자료구조 11주차 과제

제출일: 2022-11-14
학번/이름: 2016121150 / 윤준영

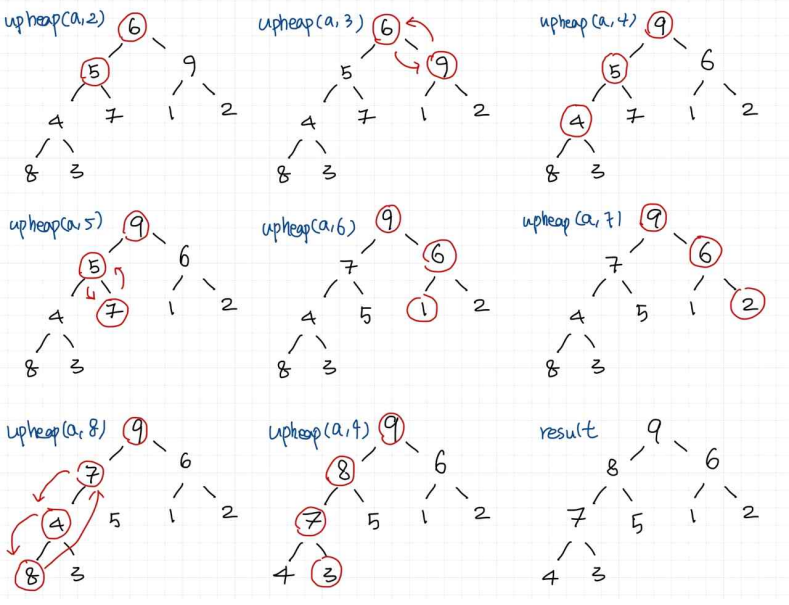
문1) 힙 정렬에 대하여 다음 물음에 답하시오.

```
#include <stdio.h>
void upheap(int a[], int k) {
    int v = a[k];
    while (a[k/2] <= v) {
        a[k] = a[k/2];
        k /= 2;
    }
    a[k] = v;
}
void downheap(int a[], int n, int k)
{
    int i, v;
    v = a[k];
    while (k <= n/2) {
        i = k<<1;
        if (i < n && a[i] < a[i+1])
           i++;
        if (v >= a[i]) break;
        a[k] = a[i];
        k = i;
    }
    a[k] = v;
}
```

```
void heap_sort(int a[], int n) {
    int i;
    for (i= 2; i<= n; i++) {
        upheap(a,i);
    }
    for (i= n; i> 1; i--) {
        int v = a[1];
        a[1] = a[i];
        a[i] = v;
        downheap(a, i-1, 1);
    }
}

int main(int argc, char* argv[]) {
        int a[] =
             {999,6,5,9,4,7,1,2,8,3};
        heap_sort(a, 9);
}
```

1) a가 배열을 사용한 이진나무의 표현법을 사용하여 트리를 표현하고 있다고 할 경우에, a가 나타내는 트리가 힙이 되어가는 과정을 밑줄 친 upheap을 수행할 때 마다의 트리 모양으로 그리시오. (원소 999는 보초값이므로 무시함) (1점)



6 5 9 4 7 1 2 8 3 < original
6 5 9 4 7 1 2 8 3 < upheap(a,2)
9 5 6 4 7 1 2 8 3 < upheap(a,3)
9 5 6 4 7 1 2 8 3 < upheap(a,4)
9 7 6 4 5 1 2 8 3 < upheap(a,5)
9 7 6 4 5 1 2 8 3 < upheap(a,6)
9 7 6 4 5 1 2 8 3 < upheap(a,7)
9 8 6 7 5 1 2 4 3 < upheap(a,8)
9 8 6 7 5 1 2 4 3 < upheap(a,9)

2) down 함수에서 while 문의 조건에 k<=n/2이 사용된 이유는 무엇인가? (1점)
downheap을 호출하기 전 루트 노드와 가장 끝의 원소를 바꾸게 된다. 원래 루트 노드는 가장 큰 원소가 들어가 있어 바꾼 후에는 가장 큰 원소가 가장 끝으로 들어가게 된다. k<=n/2 조건을 통해 가장 끝의 노드의 조상 노드부터 정렬을 수행하게 한다.

