# 자료구조응용

# 20. Sorting: merge sort, heap sort

1. 입력파일(input.txt)로부터 key를 읽어 반복을 통한 합병정렬(iterative merge sort)을 수행하고자 한다. mergeSort(Program 7.9)의 while 문에서 각 mergePass 호출 후의 배열 a와 extra의 상태를 단계적으로 나타내 보라. 초기 입력 데이터는 배열 a[1:n] 에 있다.

**<실행순서>**

① 입력파일(input.txt)로부터 key를 읽어 들여 구조체 배열 a에 저장한다.

※ element 타입은 key 필드만으로 구성된 구조체를 재정의한 것으로 가정한다.

| input.txt |
|---|
| 11 |
| 12 2 16 30 8 28 4 10 20 6 18 |

※ 첫 줄의 11은 입력키의 개수

② 각 레코드의 key에 대해 반복을 통한 합병정렬을 실행한다. 이때, mergeSort 함수를 수정하여 mergePass 수행마다 **세그먼트 크기(s), 배열 a와 extra 상태**를 화면에 출력하라.

③ 최종 정렬결과를 화면에 출력한다.

**④ 아래 코드는 참조만 할 것**

```
void merge(element initList[], element mergedList[],
           int i, int m, int n)
{/*  the sorted lists initList[i:m] and initList[m+1:n] are
     merged to obtain the sorted list mergedList[i:n] */
   int j,k,t;
   j = m+1;        /* index for the second sublist */
   k = i;          /* index for the merged list */

   while (i <= m && j <= n) {
      if (initList[i].key <= initList[j].key)
         mergedList[k++] = initList[i++];
      else
         mergedList[k++] = initList[j++];
   }
   if (i > m)
   /* mergedList[k:n] = initList[j:n] */
      for (t = j; t <= n; t++)
         mergedList[t] = initList[t];
   else
   /* mergedList[k:n] = initList[i:m] */
      for (t = i; t <= m; t++)
         mergedList[k+t-i] = initList[t];
}
```

**Program 7.7:** Merging two sorted lists

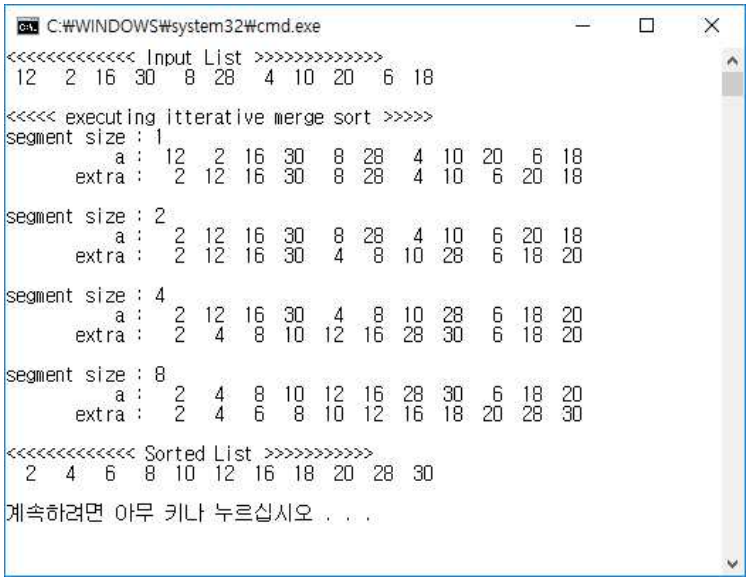
```
void mergePass(element initList[], element mergedList[],
               int n, int s)
{/* perform one pass of the merge sort, merge adjacent
    pairs of sorted segments from initList[] into mergedList[],
    n is the number of elements in the list, s is
    the size of each sorted segment */
  int i,j;
  for (i = 1; i <= n - 2 * s + 1; i += 2 * s)
    merge(initList,mergedList,i,i + s - 1,i + 2 * s - 1);
 if (i + s - 1 < n)
    merge(initList,mergedList,i,i + s - 1,n);
 else
    for (j = i; j <= n; j++)
      mergedList[j] = initList[j];
}
```

**Program 7.8:** A merge pass

```
void mergeSort(element a[], int n)
{/* sort a[1:n] using the merge sort method */
   int s = 1; /* current segment size */
   element extra[MAX_SIZE];

   while (s < n) {
      mergePass(a, extra, n, s);
      s *= 2;
      mergePass(extra, a, n, s);
      s *= 2;
   }
}
```

