The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 15, No. 6, pp.95-102, Dec. 31, 2015. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2015.15.6.95
JIIBC 2015-6-13

가상의 기수계수버킷 정렬

Virtual Radix Counting Bucket sort

이상운*

Sang-Un Lee*

요 약 데이터를 정렬하는 방법들 중 $O(n\log n)$보다 빠른 방법은 알려져 있지 않고 있으며, 가장 빠른 방법으로 퀵정렬이 있으며, 이 정렬법은 n개의 데이터에 대해 최적과 평균의 경우 $O(n\log n)$, 최악의 경우 $O(n^2)$ 수행 복잡도를 갖고 있다. 본 논문에서는 리스트를 기수 숫자별로 빈도수를 계수하여 해당 가상 버킷에 저장하는 가상분할방법을 적용하였다. 또한 추가적인 메모리를 최소화시키기 위해 리스트 상에서 해당 버킷에 데이터들을 이동시키는 방법을 적용하였다. 제안된 알고리즘은 주어진 숫자의 자리수 k만큼 분할되며, 각 자리수에 대해 수행복잡도가 $O(n)$으로 $O(kn)$ 알고리즘이다.

Abstract Generally, there is no sorting algorithm much faster than $O(n\log n)$. The quicksort has a best performance $O(n\log n)$ in best and average-case, and $O(n^2)$ in worst-case. This paper suggests virtual radix counting bucket sort such that counting the frequency of numbers in each radix digit, and moves the arbitrary number to proper virtual bucket in the array, and divides the array into radix digit numbers virtually. Also, this algorithm moves the data to proper location within an array for using the minimum auxiliary memory. This algorithm performs k-times such that the number of k digits in given data and the time complexity is $O(n)$. Therefore, this algorithm has a $O(kn)$ time complexity.

Key Words : Quicksort, Radix sort, Counting sort, Bucket sort

Ⅰ. 서 론

기업이나 소셜네트워크 서비스 (SNS)의 정형과 비정형 데이터로부터 새로운 가치를 창출하고자 하는 빅 데이터 (big data) 검색이 새로운 화두로 제기되고 있다. 폭증하는 빅 데이터에서 필요한 정보를 빠르게 검색하여 새로운 가치를 창출하기 위해서는 보다 빠른 탐색 방법이 요구된다.

정보검색에서 원하는 데이터를 빠른 속도로 탐색 (search)하기 위해서는 저장된 데이터가 정렬 (sort)되어 있어야만 한다. 오름차순으로 정렬된 길이가 n인 리스트 (list)에서 임의의 값을 탐색하는 가장 빠른 방법은 이진 탐색법 (binary search)으로 $O(\log n)$의 수행 복잡도를 갖고 있다.[1] 다양한 데이터 정렬 방법들 중에서 퀵 정렬 (quicksort)이 가장 빠른 방법으로 알려져 있다. 퀵 정렬은 하나의 리스트를 피벗 (pivot)값을 기준으로 계속적으로 양분하는 방법으로, 균형된 분할 (balanced partition)이 수행되는 최적과 평균의 경우 $O(n\log n)$의 수행 복잡도를, 불균형적으로 분할 (unbalanced partition)되는 최악의 경우 $O(n^2)$의 수행 복잡도를 갖고 있다.[1-3]

*정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 : 2015년 7월 3일, 수정완료 : 2015년 11월 7일
게재확정일자 : 2015년 12월 11일

Received: 3 July, 2015 / Revised: 7 November, 2015 / Accepted: 11 December, 2015
*Corresponding Author: sulee@gwnu.ac.kr
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

평균적인 시간 복잡도가 $O(n\log n)$인 정렬 알고리즘으로는 Quick sort, Merge sort, Heap sort, Tim sort, Binary tree sort와 Smooth sort가 있다.[3] 데이터 정렬에 있어서 지금까지는 수행 복잡도 $O(n\log n)$보다 빠른 방법은 일반적으로 존재하지 않는 것으로 알려져 있다.[4]

본 논문에서는 수행복잡도 $O(kn)$인 정렬 알고리즘을 제안한다. 2장에서는 가장 효율적인 방법으로 알려진 퀵 정렬법을 고찰해 본다. 3장에서는 리스트 상에서 가상적으로 기수 (자리수) 별 숫자의 빈도수 (계수)에 대한 버킷을 적용한 VRCB (virtual Rrdix counting bucket) 정렬법을 제안한다. 4장에서는 실험 데이터에 대해 최우측 피벗 퀵정렬과 가상 기수계수버킷 정렬법의 수행 복잡도를 비교하여 본다.

