자료구조 10주차 과제

제출일: 2022-11-07
학번/이름: 2016121150 / 윤준영

1. 다음은 n개의 정수를 가진 배열 a를 받아 분포수 세기 정렬을 수행하는 함수이다. 물음에 답하시오.

```
void dist_count(int a[], int n) {
     int i;
     int *b, *count;
     b = (int *)malloc(sizeof(int)*n);
     count = (int *)malloc(sizeof(int)*10);
     for (i = 0; i < 10; i++) // 1
          count[i] = 0;
     for (i = 0; i < n; i++) // 2
          count[a[i]]++;
     for (i = 1; i < 10; i++) // 3
          count[i] = count[i]+count[i-1];
     for (i = 0; i < n; i++) // 4
          b[--count[a[i]]] = a[i];
     for (i=0; i<n; i++)  a[i] = b[i]; // 5
}
```

1) 위 함수가 정상적으로 작동하기 위한 배열 a에 저장된 키값들의 범위는 어떻게 되는지 정확히 쓰시오.(1점)

분포수를 세는 행렬인 count의 행렬은 0~9의 인덱스를 가지는 행렬이며, 각 원소는 a행렬에서의 해당 원소의 개수를 나타낸다. 위 함수에서 a에 저장된 키값이 **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**를 가져야 정상적으로 작동한다.

2) 주어진 배열 원소의 개수 n에 대한 위 함수의 worst case asymptotic time complexity는 ? (1점)

T(n) = C + T1(n) + T2(n) + T3(n) + T4(n) + T5(n)
= C + C1 + n + 10 + n + C5
= C' + 2n ~ **O(n)**

각 반복문 1, 2, 3, 4, 5에 대해 시간 복잡도는 위와 같이 계산할 수 있다. 또한 어떠한 상황에서도 계산횟수는 같으므로 모든 case에 대해 점근적 시간 복잡도는 **O(n)**이다.

3) 위 정렬 함수는 stable하지 않다. 한 줄의 코드를 고쳐, 위 정렬 함수가 stable하도록 만드시오. (이렇게 수정하면 stable 한 이유를 설명하시오.) (1점)

4번째 for문의 반복 조건을 다음과 같이 바꾼다.

for (i = n-1; i >= 0; i--) b[--count[a[i]]] = a[i];

원래의 함수를 사용하였을 때에는 누적분포로 저장되어 있는 count배열에서 현재 원소가 들어가야 할 곳을 찾고, 해당하는 count배열의 값을 감소시킨다. 이렇게 정렬하면 같은 키값을 가지는 원소가 들어가야 할 부분의 마지막 부분부터 채워지게 되어 역순으로 정렬되어 stable하

지 않다. 따라서 4번째 for문의 반복 조건을 n−1에서 0으로 역순으로 감소시키게 되면 같은 키값을 가지는 원소들이 다시 역순으로 정렬되어 stable하게 정렬된다.

2. 다음 기수 교환 정렬 함수이다.

```
unsigned bits(unsigned x, int k, int j) {
    return (x >> k) & ~(~0 << j);
}
void radix_exchange_sort(char a[], int n, int b) {
    int i, j;
    char t;
    if (n > 1  &&  b >= 0) {
        i= 0;
        j = n-1;
        while (i!= j) {
            while (bits(a[i], b, 1) == 0 && i< j) i++;
            while (bits(a[j], b, 1) != 0 && i< j) j--;
            t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
        if (bits(a[n-1], b, 1) == 0) {
                j++;
                printf("%c %d\n",a[n-1], n-1);
        }
        radix_exchange_sort(a, j, b-1);
        radix_exchange_sort(a+j, n-j, b-1);
    }
}
```

1) main 함수를 작성하여, 배열 a[] = {‘t’, ‘h, ’i’, ‘s’, ‘p’, ‘r’, ‘o’, ‘b’, ‘l’, ‘e’, ‘m’}을 정렬하시오.(2점)

```
34  int main() {
35      char a[] = {'t','h','i','s','p','r','o','b','l','e','m'};
36      radix_exchange_sort(a, 11, 7);
37      for (int i=0; i<11; i++)
38          printf("%c ", a[i]);
39      printf("\n");
40  }
```

위와 같이 main 함수를 작성하였다. radix_exchange_sort에서 입력해줘야 할 인자 n은 배열의 크기 11, 인자 b는 각 문자의 이진수의 자리수가 7개이므로 7을 넣어주었다.
결과는 다음과 같이 알파벳 순서로 잘 정렬된 것을 확인할 수 있다.

```
m 10
h 1
t 3
b e h i l m o p r s t
* 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

2) 1의 정렬 시에 밑줄의 printf문에 의하여 다음이 출력된다. 그 원인을 설명하시오.(2점)

```
m 10
h 1
t 3
```

if (bits(a[n-1], b, 1) == 0) 부분에서 해당 조건이 만족한다는 뜻은 함수에서 행렬 a의 비교하려는 bit가 모두 0으로 이루어졌을 때이다. a[n-1]은 마지막 원소를 의미하며 bits함수에 의해 비교하려는 bit가 0일 때 true가 된다. 행렬 a는 위 while문에서 해당 bit에 대해 이미 정렬이 이루어져 있으므로 마지막 원소의 해당 bit가 0이라는 뜻은 모든 해당 bit가 0이라는 뜻이다.

해당 bit가 모두 0인 경우는 3가지가 있으며 다음과 같다.