**Program 7.9:** Merge sort

<실행예>


C:₩WINDOWS₩system32₩cmd.exe
<<<<<<<<<<<<< Input List >>>>>>>>>>>>>
12 2 16 30 8 28 4 10 20 6 18
<<<<< executing itterative merge sort >>>>>
segment size : 1
a : 12 2 16 30 8 28 4 10 20 6 18
extra : 2 12 16 30 8 28 4 10 6 20 18
segment size : 2
a : 2 12 16 30 8 28 4 10 6 20 18
extra : 2 12 16 30 4 8 10 28 6 18 20
segment size : 4
a : 2 12 16 30 4 8 10 28 6 18 20
extra : 2 4 8 10 12 16 28 30 6 18 20
segment size : 8
a : 2 4 8 10 12 16 28 30 6 18 20
extra : 2 4 6 8 10 12 16 18 20 28 30
<<<<<<<<<<<< Sorted List >>>>>>>>>>>
2 4 6 8 10 12 16 18 20 28 30
계속하려면 아무 키나 누르십시오 . . .

2. 입력파일(input.txt)로부터 key를 읽어 **재귀적인 합병정렬(recursive merge sort)을 수행하고자 한다. 서브리스트로 분할된 후 합병되는 과정을 실행의 예 처럼 출력 하시오.**

**<실행순서>**

① 입력파일(input.txt)로부터 key를 읽어 들여 정수형 배열 a에 저장한다.

| input.txt |
| --- |
| 13 |
| 8 3 13 6 2 14 5 9 10 1 7 12 4 |

※ 첫 줄의 13은 입력키의 개수

② 각 레코드의 key에 대해 재귀적인 합병정렬을 실행한다.

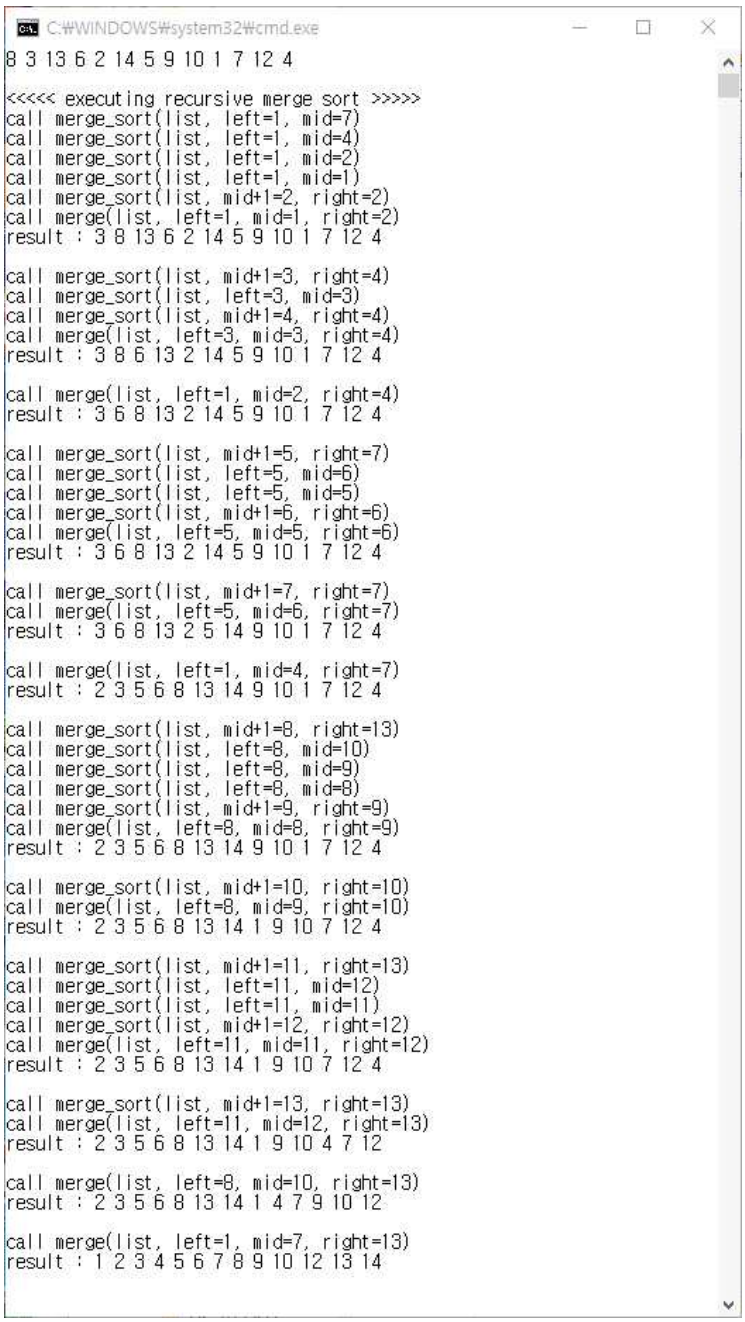
③ 재귀적호출과 합병하는 과정을 출력하라.