3) 퀵정렬과 비교한 힙정렬의 장점은 무엇인가? 병합 정렬과 비교한 힙정렬의 장단점은 무엇인가? (1점)
퀵정렬보다 worst case에서의 성능이 좋으며, 퀵정렬은 재귀호출을 사용하는데 비해 힙정렬은 추가적인 메모리가 필요하지 않아 메모리에 있어 이득이다.
힙정렬은 안정성이 없지만 병합 정렬은 안정성이 있다. 병합 정렬은 정렬하는 과정에서 추가적인 메모리가 필요하다.

문2) 다음은 병합 정렬 함수이다. 이에 대하여 물음에 답하시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

int min(int x, int y) {
       if (x<y)
              return x;
       else
              return y;
}

void merge(int a[], int b[], int c[], int na, int nb) {
       int i=0, j=0, k=0;
       while (k < na+nb)
          if (a[i] <= b[j])
               if (i< na) c[k++] = a[i++];
               else   c[k++] = b[j++];
          else
               if (j < nb) c[k++] = b[j++];
               else c[k++] = a[i++];
}

void merge_sort(int a[], int n) {
    int i, j, k, first, second, size;
    int *b = (int*)malloc(sizeof(int)*n);
    for (size = 1; size < n; size <<= 1)       {
        first = 0; second = first + size;
        while (first < n) {
            merge(a+first, a+second, b+first,
                 size, min(size,n-second));
            first = second + size; second = first + size;
          }
        for (i= 0; i< n; i++)
            a[i] = b[i];   /* memcpy(a,b,sizeof(int)*n); */
    }
    free(b);
}
```

1) a = {1,3,5,7}, b={2,4,6,8}일 경우 다음 호출 후에 c에 저장되는 값들을 저장 순서대로 쓰시오. (1점)

1-1) merge(a,b,c,4,2) c = {1, 2, 3, 4, 5, 7}

1-2) merge(a,b,c,4,-1) c = {1, 3, 5}

2) merge_sort(a, 18)을 수행한다고 가정하자.(2점)

(na, nb) = (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1),
(2, 2), (2, 2), (2, 2), (2, 2), (2, 0),
(4, 4), (4, 4), (4, −2),
(8, 8), (8, −6),
(16, 2)

2−1) merge 함수 호출 시 전달되는 size 값들을 모두 쓰시오.

size = 1(9회), 2(5회), 4(3회), 8(2회), 16(1회)

2−2) merge 함수 호출 시 size의 값이 4일 때 전달되는 정수 인자(**min(size,n-second)**) 값의 종류를 모두 쓰시오.

(4, 4), (4, 4), (4, −2)
4(2회), −2(1회)

문3) 다음 외부 정렬 프로그램에 대하여 물음에 답하시오. (Visual Studio 컴파일 권장) (4점)

```
/*  EXTSORT.C  :  External Sort Alogorithm Test                    */
```

1) 위 프로그램을 실행하면 중간파일을 생성하고 사용 후 삭제한다. 중간 파일이 남아 있도록 (삭제되지 않도록) 프로그램을 수정하시오.

external_sort 함수에서 unlink(tmp[i].fname) 부분이 중간파일을 삭제한다. unlink 명령을 없애주면 중간 파일이 남아있게 된다.

HW > HW11 > C hw11_3.c > external_sort(FILE *, FILE *,

```
111     for (i = 0; i < nfile; i++) {
112         fclose(tmp[i].fp);
113         //unlink(tmp[i].fname);
114         free(tmp[i].v);
115     }
116     free(min);
117     free(tmp);
118 }
```

RANDOM.DAT	2022-11-14 오후 8:58	DAT 파일	12KB
RESULT.DAT	2022-11-14 오후 8:58	DAT 파일	12KB
0.TMP	2022-11-14 오후 8:58	TMP 파일	4KB
1.TMP	2022-11-14 오후 8:58	TMP 파일	4KB
2.TMP	2022-11-14 오후 8:58	TMP 파일	4KB

TMP 파일이 남아있는 것을 확인할 수 있다.

2) 위 프로그램을 수행하면 정렬된 정수 파일 RESULT.DAT가 만들어진다. 정렬된 결과의 원소들을 출력하도록 프로그램을 수정하시오.

print_dat 함수를 작성하여 출력하였다.

