Journal of The Korea Society of Computer and Information
Vol. 17 No. 4 , April 2012

2012-17-4-4-2

범위 피벗 퀵정렬

이 상 운*

Quicksort Using Range Pivot

Sang-Un, Lee*

요 약

퀵정렬은 피벗값을 최좌측, 최우측, 중간 값 또는 랜덤하게 결정하는 방법을 적용하고 있다. 본 논문은 범위 중간값을 P_0 피벗값으로 결정하고 계속적으로 양분하는 퀵정렬 방법을 제안하였다. 제안된 방법은 길이가 n인 리스트 A의 $i=1, j=n$에서 최소값 L과 최대값 H를 탐색하여 초기 피벗 키 값으로 $P_0=(H+L)/2$로 설정하고 $i=j$ 또는 $i>j$가 될 때까지 $a[i] \geq P_0, a[j] < P_0$를 교환하는 방식을 적용하였다. 피벗값에 따라 값의 교환이 완료되면 리스트 A_0는 2개의 부분리스트 $a[1] \leq A_1 \leq a[j]$과 $a[i] \leq A_2 \leq a[n]$으로 분할되며 이 경우 피벗 키는 $P_1=P_0/2,\ P_2=P_0+P_1$으로 설정된다. 이러한 과정을 분할된 리스트의 길이가 2일 때까지 수행되며, 리스트 길이가 2이고, $a[1]>a[2]$이면 $a[1] \leftrightarrow a[2]$로 교환한다. 제안된 방법은 퀵정렬에 비해 빠르며,, 최악의 경우 수행 복잡도 $O(n^2)$을 $O(n\log n)$으로 향상시켰다.

▸ 키워드 : 퀵정렬, 피벗, 분할-정복 전략, 범위

Abstract

Generally, Quicksort selects the pivot from leftmost, rightmost, middle, or random location in the array. This paper suggests Quicksort using middle range pivot P_0 and continually divides into 2. This method searches the minimum value L and maximum value H in the length n of list A. Then compute the initial pivot key $P_0=(H+L)/2$ and swaps $a[i] \geq P_0, a[j] < P_0$ until $i=j$ or $i>j$. After the swap, the length of list A_0 separates in two lists $a[1] \leq A_1 \leq a[j]$ and $a[i] \leq A_2 \leq a[n]$ and the pivot values are selected by $P_1=P_0/2,\ P_2=P_0+P_1$. This process repeated until the length of partial list is two. At the length of list is

• 제1저자 : 이상운
• 투고일 : 2012. 01. 04, 심사일 : 2012. 01. 25, 게재확정일 : 2012. 02. 15.
* 강릉원주대학교 멀티미디어공학과 (Dept. of Multimedia Science, Gangneung-Wonju National University)

www.kci.go.kr

two and $a[1] > a[2]$, swaps as $a[1] \leftrightarrow a[2]$. This method is simpler pivot key process than Quicksort and improved the worst-case computational complexity $O(n^2)$ to $O(n\log n)$.

▸ Keywords : Quicksort, Pivot, Divide-and-conquer strategy, Range

I. 서론

데이터베이스나 인터넷에서 원하는 데이터를 빠른 속도로 탐색 (search)하기 위해서는 저장된 데이터가 정렬 (sort)되어 있어야만 한다. 길이가 n인 리스트 (list)에서 임의의 값을 탐색하는 가장 빠른 방법은 이진 탐색법 (binary search)으로 $O(\log n)$의 수행 복잡도를 갖고 있으나 이 방법을 적용하기 위해서는 반드시 데이터가 정렬되어 있어야만 한다. 정렬 방법에는 다양한 방법들이 존재하지만 퀵정렬 (Quicksort)이 최악의 경우 $O(n^2)$, 최적의 경우 $O(n\log n)$의 수행 복잡도로 가장 빠른 방법으로 알려져 있다[1-3].

퀵정렬을 적용하기 위해서는 리스트 (list)를 양분하기 위한 기준 값인 피벗 (Pivot)을 결정해야 하며, 만약, 리스트가 분할되었을 경우 분할된 해당 하위 리스트 (sub list)내의 데이터를 다시 정렬시키기 위해 해당 하위 리스트의 시작점과 끝점을 저장하고 있어야만 한다[1].