**④ 아래 코드는 참조만 할 것**

```
int sorted[MAX_SIZE];    // additional space

/* i left index of sorted list
    j right index of sorted list
    k index  */
void merge(int list[], int left, int mid, int right)
{
int i, j, k, l;
i=left;  j=mid+1;  k=left;

/* merge of sorted lists */
while(i<=mid && j<=right) {
          if(list[i]<=list[j])
           sorted[k++] = list[i++];
          else
           sorted[k++] = list[j++];
}
if(i>mid)/* copy of remained elemnets */
          for(l=j; l<=right; l++)
           sorted[k++] = list[l];
else/* copy of remained elemnets */
          for(l=i; l<=mid; l++)
          sorted[k++] = list[l];
/* copy  sorted[] to  list[] */
for(l=left; l<=right; l++)
          list[l] = sorted[l];
}

void merge_sort(int list[], int left, int right)
{
int mid;
if(left<right) {
          mid = (left+right)/2;
          merge_sort(list, left, mid);      /* sort partioned lists */
          merge_sort(list, mid+1, right); /* sort partioned lists */
          merge(list, left, mid, right);      /*merge */
          }
}
```

< 실행예 >


input - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
13
8 3 13 6 2 14 5 9 10 1 7 12 4


C:₩WINDOWS₩system32₩cmd.exe
8 3 13 6 2 14 5 9 10 1 7 12 4
<<<<< executing recursive merge sort >>>>>
call merge_sort(list, left=1, mid=7)
call merge_sort(list, left=1, mid=4)
call merge_sort(list, left=1, mid=2)
call merge_sort(list, left=1, mid=1)
call merge_sort(list, mid+1=2, right=2)
call merge(list, left=1, mid=1, right=2)
result : 3 8 13 6 2 14 5 9 10 1 7 12 4
call merge_sort(list, mid+1=3, right=4)
call merge_sort(list, left=3, mid=3)
call merge_sort(list, mid+1=4, right=4)
call merge(list, left=3, mid=3, right=4)
result : 3 8 6 13 2 14 5 9 10 1 7 12 4
call merge(list, left=1, mid=2, right=4)
result : 3 6 8 13 2 14 5 9 10 1 7 12 4
call merge_sort(list, mid+1=5, right=7)
call merge_sort(list, left=5, mid=6)
call merge_sort(list, left=5, mid=5)
call merge_sort(list, mid+1=6, right=6)
call merge(list, left=5, mid=5, right=6)
result : 3 6 8 13 2 14 5 9 10 1 7 12 4
call merge_sort(list, mid+1=7, right=7)
call merge(list, left=5, mid=6, right=7)
result : 3 6 8 13 2 5 14 9 10 1 7 12 4
call merge(list, left=1, mid=4, right=7)
result : 2 3 5 6 8 13 14 9 10 1 7 12 4
call merge_sort(list, mid+1=8, right=13)
call merge_sort(list, left=8, mid=10)
call merge_sort(list, left=8, mid=9)
call merge_sort(list, left=8, mid=8)
call merge_sort(list, mid+1=9, right=9)
call merge(list, left=8, mid=8, right=9)
result : 2 3 5 6 8 13 14 9 10 1 7 12 4
call merge_sort(list, mid+1=10, right=10)
call merge(list, left=8, mid=9, right=10)
result : 2 3 5 6 8 13 14 1 9 10 7 12 4
call merge_sort(list, mid+1=11, right=13)
call merge_sort(list, left=11, mid=12)
call merge_sort(list, left=11, mid=11)
call merge_sort(list, mid+1=12, right=12)
call merge(list, left=11, mid=11, right=12)
result : 2 3 5 6 8 13 14 1 9 10 7 12 4
call merge_sort(list, mid+1=13, right=13)
call merge(list, left=11, mid=12, right=13)
result : 2 3 5 6 8 13 14 1 9 10 4 7 12
call merge(list, left=8, mid=10, right=13)
result : 2 3 5 6 8 13 14 1 4 7 9 10 12
call merge(list, left=1, mid=7, right=13)
result : 1 2 3 4 5 6 7 8 9 10 12 13 14

3. 입력파일(input.txt)로부터 key를 읽어 힙정렬(heap sort)을 수행하고자 한다.
입력 리스트에 대하여 MAX HEAP을 구축 후 최대값를 출력 할 때마다  변화하는 Heap을 출력 하시오

**<실행순서>**

① 입력파일(input.txt)로부터 key를 읽어 들여 구조체 배열 a에 저장한다.