II. 퀵정렬

퀵 정렬 방법은 분할정복 전략 (strategy of divide-and-conquer)을 통해 리스트를 정렬하는 방법으로, 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.[1-3] 퀵 정렬의 피벗값은 최좌측, 최우측, 중앙값 또는 랜덤하게 선택하는 방식을 적용한다.[5] 퀵 정렬은 그림 1과 같이 수행한다.

Step 1.	모든 분할된 부분 리스트들 (sub-lists)의 길이가 0 또는 1이 될 때까지 Step 2~ Step 4를 반복 수행한다.
Step 2.	리스트에서 최좌측 (leftmost), 최우측 (rightmost), 중앙 (middle), 중위 (median), 랜덤 등 다양한 방법을 적용하여 하나의 원소를 피벗 (pivot, P)으로 선택한다.
Step 3.	$P = A[k]$ 를 $A[n]$으로 이동시킨다. $A[k] \leftrightarrow A[n]$
Step 4.	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, $j < i$이면 $A[i] \leftrightarrow A[k]$ 로 교환한다.

그림 1. 퀵 정렬
Fig. 1. Quick sort

퀵 정렬은 길이 n인 리스트를 계속적으로 양분하여 결국에는 각각이 길이 1이 되어야 하기 때문에 전이진 트리 (full binary tree)의 레벨 l에 대해 $2^{l-1} < n < 2^l$인 l회가 수행된다. 이는 수행 복잡도가 $O(\log n)$ 또는 $O(\log_2 n)$이다. 퀵 정렬은 그림 2와 같이 리스트가 균형된 분할 (balanced partition)이 수행되는 최적과 평균의 경우 $O(n\log n)$의 수행 복잡도를, 불균형적으로 분할 (unbalanced partition)되는 최악의 경우 $O(n^2)$의 수행 복잡도를 갖고 있다.[6]

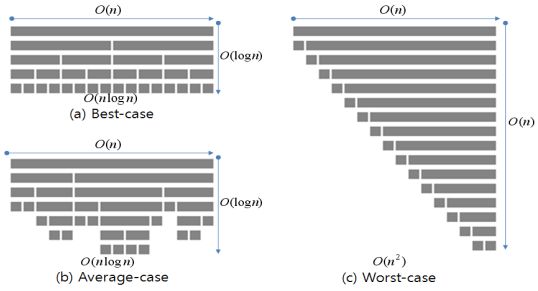


그림 2. 퀵정렬의 재귀 깊이
Fig. 2. Recursion depth of Quick sort

퀵 정렬에서 하나의 부분 리스트를 2개의 하위 부분 리스트로 균형있게 분할할 수 있는 최적의 방법은 주어진 데이터를 오름차순으로 정렬하고 중위값 (median)을 취하는 방법이다. 그러나 이 방법을 적용하려면 사전에 정렬을 시켜야만 하는 단점이 있다.

대용량의 데이터베이스 관리에 있어서, 초기에 데이터베이스 자료들을 생성하는 경우는 1회에 한정되며, 기존의 데이터베이스에 추가, 삭제 또는 수정되는 경우가 일상적으로 발생한다. 랜덤하게 저장된 초기 데이터베이스를 생성 데이터에 대해 퀵 정렬을 적용하면 수행복잡도는 $O(n\log n)$이 소요된다. 그러나 기존 대용량의 데이터가 정렬되어 있는 상태에서 일부분의 데이터가 추가 또는 삭제되어 재정렬을 수행하는 경우에는 최악의 경우인 $O(n^2)$이 필요한 경우가 보다 일반적인 현상이다. 이러한 경우에 대한 사례는 그림 3의 데이터에서 알 수 있다.