퀵정렬은 피벗값을 최좌측, 최우측, 중앙값 또는 랜덤하게 선택한다[4]. 본 논문에서는 주어진 데이터 범위 (최대값-최소값)의 중간값을 피벗으로 활용하는 방법을 제안한다.

2장에서는 퀵정렬을 고찰해 본다. 3장에서는 간단하면서도 퀵정렬보다 빠른 범위피벗을 적용한 퀵정렬을 제안한다. 4장에서는 실험 데이터에 대해 퀵정렬과 범위피벗 퀵정렬을 비교하여 본다.

II. 퀵정렬

퀵정렬은 분할정복 전략 (strategy of divide-and-conquer)을 통해 리스트를 정렬하는 방법으로 다음과 같이 수행한다[1].

Step 1. 리스트에서 최좌측 (leftmost), 최우측 (rightmost), 중앙 (middle), 중위 (median), 랜덤 등 다양한 방법을 적용하여 하나의 원소를 피벗 (pivot, P)으로 선택한다.

Step 2. $P = a[k]$를 $a[n]$으로 이동시킨다. $a[k] \leftrightarrow a[n]$

Step 3. i는 최좌측 $a[1]$에서 순방향 탐색으로 피벗보다 큰 값 $a[i] \geq P$을 탐색하고, j는 최우측 $a[n-1]$에서 역방향으로 피벗보다 작은 값 $a[j] \leq P$를 탐색하여 $i < j$로 교차하지 않으면 $a[i] \leftrightarrow a[j]$로 상호 교환한다. 만약, $j < i$로 교차한다면 $a[i] \leftrightarrow a[k]$로 교환한다. 일단 피벗값의 위치가 결정되면 피벗값 위치를 기준으로 좌우로 리스트를 분할 (partition)한다. 이와 같이 수행하면 피벗 앞에는 피벗보다 작은 값들이 위치하고, 피벗 뒤에는 큰 값들이 위치한다.

Step 4. 모든 분할된 부분 리스트들 (sub-lists)의 길이가 0 또는 1이 될 때까지 Step 1~ Step 3을 반복 수행한다.

퀵정렬은 재귀 호출이 1회 진행될 때마다 좌측과 우측에서 동시에 탐색하면서 최소한 하나의 피벗값 위치는 최종적으로 위치가 결정된다. 따라서, $n, n-1, n-2, \cdots, 2, 1$로 리스트의 길이가 감소하며, 감소된 리스트는 $n, n/2, n/4, \cdots$로 계속해서 양분되기 때문에 전이진트리 (full binary tree)의 레벨 l에 대해 $2^{l-1} < n < 2^l$인 l회가 수행된다. 이는 수행 복잡도가 $O(\log n)$이다. 따라서 퀵정렬은 n개 데이터에 대해 평균적으로 $O(\log n)$이 수행된다. 그러나 최악의 경우 불균형적인 분할이 되면 $O(n)$이 수행된다. 따라서 퀵정렬은 최적 또는 평균적으로는 수행 복잡도가 $O(n\log n)$이며, 최악의 경우 $O(n^2)$이다. 이는 그림 1에 제시되어 있다. 퀵정렬을 수행하려면 모든 분할 리스트에 대한 Pivot Index, 리스트 시작점과 끝점을 저장하고 있어야만 한다.

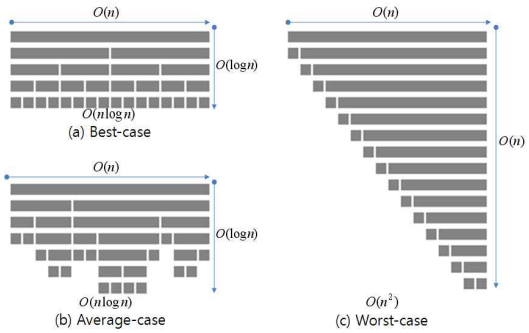


그림 1. 퀵정렬 수행 복잡도
Fig. 1. Time complexity of Quicksort

데이터 정렬에 있어서 수행 복잡도 $O(n\log n)$보다 빠른 방법은 일반적으로 존재하지 않는 것으로 알려져 있다. 평균적인 시간 복잡도가 $O(n\log n)$인 정렬 알고리즘으로는 퀵정렬 이외에 병합정렬 (Merge sort), 힙정렬 (Heap sort), 팀정렬 (Timsort), 이진트리정렬 (Binary tree sort)과 평활정렬

www.kci.go.kr

(Smoothsort)도 있다.

