CHAP7 SORT

1. 개요

. 객체의 집합 - list, file

. Single Object 의 정보 - record, field

(ex. list 가 전화 번호부라면, 이름, 주소, 등의 field 로 구성된 record 정의)

. record 의 list 탐색 - record 식별 위한 특정 field 로 탐색 => key (전화번호부의 경우, 이름 key, 전화번호 key 등)

. Search 의 효율 -> record 의 배열순서에 따라 달라짐

- Sorting
  1) Internal Sorting (메모리에서 처리가능)
  2) External Sorting (Data 양이 많음, 보조기억장치 필요)

  ■ Ascending order, Descending order

- **Internal Sorting**

. Insertion Sort — (삽입)

. Selection Sort
. Bubble Sort
. Quick Sort
(교환)

. Heap Sort — (선택)

comparative sort
(key 를 비교하여 순서정함)

. Merge Sort — (병합)

. Radix Sort — Distributive sort

■ Which sort to select (컴퓨터의 특성, Data file 의 크기, 키 값의 분포, 공간 및 시간등을 고려해서 선택함)

1) HEAP sort (O(nlogn) - worst, average case)

- HEAP is a special kind of binary tree
  (complete BT, each node’s data > it’s children’s)

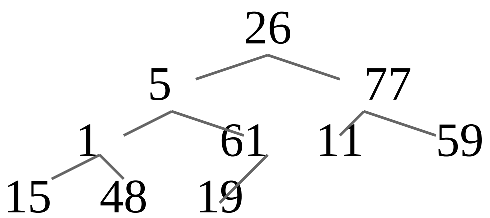
- Heap sort 의 두 단계
  1) 화일 표현하는 tree 를 max HEAP 변환
  2) ROOT 출력하고 나머지 tree 를 다시 HEAP 으로 만드는 process 계속 (**Heapify**)

```
Void HEAPSORT (list[], int n)
{
   for (I = n/2; I>0; I--) {
        adjust(list, I, n);        //heap 변환
    for (I=n-1; I>0; I--) {
        swap(list[1], list[I+1], temp);    //root 출력
        adjust(list, 1, I);                    //heap 변환
     }
  }
```

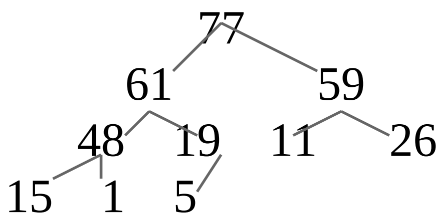
*** ADJUST (first FOR LOOP adjust => make MAX HEAP)**

ex) 26, 5, 77, 1, 61, 11, 59, 15, 48, 19

1) interpret as BT



2) Convert into HEAP



⇨ to convert into MAX HEAP follow the (adjust algorithm) in the textbook”

** Adjust main code

```
for (i=n/2;    i>0;    i--)
     adjust(Heap,i,n);

adjust(Heap, i, n)    {
   int child,    j;
   child = 2*i;
   temp = Heap[i];

   while (n >= child) {
          if (n>child && Heap[child] < Heap[child+1])
             child =child+1;    //right child  선택
          if(temp >= Heap[child]) break;
           j=child/2;
           Heap[j] = Heap[child];
           child=2*child;
   }

   j=child/2;
   Heap[j]=temp;
   return;
  }
```

1	2	3	4	5	6	7	8	9	10
26	5	77	1	61	11	59	15	48	19

1) 1^{st} loop, I = 5,

Temp = list[5] = 61   child = 10 (root*2)

if   61 > 19, break;

2) 2^{nd} loop, I=4,

Temp= list[4] = 1, child = 4*2 = 8, (list[child] <list[child+1]) => child = 9

compare (1 , 48) => list[4] = list[9]

list[9] = temp=1

1	2	3	4	5	6	7	8	9	10
26	5	77	**48**	61	11	59	15	**1**	19

3) 3^{rd} lop, I= 3,

temp = list[3] = 77, child = 3*2 = 6 (list[6] < list[7] => child = 7)

compare(77, > list[7]=59) =>break

1	2	3	4	5	6	7	8	9	10
26	5	**77**	48	61	11	59	15	1	19

4) 4^{th} loop, I = 2,

temp = list[2] = 5, child = 4 & (since l[4] <L[5], child => 5)