그림 3의 (a)는 14개의 랜덤하게 저장된 데이터로 초기에 생성된 데이터베이스를 정렬하는 경우이고, (b)는 기존에 정렬되어 저장된 데이터에 일부 데이터가 추가되어 재정렬을 수행해야 하는 경우이다.

그림 3의 (a) 데이터에 대해 최우측 피벗 퀵정렬을 수행한 결과는 그림 4와 같다.

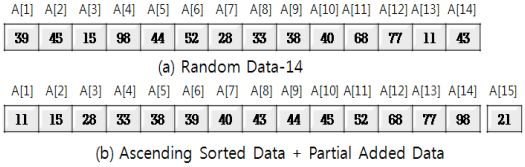


그림 3. 예제 데이터
Fig. 3. Example Data

$2^3 < n = 14 < 2^4$으로 레벨은 4까지 수행되어야 하나 일부 불균형 분할이 수행되어 레벨 5까지 수행되었으며, 실제 스택에 피벗을 저장하면서 수행하는 경우에는 10회가 수행된다. 따라서 수행 복잡도는 $O(n\log n)$이다.

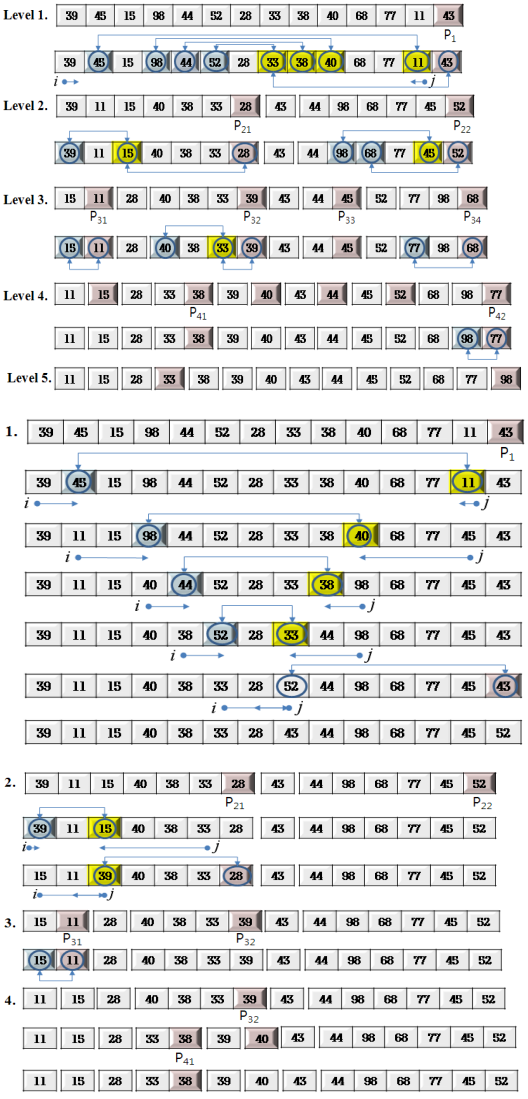
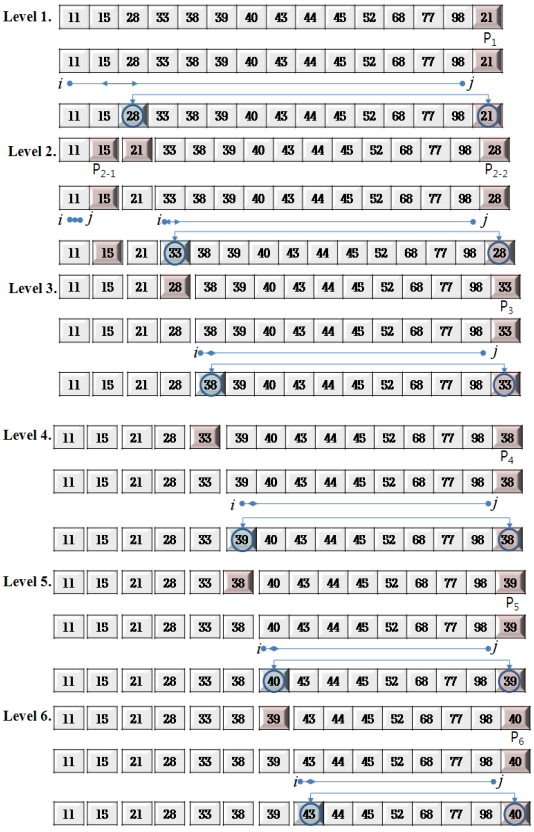