III. 범위 피벗 퀵정렬

정렬하고자 하는 데이터는 랜덤하게 저장된 데이터가 대부분이며, 이미 오름차순 또는 내림차순으로 정렬된 데이터는 거의 존재하지 않는다고 할 수 있다. 그러나 이러한 희박한 존재 가능성에 대해서는 굳이 퀵정렬을 수행하지 않아도 되며, 일반적인 랜덤하게 배열된 데이터에 대해서만 퀵정렬을 수행하면 수행시간이 보다 빠를 수 있다.

본 장에서는 퀵정렬 방법이 주어진 데이터가 오름차순, 내림차순 또는 랜덤하게 저장되어 있는지 판정하지 못하는 단점을 보완하였으며, 그림 2에 제시되어 있다.

```
s : number of swap
L : 최소값, H : 최대값
L= H= 0.
i = 1, j = n.
for     i = 1 to ⌊ n/2 ⌋
        if   a[i] > a[j] then a[i]↔a[j]
        /* L과 H를 구함.
        if      a[i] < a[j]
                if      a[j] > H then H←a[j]
                else if  a[i] < L then L←a[i]
        else if  a[i] > a[j]
                if      a[i] > H then H←a[i]
                else if  a[j] < L then L←a[j]
        i←i+1, j←j−1
end
P0 = (H+L)/2
push P0 /* Stack 이용
while   스택=Bottom do /* O(logn)
        pop Top
        i = 1, j = i+1, s = 0.
        for     i = 1 to n−1
                if   a[i] > a[j]   then   s = 1,   exit.
                     i←i+1, j←i+1.
        end
        if      s = 0 then Next Partition 수행
        else if  s = 1 then
i = 1, j = n, Range Pivot Quicksort()
end
Range Pivot Quicksort()
a[i] ≥ P, a[j] < P 탐색.
while   i > j do
        a[i]↔a[j], a[i] ≥ P, a[j] < P 탐색
        if      i > j
                {
                리스트를 a[j]∈l1 의 l1 과 l2 로 분할,
                P1 = P0/2, P2 = P0 + P1,
                push P2, l2 리스트 범위,
                push P1, l1 리스트 범위,
                }
```

그림 2. 범위피벗 퀵정렬
Fig. 2. Range Pivot Quicksort

제안된 알고리즘은 초기 피벗 값 P_0을 $L+(H-L)/2 = (L+H)/2$로 단 1회만 계산한다. 이후부터의 피벗값은 다음과 같이 계산된다. 만약, i번째 레벨에 대해 k번째 리스트 A_{ik}의 피벗값을 P_{ik}, 첫 번째 리스트 A_{i1}의 피벗값을 P_{i1}이라 하면 $i+1$번째 레벨로 분할되는 부분 리스트 $A_{(i+1)k1}$, $A_{(i+1)k2}$에 대한 피벗값 $P_{(i+1)k1}, P_{(i+1)k2}$는 $P_{(i+1)k1} = P_{ik} - P_{i1}/2$, $P_{(i+1)k2} = P_{ik} + P_{i1}/2$가 된다. 또한, 해당 분할 리스트에 대해 $i=[1,n/2], j=[n,n/2]$에 대해 $a[i]\leftrightarrow a[j]$로 정렬한다. 이 과정에서 내림차순 데이터가 오름차순으로 정렬된다. 다음으로 $i=[1,n/2]$, $j=i+1$에 대해 $a[i] > a[j]$가 하나라도 존재하면 퀵 정렬을 수행하며, $s=0$이면 이미 오름차순으로 정렬되어 있는 상태이다.

범위피벗 퀵정렬은 $(L+H)/2$로 설정하고 $a[i] \geq P_0$, $a[j] \leq P_0$에 대해 $a[i]\leftrightarrow a[j]$ 과정을 거치는 퀵정렬 방법이다. 만약, 해당 리스트가 다시 P_1과 P_2의 피벗값을 가진 부분 리스트로 분할되는 경우 $P_1 = P_0/2, P_2 = P_0 + P_1$으로 설정된다. 이러한 과정을 각 분할된 리스트의 길이가 2일 때까지 수행되며, 피벗값과 부분분할 리스트의 시작과 끝은 스택(stack)에 저장된다. $a[i]\leftrightarrow a[j]$를 수행하기 위해서는 버퍼 1개가 필요하다. 만약, 분할 리스트의 길이가 2이고, $a[1] > a[2]$이면 $a[1]\leftrightarrow a[2]$를 수행한다.