compare(5, list[5]=61) => list[2] =list[5],

1	2	3	4	5	6	7	8	9	10
26	**61**	77	48	61	11	59	15	1	19

child = child*2 = 10

compare (5, < list[10]=19) => list[5]=list[10], list[10]=temp

1	2	3	4	5	6	7	8	9	10
26	61	77	48	**19**	11	59	15	1	**5**

5) 5^{th} loop, I = 1,

temp = 26, child = 2,3=>chld=3

compare(26, <list[3]=77) => list[1] = list[3] child= 6

1	2	3	4	5	6	7	8	9	10
77	61	77	48	19	11	59	15	1	5

child=6,7 => child=7

compare(26, <list[7]=59)   list[3]=list[7], chld=12 => list[7]=26

1	2	3	4	5	6	7	8	9	10
77	61	**59**	48	19	11	**26**	15	1	5

결과는 MAXHEAP

77
61 59
48 19 11 26
15 1 5

- 다음은 max heap 을 HEAPIFY 한다.

- **Heapify**

1	2	3	4	5	6	7	8	9	10
77	61	59	48	19	11	26	15	1	5

1)1st LOOP: I=9,   SWAP (list[1], list[I+1], temp)

1	2	3	4	5	6	7	8	9	10
5	61	59	48	19	11	26	15	1	77

** adjust(HEAP, i, n)

- adjust(list, 1, i)

  temp=5.   Child=2   n=9

  - 2<9, child=(2,3) = 2 선택,
  - compare(5, list[2]=61) => list[1]=list[2], child=4

1	2	3	4	5	6	7	8	9	10
61	61	59	48	19	11	26	15	1	77

  - 4<9, child=(4,5)=4 선택,
    compare(5, list[4]=48)=>list[2]=list[4], child=8

1	2	3	4	5	6	7	8	9	10
61	**48**	59	48	19	11	26	15	1	77

  - 8<9, child=(8,9)=8 선택,
  - compare(5, list[8]=15)=>list[4]=list[8], child=16

1	2	3	4	5	6	7	8	9	10
61	48	59	15	19	11	26	15	1	77

  - list[8]=5

1	2	3	4	5	6	7	8	9	10
61	48	59	15	19	11	26	5	1	77

결과=>

61

48 59

15 19 11 26

5 1

2) 2nd LOOP: I=8,   SWAP (list[1], list[I+1], temp)

1	2	3	4	5	6	7	8	9	10
1	48	59	15	19	11	26	5	61	77

- adjust(list, 1, I) , I = 8
  temp=1, Child=2 n=8
  - 2<8, child=(2,3) = 3 선택,
  - compare(1, list[3]=59) => list[1]=list[3], child=6

1	2	3	4	5	6	7	8	9	10
59	48	59	15	19	11	26	5	61	77

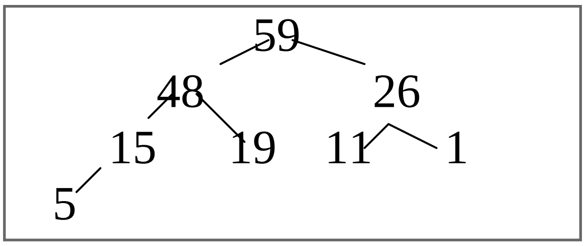
  - 6<8, child=(6,7)=7 선택,
  - compare(1, list[7]=26)=>list[3]=list[7], child=14

1	2	3	4	5	6	7	8	9	10
59	48	**26**	15	19	11	26	5	61	77

  - list[7]=1

1	2	3	4	5	6	7	8	9	10
59	48	26	15	19	11	**1**	5	61	77

결과=>



3) 3rd LOOP: I=7, SWAP (list[1], list[I+1], temp)

1	2	3	4	5	6	7	8	9	10
5	48	26	15	19	11	1	59	61	77

- adjust(list, 1, I) , I = 7

  temp=5, Child=2 n=7

  - 2<7, child=(2,3) = 2 선택,
  - compare(5, list[2]=48) => list[1]=list[2], child=4

1	2	3	4	5	6	7	8	9	10
48	48	26	15	19	11	1	59	61	77

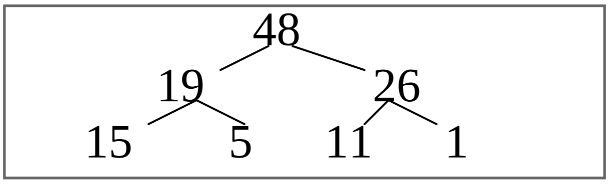
  - 4<8, child=(4,5)=5 선택,
  - compare(5, list[5]=19)=>list[2]=list[5], child=10