그림 4. 랜덤 데이터-14에 대한 퀵 정렬
Fig. 4. Quick sort for Random Data-14

그림 3의 (b) 데이터에 대해 최우측 피벗 퀵정렬을 수행한 결과는 그림 5와 같다. 이 경우에는 레벨 12까지 진행되어 수행복잡도는 $O(n^2)$이다.



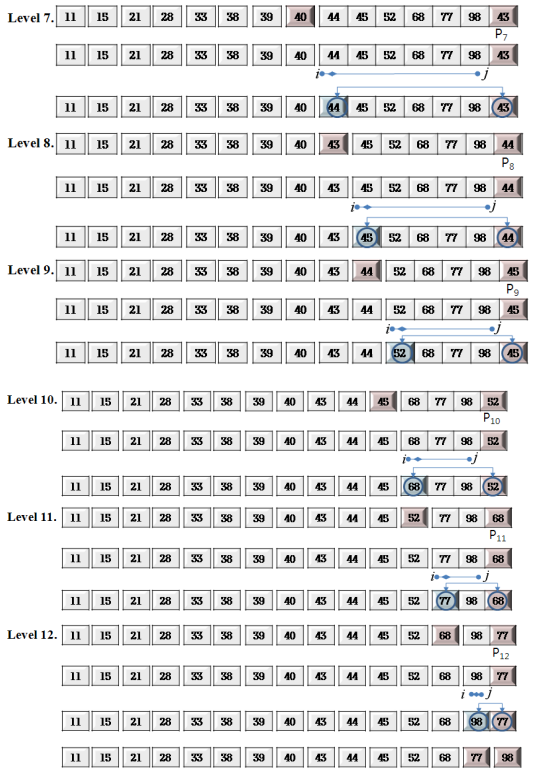


그림 5. 정렬 데이터에 일부 추가된 데이터의 퀵정렬
Fig. 5. Quick sort for Sorted & Partial Added Data-14

Ⅲ. 가상 기수계수버킷 정렬

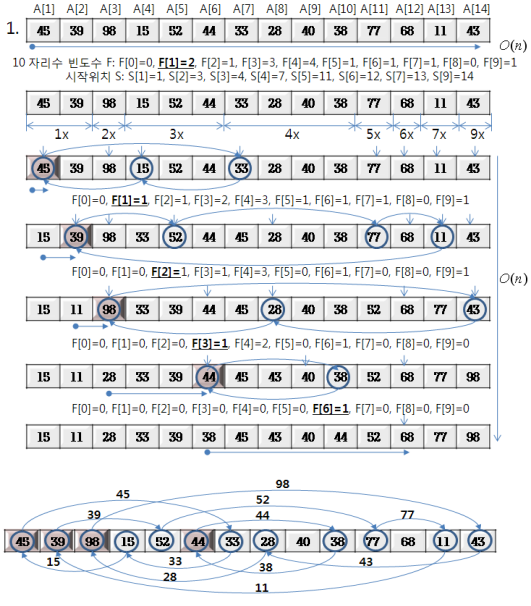
본 장에서는 기수(자리수)의 숫자별 빈도수(frequency)에 기반한 가변적인 가상 버킷으로 리스트를 분할하는 방식으로 기수 (radix), 버킷 (bucket)과 계수(counting) 정렬 방식[1]의 개념을 도입하였으며, 분할된 가상 버킷 범위에 대해 다음 자리수로 다시 분할하여 정렬하는 분할정복 개념의 퀵정렬 방식을 적용하였다. 따라서 기존의 다양한 정렬법들의 장점들을 적용한 하이브리드형 정렬방법을 제안한다. 제안된 정렬 알고리즘은 그림 6에 제시되어 있다.

