T, S+i*size, size);
        j = i;
        while(++j < n && (*cmp)(T, S+j*size) > 0);
        if(--j == i) continue;
        memcpy(S+i*size, S+(i+1)*size, size*(j-i));
        memcpy(S+j*size, T, size);
    }
    free(T);
    return;
}
```

4. 실행시간

A. Qsort

```
139:2 uppo97$ !.
./sort
Input size :   10000000
With algorithm 1, 'qsort'
Sorted 10 times
1 : 2348.743ms
2 : 2372.273ms
3 : 2357.604ms
4 : 2390.632ms
5 : 2409.653ms
6 : 2383.181ms
7 : 2344.594ms
8 : 2383.954ms
9 : 2354.117ms
10 : 2377.231ms
Average : 2372.198ms
```

B. Qsort_orig

```
139:2 uppo97$ !.
./sort
Input size :   10000000
With algorithm 21, 'qsort_orig'
Sorted 10 times
1 : 6210.643ms
2 : 6302.373ms
3 : 6492.363ms
4 : 6590.689ms
5 : 6879.163ms
6 : 6784.678ms
7 : 6647.913ms
8 : 6788.152ms
9 : 6810.368ms
10 : 6786.061ms
Average : 6629.240ms
```

C. Qsort_median_insert

```
139:2 uppo97$ !.
./sort
Input size :   10000000
With algorithm 22, 'qsort_median_insert'
Sorted 10 times
1 : 6275.265ms
2 : 5723.331ms
3 : 5814.078ms
4 : 5923.849ms
5 : 5721.903ms
6 : 5949.888ms
7 : 5629.897ms
8 : 5941.099ms
9 : 5980.275ms
10 : 6026.193ms
Average : 5898.578ms
```

D. Qsort_median_insert_iter

```
139:2 uppo97$ !.
./sort
Input size :   10000000
With algorithm 23, 'qsort_median_insert_iter'
Sorted 10 times
1 : 5683.907ms
2 : 5959.497ms
3 : 5789.738ms
4 : 5648.383ms
5 : 5861.158ms
6 : 5829.185ms
7 : 6034.334ms
8 : 7022.931ms
9 : 6850.903ms
10 : 6604.973ms
Average : 6128.501ms
```

E. Qsort_final

```
139:2 uppo97$ !.
./sort
Input size :   10000000
With algorithm 24, 'qsort_final'
Sorted 10 times
1 : 4232.488ms
2 : 4363.646ms
3 : 4342.952ms
4 : 4394.923ms
5 : 4238.540ms
6 : 4414.380ms
7 : 4426.388ms
8 : 4536.589ms
9 : 4311.413ms
10 : 4315.112ms
Average : 4357.643ms
```

5. 실험 환경

A. OS : OS X El Capitan Ver. 10.11.6

B. CPU: 1.7 GHz Intel Core i7

C. Ram: 8.00GB

D. Compiler: gcc compiler Ver. 4.2.1 with –o optimization option