1	2	3	4	5	6	7	8	9	10
48	**19**	26	15	19	11	1	59	61	77

  - list[5]=5

1	2	3	4	5	6	7	8	9	10
48	19	26	15	**5**	11	1	59	61	77

결과=>



- Sorted List: 1, 5, 11, 19, 26, 48, 59, 61, 77

- Heap sort 요약 28, 7, 79, 3, 64, 15, 58, 17, 46,23

*** create MAX HEAP**

28
7 79
3 64 15 58
17 46 23

n div 2 => 5, 5 번노드부터 시작

1) 5 번노드 -> ok
2) 4 번노드 -> 46
17 3
3) 3 번노드 -> ok
4) 2 번노드 -> 64
46 23
17 3 7

5) 1 번노드 79
64 58
46 23 15 28
17 3 7

* Heapify

(79)
64
46 58
17 23 15 28
5 3 7

(79, 64)
58
46 28
17 23 15 7
5 3

(79, 64, 58)
46
23 28
17 3 15 7
5

(79, 64, 58, 46)
28
23 15
17 3 5 7

(79, 64, 58, 46, 28)
23
17 15
7 3 5

(79, 64, 58, 46, 28, 23)
17
7 15
5 3

(79, 64, 58, 46, 28,23, 17)
15
7 3
5

(79, 64, 58, 46, 28,23, 17, 15)
7
5 3

결과 => (79, 64, 58, 46, 28,23, 17, 7, 5, 3)

2) Quick Sort (C.A.R. Hoare) (가장 좋은 평균 수행도)

*algorithm:

Key 선정후,

data < key => left of key, data>key => right of key

⇨ becomes two sublists and then keep doing the same way

quick-sort (list, left, right) <= 순환 알고리즘

```
int list[], left, right;
{
  int i,j, pivot;

  if (left <right)   {
        i = left;
        j = right;
        pivot = list[left];
        do   {
              do i++;
                    while (list[i] <pivot);
              do j--;
                    while (list[j] >pivot);
              if (i < j)
                    swap(list[i] , list[j]);
         }   while (i<j);

        If   list[left]> list[j]   swap(list[left], list[j]);
        quick-sort(list, left, j-1);
        quick-sort(list, j+1, right);
     }
}
```

ex) 6, 3, 8, 2, 7, 1, 9 , 4

⇨ select key : 6

⇨ Do two searches (from left >6, from right <=6)

(6) 3 8 .... 4 발견

=> exchange (8, 4)

(6)3, 4, 2, 7, 1, 9, 8 발견

(6) 3, 4, 2, 1, 7, 9, 8 => 교환

(3, 4, 2, 1) 6 (7, 9, 8): 교환 및 분할

⇨ (6, 3, 8, 2, 7, 1, 9, 4) 초기상태

⇨ 1st pass: (3 4 2 1) 6 (7 9 8)

⇨ 2nd pass: (1 2) 3 (4) 6 (7) (9 8)

⇨ 3rd pass: (1 2 3 4 6 7) 8 9

ex) 8 (2 13 5 14 3 7) => 8 (2 7 5 3 14 13)

(2 7 5 3) 8 (14 13)

2 (7 5 3) 13 14

(5 3) 7

3 5

- Analysis:

=> worst case 는 리스트의 크기가 n-1 인 왼쪽 서브화일과 크기 0 인 오른쪽 서브화일로 될경우: $O(n^2)$

=> 바람직한 경우(best case)는 리스트가 균등하게 나뉘는것 : O(nlogn) 최대순환 깊이는 logN

=> Quick sort 의 장점은 average case 도 역시 O(nlogn)

3) Insertion Sort

```
For (i=1; i<n; i++) {
     key = list[i];
     For (j = i-1;   j>=0 && key <list[j];   j-- )
            list[j+1] = list[j];
     list[j+1]=key;
  }
```

예) 리스트 (7 -5 2 16 4)를 insertion sort 로 정렬하라

- Insertion sort is Good for small list ($n \leq 20$)
- Worst case $O(n^2)$, Average case $O(n^2)$, best case $O(n)$

4) Selection sort

```
for (I=0; I<n-1; I++)    {
     min = I;
     for (j=I+1; j<n; j++)
             if (list[j] < list[min])    min = j;
     swap(list[j],    list[min]);
  }
```

ex) (3,7,5,12,4)를 selection sort 를 이용하여 sort 하라

( )(3,7,5,12,4)
→(3)(7,5,12,4)
→ (3,4) (5,12,7)
→ (3,4,5) (12,7)
→ (3,4,5,7) (12)
→ (3,4,5,7,12)