제안된 알고리즘은 특정 값 $F[i]\neq 0$인 $A[S[i+1]-F[i]], d_k \neq i$로 자신의 값이 원하는 가상 버킷에 잘못 저장된 데이터에 대해 시작 지점이 $S[i]$인 정확한 가상버킷인 $d_k = i$번째 버킷의 $A[S[i+1]-F[i]]$ 위치로 이동시키는 방법을 적용하였다. 이 방법을 적용함으로 인해 이동시켜야 할 $d_k = i$인 $A[i]$값은 i번째 가상 버킷의 시작점 $S[i]$에서 현재 다른 가상 버킷으로 옮겨야할 값이 저장된 위치인 $A[S[i+1]-F[i]]$까지 계속적으로 탐색하는 시간을 단축시켰다.

Step 1.	/* 수행 복잡도 : $O(k)$ 리스트의 최대값 자리수 $d_j d_{j-1}\cdots d_2 d_1, k=j, j-1,\cdots,1$에 대해 Step 2와 3을 반복 수행한다.
Step 2.	/* 수행 복잡도 : $O(n)$ 동일한 $d_j d_{j-1}\cdots d_{k-1}$자리 숫자 범위에 대해 k 자리 숫자 $i=0,1,\cdots,9$ 에 대한 빈도수 $F_k[i]$를 계산한다. $S_k[1]=d_j d_{j-1}\cdots d_{k-1}$ 자리 숫자가 시작된 위치 $S_k[i]=A[S[i-1]+F[i-1])], i=2,3,\cdots,9$ $F[i]\neq 0$인 $A[S[i+1]-F[i]], d_k \neq i$ 값을 $d_k = i$로 시작하는 $S[i]$ 가상버킷의 $A[S[i+1]-F[i]]$로 이동시킨다. 이동된 $A[i]$의 $d_k = i$이면 $d_k \neq i$인 $A[i]$로 이동되고 이동된 길이를 $F[i]$에서 뺀다.
Step 3.	/* 수행 복잡도 : $O(n)$ $d_k \neq i$인 $A[S[i+1]-F[i]]$ 에 대해 $F[i]=F[i]-1$로 설정하고, $A[S[i+1]-F[i]]$ 에 저장된 기존의 값을 다시 $d_k = i$로 시작하는 $S[i]$ 가상버킷에서 다른 버킷으로 이동시킬 값이 저장된 $A[S[i+1]-F[i]]$로 계속적으로 이동시킨다. 단 $A[S[i+1]-F[i]]$의 $d_k = i$이면 $d_k \neq i$인 $A[i]$로 다시 이동되고 이동된 길이를 $F[i]$에서 뺀다.

그림 6. VRCB 정렬
Fig. 6. VRCB sort

제안된 알고리즘을 그림 3의 (a) 데이터에 적용한 결과는 그림 7과 같다.



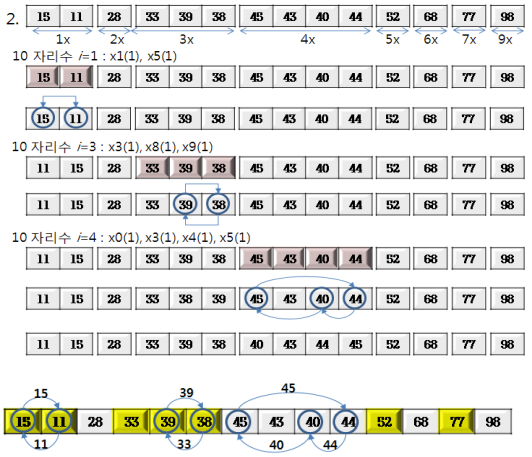


그림 7. 랜덤 데이터-14에 대한 VRCB 정렬
Fig. 7. VRCB sort for Random Data-14

10 자리수의 빈도수를 계산하면 F[0]=0, F[1]=2, F[2]=1, F[3]=3, F[4]=4, F[5]=1, F[6]=1, F[7]=1, F[8]= 0, F[9]=1이다. 따라서 각 가상 버킷의 시작번지는 S[1]=1, S[2]=3, S[3]=4, S[4]=7, S[5]=11, S[6]= 12, S[7]=13, S[9]=14이다. 따라서 10 자리수는 A[1]≤1≤A[2], A[3]=2, A[4]≤3≤A[6], A[7]≤4≤A[10], A[11]= 5, A[12]=6, A[13]=7, A[14]=9가 배치되도록 위치가 변경된다.