최좌측 또는 중간 값을 피벗값으로 적용하는 퀵정렬에서는 피벗값을 리스트의 끝부분으로 옮기는 과정이 필요하지만 제안된 방법은 이 과정이 불필요하며, 피벗값이 주어진 리스트에 없는 경우에도 전혀 상관없는 장점이 있다. 또한, 제안된 방법은 피벗 기준값 P_0가 일단 결정되면 나머지 피벗값들은 계속적으로 양분하면서 계산되어 피벗값을 결정하는 방법이 간단하다.

범위피벗 퀵정렬에 대해 리스트가 분할되는 과정은 그림3과 같다.

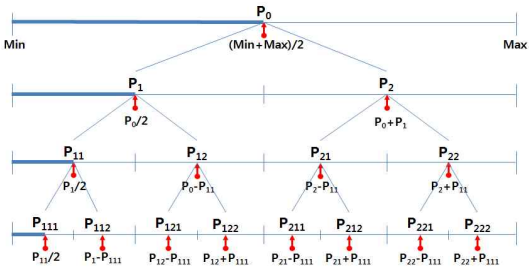


그림 3. 범위피벗 퀵정렬 피벗 분할
Fig. 3. Fivot partition of range pivot Quicksort

만약, 주어진 데이터가 오름차순이나 내림차순 데이터가 존재하지 않고, 대부분 랜덤하게 저장된 데이터라 가정할 경우, Verify sorting()을 적용하지 않고 Range Pivot Quicksort()만을 적용한다고 가정하여 보자. 모든 값의 빈도수 (frequency)가 유사하다면 제안된 알고리즘은 균형있는 분할이 수행되어 최적의 경우와 최악의 경우 모두 수행복잡도가 $O(n\log n)$임을 알 수

www.kci.go.kr

있다. 이는 퀵정렬의 최악의 경우 수행 복잡도 $O(n^2)$을 개선한 결과로 퀵정렬의 수행복잡도를 항상 $O(n\log n)$로 할 수 있는 획기적인 방법이다.

범위피벗 퀵정렬과 기존의 퀵정렬의 차이점은 다음과 같다.

(1) 퀵정렬은 피벗값을 최좌측, 중간, 또는 임의의 값을 적용할 경우 최우측으로 이동시키며, 일단 피벗값이 원하는 위치에 배치되면 고정된다. 반면에, 범위피벗 퀵정렬은 단 1회의 초기 피벗값만을 계산하고, 이를 계속 양분하면서 적용하며, 피벗값이 정수가 아니어도 상관없다.

(2) 퀵정렬은 i와 j가 교차시 피벗 P인 $a[n]$과 $a[i]$에 대해 $a[i]\leftrightarrow a[n]$을 수행하고 $a[1:P-1],a[P],a[P+1:n]$으로 분할하나 범위피벗 퀵정렬은 이 과정을 수행하지 않고 $a[1:j]$과 $a[i:n]$으로 양분한다.

(3) 기존의 퀵정렬은 분할 리스트 길이가 1 또는 0인 경우 알고리즘이 종료되나 범위피벗 퀵정렬은 2 이상의 길이에서도 오름차순으로 정렬되어 있으면 알고리즘이 종료되며, 최소 길이는 2이다. 따라서 이진트리의 레벨 수를 줄일 수 있다.

(4) 기존의 퀵정렬은 최악의 경우인 불균형 분할이 수행되는 경우 수행복잡도는 $O(n^2)$이다. 반면에, 이러한 오름차순이나 내림차순 데이터의 경우 $O(n)$으로 알고리즘이 종료되며, 랜덤한 데이터에 대해서는 $O(n\log n)$의 복잡도를 갖고 있다.

IV. 적용 결과 및 분석

그림 4의 (a), (b)와 (c)는 리스트 길이가 10인 데이터로 각각 오름차순으로 정렬된 데이터, 내림차순으로 정렬된 데이터와 랜덤하게 저장된 데이터이다. (d)는 Wikipedia에서 제시하고 있는 퀵정렬 예제로 33개 데이터 리스트이다[5]. 그림 4의 (a), (b)와 (c) 데이터들에 퀵정렬을 적용한 결과는 그림 5에 제시되어 있다.