```
void print_dat(FILE* dst, size_t nelem, size_t width, FCMP fcmp) {
    void* v = malloc(width);
    void* z = calloc(1, width);
    printf("\n Result : ");
    for (int i = 0; i < nelem; i++) {
        fread(v, width, 1, dst);
        printf("%d ", fcmp(v, z));
    }
}
```

HW > HW11 > C hw11_3.c > external_sort(FILE *, FILE *, si

```
126     free(min);
127     free(tmp);
128     rewind(dst);
129     print_dat(dst, nelem, width, fcmp);
130 }
```

print_dat 함수는 external_sort 마지막에 추가하여 RESULT.DAT의 파일 포인터, 원소의 개수, 원소의 크기와 비교하는 함수 포인터를 받는다.
파일 포인터로부터 파일을 원소의 크기만큼 읽어 v에 저장한 후, v와 0이 저장되어 있는 z를 비교하여 출력한다. 비교하는 함수는 뺄셈으로 이루어져 있으므로 v의 값을 반환하게 된다. 이를 원소의 개수만큼 반복한다.

```
Reading & Sorting & Writing 0.TMP.
Now merging...
Result : 107 1151 2043 2132 2175 2681 2839 3131 3223 3297 3348 3569 3714 3931 4425 4837 4887 5617 5679 5750 6232 7115 7130 7435 7483
 7549 7602 7673 7982 8142 8673 8756 9334 10300 10409 10892 11267 11639 11936 12356 12450 12872 12876 13243 13646 13744 14178 14402 14
980 14997 15317 15424 15741 16454 17616 17637 17796 18809 19184 19302 19353 19736 20040 20298 20306 21068 21465 21602 21656 21965 221
41 22377 22670 22834 23425 25594 25612 25636 25923 25966 26349 26596 26678 26756 28060 28235 28599 28817 28965 29705 29725 29782 3009
5 30498 30633 30739 31298 31667 31876 32118 * 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

결과가 출력되는 것을 확인할 수 있다.

3) 위 프로그램에서 정렬하는 원소들의 개수, 생성되는 중간 임시파일들의 개수를 출력하도록 프로그램을 수정하시오.
external_sort 함수에 다음 명령을 추가하여 원소들의 개수와 임시파일들의 개수를 출력하였다.

HW > HW11 > C hw11_3.c > external_sort(FILE *, FILE *, size_t, void *, size_t, FCMP)

```
65
66      fseek(src, 0L, SEEK_END);
67      nelem = (int)(ftell(src) / width);
68      nbuf = buflen / width;
69      nfile = nelem / nbuf + 1;
70      printf("\nnumber of element: %d\nnumber of tmp files: %d", nelem, nfile);
```

다음과 같이 원소들의 개수와 임시파일들의 개수를 확인할 수 있다.

```
number of element: 3000
number of tmp files: 3
 Reading & Sorting & Writing 0.TMP.
 Reading & Sorting & Writing 1.TMP.
 Reading & Sorting & Writing 2.TMP.
 Now merging...
 Result : 2 14 29 51 54 56 57 61 65
```

4) 위 프로그램은 안정성을 가지고 있는가? 안정성을 가지고 있다면 그 근거를 제시하시오. 안정성을 가지고 있지 않다면 안정성을 갖도록 프로그램을 수정하시오.
위 프로그램은 쉘정렬을 사용하기 때문에 안정성을 가지고 있지 않다. 안정성을 가지기 위해 쉘정렬을 문2)의 병합 정렬을 모든 자료형에 대해 활용 가능하게 바꾸어 프로그램을 수정하였다.