```
[thisproblem](00000000000)      n=11, b=7              print m 10
[thisproblem](11111111111)      n=11, b=6
[thisproblem](11111111111)      n=11, b=5
[thisproblem](10011100000)      n=11, b=4
    [mhielbo](1110101)          n=7, b=3
        [be](01)                n=2, b=2
            [b](1)              n=1, b=1    sorted b
            [e](0)              n=1, b=1    sorted e
        [ihlmo](00111)          n=5, b=2
            [ih](00)            n=2, b=1               print h 1
                [h](0)          n=1, b=0    sorted h
                [i](1)          n=1, b=0    sorted i
            [lmo](001)          n=3, b=1
                [lm](01)        n=2, b=0
                    [l]         n=1, b=-1   sorted l
                    [m]         n=1, b=-1   sorted m
                [o](1)          n=1, b=-1   sorted o
    [rpst](0000)                n=4, b=3               print t 3
        [rpst](0001)            n=4, b=2
            [rps](101)          n=3, b=1
                [p](0)          n=1, b=0    sorted p
                [rs](01)        n=2, b=0
                    [r]         n=1, b=-1   sorted r
                    [s]         n=1, b=-1   sorted s
            [t](0)              n=1, b=1    sorted t
>>  b e h i l m o p r s t
```

문3) 다음 함수를 일반적인 자료형을 위한 쉘 정렬 함수로 고쳐 작성하시오. (3점)

```
#define BASE(i)     ((char*)base + (i)*width)
typedef int (*FCMP)(const void*, const void*);

void shell_sort(int a[], int n) {
    int i, j, k, h, v;
    for (h = n/2; h > 0; h /= 2) {
          for (i= 0; i< h; i++) {
              for (j = i+h; j < n; j += h) {
                 v = a[j];
                 k = j;
```

```
                while (k > h-1 && a[k-h] > v) {
                   a[k] = a[k-h];
                   k -= h;
                }
                a[k] = v;
             }
          }
    }
}
```

위 함수를 다음과 같이 고쳐 작성하였다.

```
5   #define BASE(i)     ((char*)base + (i)*width)
6   typedef int (*FCMP)(const void*, const void*);
7
8
9   void shell_sort(void *base, size_t n, size_t width,
10                  int (*fcmp)(const void*, const void*)){
11      void *v = malloc(width);
12      int i, j, k, h;
13      for (h = n/2; h > 0; h /= 2) {
14         for (i= 0; i< h; i++) {
15            for (j = i+h; j < n; j += h) {
16               memcpy(v, BASE(j), width);
17               k = j;
18               while (k > h-1 && fcmp(BASE(k-h), v)>0) {
19                  memcpy(BASE(k), BASE(k-h), width);
20                  k -= h;
21               }
22               memcpy(BASE(k), v, width);
23  }}}}
24
25  int int_cmp(const void *a, const void *b){
26      /* compare float */
27      return (*(int *)a - *(int *)b);
28  }
29
30  int double_cmp(const void *a, const void *b){
31      /* compare float */
32      return (*(double *)a - *(double *)b + 1);
33  }
```

함수 포인터를 사용하여 원하는 자료형을 비교하는 함수를 사용하면 일반적인 자료형의 쉘 정렬을 할 수 있다. 임의로 사용할 변수는 malloc을 사용하여 동적 할당하였으며, 대입이나 복사를 하는 부분은 memcpy를 이용하여 자료형의 크기만큼의 메모리를 복사하였다.

다음 main 함수를 이용하여 쉘 정렬을 실행해 본 결과는 다음과 같다.

```c
int main() {
    int a[] = {9, 8, 7, 5, 1, 6, 7, 8, 6, 5, 4, 3, 1, 1, 0};
    shell_sort(a, 15, sizeof(int), int_cmp);
    for (int i=0; i<15; i++)
        printf("%d ", a[i]);
    printf("\n");

    double b[] = {4.2, 3.4, 5.6, 1.2, 3.3, 7.7};
    shell_sort(b, 6, sizeof(double), double_cmp);
    for (int i=0; i<6; i++)
        printf("%lf ", b[i]);
    printf("\n");
}
```

```
0 1 1 1 3 4 5 5 6 6 7 7 8 8 9
1.200000 3.300000 3.400000 4.200000 5.600000 7.700000
* 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

int 자료형과 double 자료형 모두 쉘 정렬이 잘 실행된 것을 확인할 수 있다.