본 실험에는 최좌측 피벗을 선택한다고 가정하였다. 퀵정렬은 주어진 데이터가 오름차순 또는 내림차순으로 정렬되어 있는지 사전에 검증하지 않고 알고리즘을 적용한다. 이로 인해 오름차순으로 정렬된 데이터에 대해서는 $n-1$개의 피벗을 선택하면서 $n-1$회의 분할을 수행하여 수행복잡도는 $O(n^2)$이다. 만약, 오름차순 정렬 데이터에 최우측 피벗을 선택하는 경우도 동일하게 $O(n^2)$이 수행된다.

정렬은 각 레벨에 존재하는 피벗값들은 동시에 수행될 수 없으며 깊이 우선 방법 (depth-first search, DFS)으로 순차적으로 피벗값을 선택하면서 수행된다.

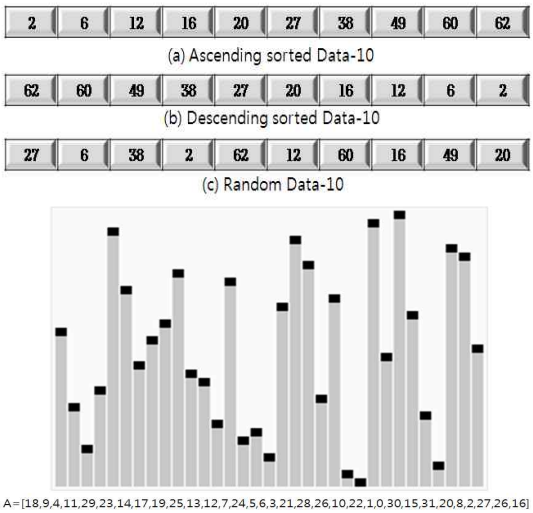
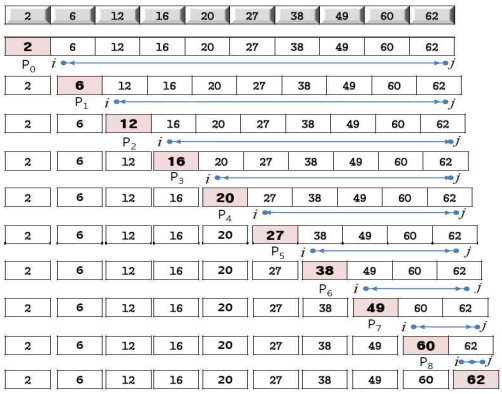
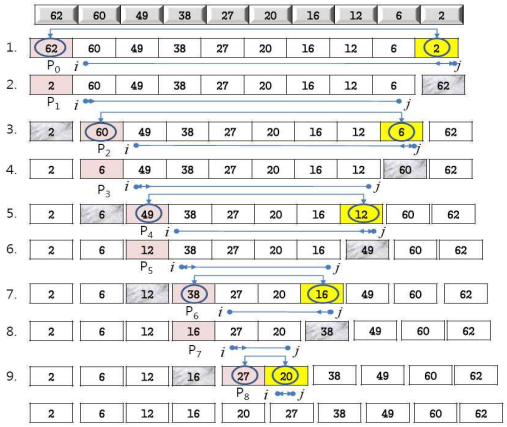