HW > HW11 > C hw11_3_stable.c > ...

```
30
31  int min(int x, int y) {
32      if (x < y) return x;
33      else return y;
34  }
35
36  void merge(void* a, void* b, void* c, int na, int nb, size_t width, FCMP fcmp) {
37      int i = 0, j = 0, k = 0;
38      while (k < na+nb)
39          if (fcmp((char*)a + i*width, (char*)b + j*width)<= 0)
40              if (i < na) memcpy((char*)c + (k++)*width, (char*)a + (i++)*width, width);
41              else memcpy((char*)c + (k++)*width, (char*)b + (j++)*width, width);
42          else
43              if (j < nb) memcpy((char*)c + (k++)*width, (char*)b + (j++)*width, width);
44              else memcpy((char*)c + (k++)*width, (char*)a + (i++)*width, width);
45  }
```

```c
void merge_sort(void* a, size_t n, size_t w, FCMP fcmp) {
    int i, j, k, p, s, size;
    int *b = malloc(w*n);
    for (size = 1; size < n; size <<= 1) {
        p = 0; s = p + size;
        while (p < n) {
            merge((char*)a + p*w, (char*)a + s*w, (char*)b + p*w, size, min(size, n-s), w, fcmp);
            p = s + size; s = p + size;
        }
        for (i = 0; i < n; i++) memcpy(a, b, sizeof(int)*n);
    }
    free(b);
}
```

위에 따라 external_sort의 함수 호출 영역도 수정하였다.

HW > HW11 > C hw11_3_stable.c > external_sort(FILE *, FILE *, size_t, void *, size_t, FCMP)

```c
            int ne;
            fread(buf, width, nbuf, src);
            merge_sort(buf, nbuf, width, fcmp);
            ne = fwrite(buf, width, nbuf, tmp[i].fp);

            printf("\n Reading & Sorting & Writing %s.", tmp[i].fname);
        }
        fread(buf, width, nelem % nbuf, src);
        merge_sort(buf, nelem % nbuf, width, fcmp);
        fwrite(buf, width, nelem % nbuf, tmp[i].fp);
```

실행 결과는 다음과 같다.

```
number of element: 3000
number of tmp files: 3
 Reading & Sorting & Writing 0.TMP.
 Reading & Sorting & Writing 1.TMP.
 Reading & Sorting & Writing 2.TMP.
 Now merging...
 Result : 0 11 30 31 47 75 75 77 93 100 100 103 106 141 143 143 154 154 155 175 187 200 219 231 246 257 257 268 274 279 282 291 296 3
13 339 344 364 373 376 416 417 427 437 459 460 462 484 501 503 518 536 542 551 580 601 603 605 620 653 654 675 682 695 720 728 749 75
1 774 790 791 809 809 810 829 903 914 929 945 969 979 984 993 996 1053 1054 1068 1078 1105 1109 1115 1120 1153 1153 1169 1187 1197 12
07 1209 1212 1213 1218 1234 1238 1238 1251 1255 1273 1288 1308 1312 1316 1327 1341 1363 1366 1380 1381 1382 1384 1441 1468 1469 1483
```

…

…

```
 31371 31388 31416 31427 31428 31433 31434 31440 31462 31474 31475 31477 31484 31487 31497 31499 31501 31503 31512 31525 31539 31543
31565 31571 31573 31579 31597 31612 31645 31655 31656 31657 31690 31707 31710 31715 31718 31718 31721 31724 31756 31803 31810 31812 3
1828 31832 31835 31837 31845 31847 31851 31851 31861 31889 31902 31920 31946 31967 31970 31973 31986 31987 31992 31993 32002 32002 32
040 32059 32069 32076 32078 32088 32090 32107 32109 32126 32130 32167 32169 32188 32190 32195 32199 32231 32240 32261 32276 32294 322
98 32308 32308 32342 32355 32376 32391 32398 32417 32420 32428 32434 32450 32469 32509 32511 32512 32526 32549 32569 32596 32610 3264
9 32656 32665 32668 32669 32671 32671 32680 32691 32701 32706 32710 32721 32733 32736 32757 32763 32767 * 터미널이 작업에서 다시 사
용됩니다. 닫으려면 아무 키나 누르세요.
```