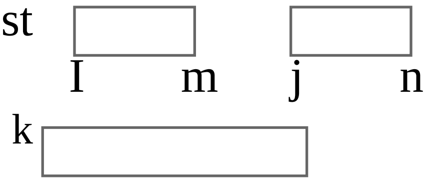
- 수행시간 - best, average, worst case => $O(n^2)$

5) Merge Sort - O(nlogn)

- Merge two sorted lists

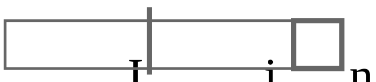
(list[$_{i}$],..list[$_{m}$]) & (list[$_{m+1}$],....x[$_{n}$]) into a single sorted list (sorted[$_{i}$],....sorted[$_{n}$])

```
void merge (list[], sorted[], I,m,n)
{
  int j, k, t;
  j = m+1;    /index for second sublist
  k = I; /index for sorted list
```



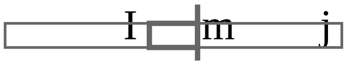

```
  while (I<= m && j<=n)      {
        if (list[I] <= list[j])
              sorted[k++] = list[I++];
        else
              sorted[k++] = list[j++];
  }

  if (I > m)
```




```
        for (t=j; t<=n; t++)    sorted[k+t-j] = list[t]

  else
```




```
      for (t=i; t<=m; t++)    sorted[k+t-i] = list[t]
}
```

6) Radix Sort

MSD sort (Most Significant Sort), LSD sort (Least Significant Sort)

- Radix sort has Several Keys   msd   lsd

  숫자의 경우는, ($0 \le k^i \le 999$)   ex) 1   2   3

  $0 \le k^i \le 9$ => requires only 10 bins (in case of decimal),

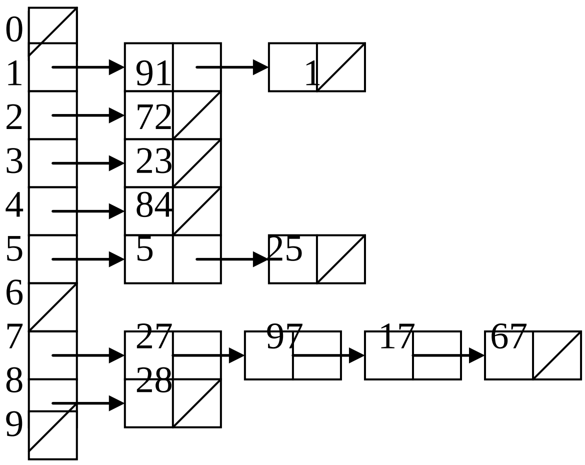
  and (it can be decimal, binary,...)

ex1)	3		19		27		115
	27		27		25		27
	115		17		19		25
	12	=>	115	=>	17	=>	19
	19		25		115		17
	4		4		12		12
	25		3		4		4
	17		12		3		3

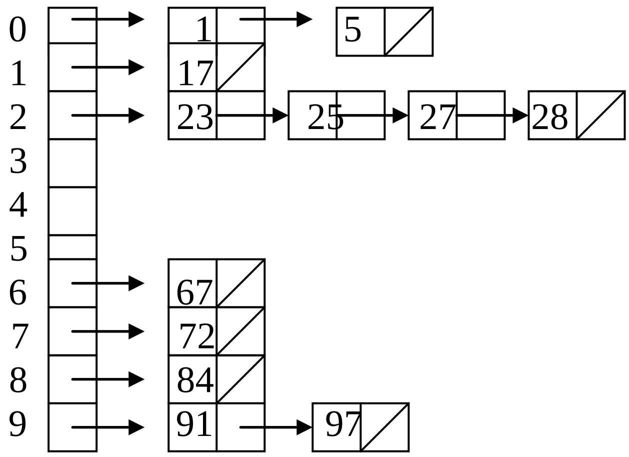
ex) List: 27   91   1   97   17   23   84   28   72   5   67   25

: using 10 bins (buckets)

First pass



Second pass



Result of first pass: 91, 1, 72, 23, 84, 5, 25, 27, 97, 17, 67, 28

Result of second pass: 1, 5, 17, 23, 25, 27, 28, 67, 72, 84, 91, 97

- **Summary of Internal sorting**
  - Insertion: good for small n
  - Merge : best-worst case and requires more storage than Heap
  - Radix : depends on size of keys and choice of r
  - Quick : best average case