그림 4. 실험 데이터
Fig. 4. Experimental Data

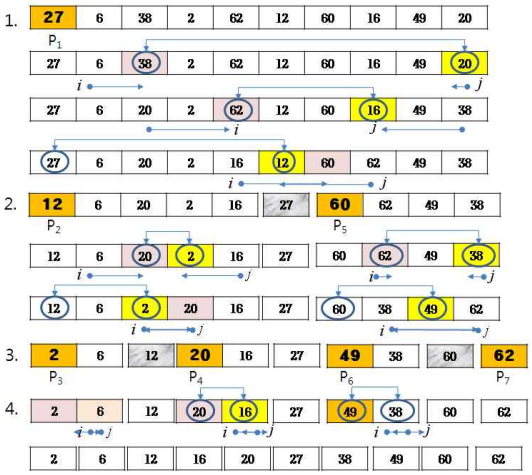


(a) 오름차순 정렬 데이터-10



(b) 내림차순 정렬 데이터-10

www.kci.go.kr



(c) 랜덤 저장 데이터-10
그림 5. 최좌측 피벗 퀵정렬
Fig. 5. Leftmost Pivot Quicksort

내림차순으로 정렬된 데이터의 경우 $\lfloor n/2 \rfloor$ 회의 교환(swap)이 발생하며, $n-1$개의 피벗 설정과 $n-1$회의 분할이 수행된다. 결국, 오름차순이나 내림차순으로 정렬되어 있는 데이터에 대해서는 균형 잡힌 분할이 되지 않아 (unbalanced partition) 수행복잡도가 $O(n^2)$이다. 랜덤하게 저장된 리스트의 경우 피벗값 $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6 \rightarrow P_7$의 7개 순서로 $O(n)$을 수행하였다. 랜덤하게 저장된 데이터 개수가 n인 전이진트리의 레벨 l $(2^{l-1} < n < 2^l)$에 대해 퀵 정렬의 수행 복잡도는 $O(n2^l) \simeq O(n \log n)$이다.

그림 4의 (a), (b)와 (c) 데이터에 범위피벗 퀵정렬을 적용한 결과는 그림 6에 제시되어 있다.

오름차순 정렬 데이터의 경우 $O(n)$ 수행으로 $s=0$를 얻어 실제의 범위피벗 퀵정렬을 수행하는 Range Pivot Quicksort()를 수행하지 않고 알고리즘이 종료되었다. 결국, 기존의 퀵정렬의 수행복잡도 $O(n^2)$을 $O(n)$으로 획기적으로 감소시켰다.

내림차순 정렬 데이터의 경우 퀵정렬은 9회의 피벗 선택으로 데이터 교환이 이루어졌으나 범위피벗 퀵정렬은 5회의 $a[i] \leftrightarrow a[j]$가 수행되어 $O(n)$의 수행복잡도를 나타내었다.

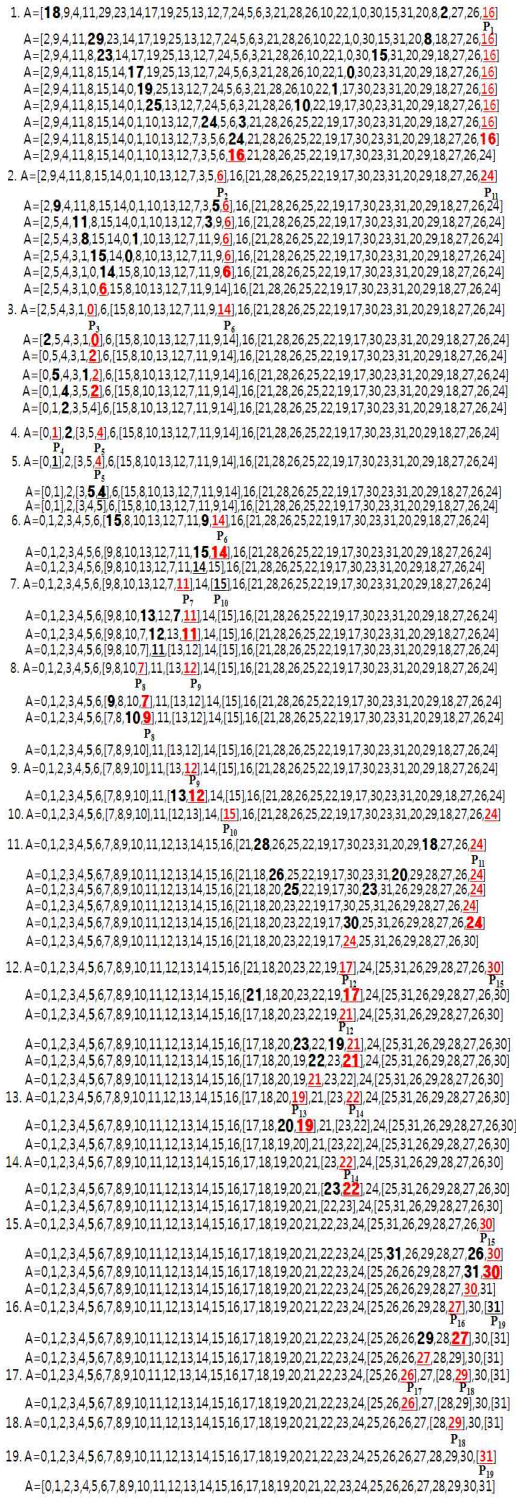
랜덤하게 저장된 데이터의 경우 범위피벗 퀵정렬은 기존의 퀵정렬과 동일하게 $O(n \log n)$의 수행복잡도를 갖지만 피벗값 $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6$의 6개 순서로 수행되어 피벗값 개수를 축소시켰다.