※ element 타입은 key 필드만으로 구성된 구조체를 재정의한 것으로 가정한다.

| input.txt |
| --- |
| 10 |
| 26 5 77 1 61 11 59 15 48 19 |

※ 첫 줄의 10은 입력키의 개수

② 입력 리스트에 대하여 MAX HEAP을 구축 후 최대값를 출력 할 때마다  변화하는 Heap을 출력한다.

③ 정렬결과를 파일(output.txt)에 저장한다.

**④ 아래 코드는 참조만 할 것**

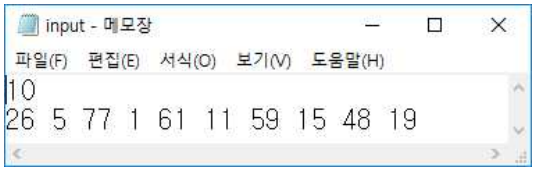
```
void heapSort(element a[], int n)
{/* perform a heap sort on a[1:n] */
   int i,j;
   element temp;

   for (i = n/2; i > 0; i--)
      adjust(a,i,n);
   for (i = n-1; i > 0; i--) {
      SWAP(a[1],a[i+1],temp);
      adjust(a,1,i);
   }
}
```

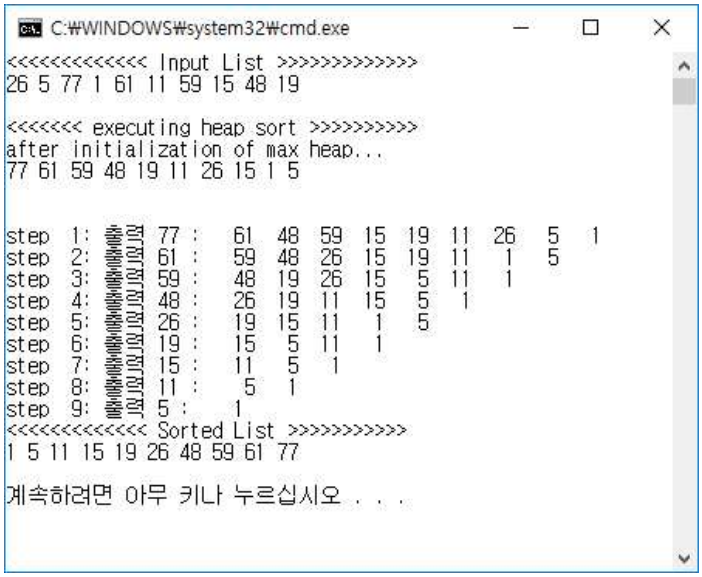
**Program 7.13:** Heap sort

```
void adjust(element a[], int root, int n)
{/* adjust the binary tree to establish the heap */
   int child,rootkey;
   element temp;
   temp = a[root];
   rootkey = a[root].key;
   child = 2 * root;                    /* left child */
   while (child <= n) {
      if ((child < n) &&
      (a[child].key < a[child+1].key))
         child++;
      if (rootkey > a[child].key) /* compare root and
                                     max. child */
         break;
      else {
         a[child / 2] = a[child]; /* move to parent */
         child *= 2;
      }
   }
   a[child/2] = temp;
}
```

**Program 7.12:** Adjusting a max heap

**<실행 예>**




■ 제출 형식

- 공학인증 시스템(ABEEK)에 과제를 올릴 때 제목:
- 1차 제출: 학번 이름 DS-20(1), 2차 제출: 학번 이름 DS-20(2)
- 솔루션 이름 : DS-20
- 프로젝트 이름 : 1, 2, 3
- 실행화면을 캡쳐하여 한글파일에 추가 후 솔루션 폴더에 포함.
- 한글 파일명 : 학번_이름_실습결과.hwp
- 솔루션 폴더를 압축하여 게시판에 제출할 것.
- 압축 파일 명: 학번_이름_DS-20.zip
- 제출은 2회걸쳐 가능(수정 시간 기준으로 처리)