결론적으로, 오름차순과 내림차순 데이터의 경우 최좌측 피벗 퀵 정렬의 수행복잡도는 $O(n^2)$인데 반해 범위피벗 퀵정렬은 $O(n)$으로 향상시켰다. 랜덤하게 저장된 데이터에 대해서는 최좌측 피벗 퀵정렬과 수행복잡도가 $O(n \log n)$이나 피벗값 선택 횟수를 감소시켰다.

(c) 랜덤 저장 데이터-10
그림 6. 범위피벗 퀵정렬
Fig. 6. Range Pivot Quicksort

보다 큰 데이터인 그림 4의 (d) 랜덤 데이터-33 $(n=33)$에 대해 최우측 피벗 퀵정렬과 범위피벗 퀵 정렬을 수행한 결과는 그림 7에 제시되어 있다. 퀵정렬은 6개의 레벨 $(2^5 < 33 < 2^6)$에 대해 19개의 피벗값을 $O(n)$회 수행하는데 반해, 범위피벗 퀵 정렬은 5개의 레벨에 대해 15개의 피벗값을 $O(n)$회 수행하는 형태로 감소시켰다.

www.kci.go.kr



(a) 최우측 피벗 퀵정렬 방법

(b) 범위피벗 퀵정렬 방법

그림 7. 랜덤 데이터-33에 대한 퀵정렬
Fig. 7. Quicksort for Random Data-33

V. 결론

본 논문은 컴퓨터과학 분야에서 가장 오래 전부터, 가장 많은 관심을 가진 연구분야인 정렬방법에 대해 퀵정렬보다 간단하면서도 빠른 방법을 제안하였다.

제안된 방법은 오름차순과 내림차순 정렬 데이터의 경우 $O(n)$으로 정렬할 수 있다. 만약, 랜덤하게 저장된 데이터의 경우 제안된 방법은 범위의 중간값을 피벗값으로 결정하고 이를 계속적으로 양분하는 방식으로 피벗값 관리를 보다 쉽게 할 수 있는 방법을 적용하였으며, 분할된 리스트 길이는 최소 2이고, $a[1] < a[2]$일 경우 $a[1] \leftrightarrow a[2]$의 교환으로 알고리즘이 종료된다.

제안된 방법은 퀵정렬 방법에 비해 피벗값 결정이 보다 간단하면서도 최악의 경우 수행복잡도 $O(n^2)$을 $O(n\log n)$으로 향상시켰으며, 최적, 최악과 평균 수행복잡도는 모두 동일하게 $O(n\log n)$인 장점을 갖고 있다.

참고문헌

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms, Chapter 7. Quicksort", 2nd Ed., pp. 145-164, MIT Press, 2005.

[2] D. B. Ring, "A Comparison of Sorting Algorithms", http://www.devx.com/vb2themax/Article/19900, 2003.

[3] S. Nilson, "The Fastest Sorting Algorithm?", http://drdobs.com/architecture-and-design/184404062,

www.kci.go.kr

Dr. Dobb's Journal, Vol. 311, pp. 38-45, 2000.
[4] R. Sedgewick, "Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching", 3rd Ed., Addison-Wesley, 1998.
[5] Wikipedia, "Quicksort," http://en.wikipedia.org/ wiki/ Quicksort, 2011.

저 자 소 개

이 상 운 (Sang-Un, Lee)

1983년~1987년 : 한국항공대학교 항공전자공학과(학사)
1995년 ~ 1997년 : 경상대학교 컴퓨터과학과 (석사)
1998년 ~ 2001년 : 경상대학교 컴퓨터과학과 (박사)
2003년 : 강원도립대학 컴퓨터응용과 전임강사
2004년 ~ 2007.2 : 국립 원주대학 여성교양과 조교수
2007.3 ~ 현재 : 강릉원주대학교 과학기술대학 멀티미디어공학과 부교수
관심분야 : 소프트웨어 프로젝트 관리, 소프트웨어 개발 방법론, 소프트웨어 척도, 분석과 설계 방법론, 소프트웨어 시험 및 품질보증, 소프트웨어 신뢰성, 신경망, 뉴로-퍼지, 그래프 알고리즘
e-mail : sulee@gwnu.ac.kr

www.kci.go.kr

www.kci.go.kr