$F[1]=2\neq 0$인 A[S[2]-F[A]]=A[1]에 대해 A[1]= 45, $d_k=4$로 1x 버킷에서 4x 버킷으로 이동시켜야 하기 때문에 A[S[5]-F[4]]=A[7]인 A[7]=33, $d_k=3$ 위치로 이동시킨다. 이 때 F[4]=4-1=3이 된다. 즉, 10 자리수 "4"는 4개 중에 1개는 이미 자리를 차지하였음을 의미하며, 나머지 3개만 추가로 자리를 차지하면 된다. A[7]=33, $d_k=3$은 3x 버킷으로 이동시키기 위해 A[S[4]-F[3]]=A[4]인 A[4]=15, $d_k=1$로 이동되고 F[3]=3-1=2가 된다. A[4]=15, $d_k=1$은 A[S[2]-F[1]] =A[1]인 A[1]=45로 이동되고, F[1]=2-1=1이 된다. 따라서 A[1]→A[7]→A[4]→A[1] 사이클이 완료된다.

다음으로, F[1]=1이므로 A[S[2]-F[1]]=A[2]에서 출발하면, $A[2]=39, d_k\neq 1$이므로 $39, d_k=3$인 A[2]= 39→A[5]=52→A[11]=77→A[13]=11→A[2]의 사이클 순으로 값들이 이동된다. 이 과정에서 F[3]=1, F[5]=0, F[7] =0, F[1]=0이 된다.

다음으로, $F[2]=1\neq 0$인 A[S[3]-F[2]]=A[3]인 $A[3]=98, d_k\neq 2$에서 출발하면 A[3]=98→A[14]=32→A[8]=28→A[3]이 되며, $A[6]=44\rightarrow A[10]=38\rightarrow A[6]$이 된다. A[7]부터 A[14]까지는 모두 자신의 기수 범위에 저장되어 있어 더 이상의 이동은 없다.

이 과정을 종합하면 A[9]=40과 A[12]=68은 이미 자신의 버킷에 저장되어 있기 때문에 이동되지 않으며, 나머지 12개 데이터에 대해서는 A[1]=45,A[2]= 39,A[3]=98, A[6]=44 순으로 출발점으로 하여 값이 이동되는 사이클을 수행한다.

1 자리수에 대해서도 $O(n)$의 동일한 과정을 수행하면 A[1]=15→A[2]=11→A[1], A[5]=39→A[6]=38→A [5], A[7]=45→A[10]=44→A[9]=40→A[7]이 된다. 이 과정에서 리스트 길이가 1인 버킷은 수행되지 않는다.

제안된 알고리즘을 그림 3의 (b) 데이터에 적용한 결과는 그림 8과 같다. 퀵 정렬은 $O(n^2)$을 수행하는데 반해 제안된 알고리즘은 $O(2n)\simeq O(kn)$을 수행하여 보다 빠르게 결과를 얻을 수 있었다.

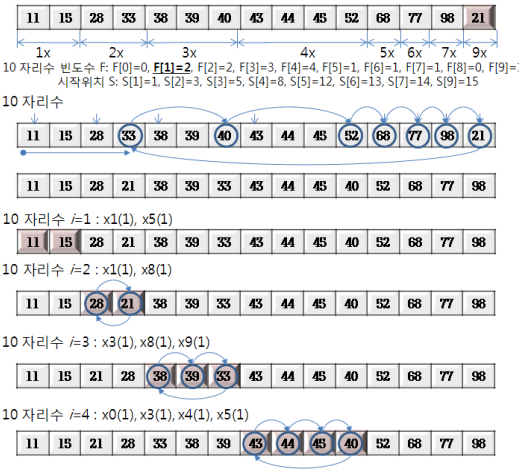


그림 8. 정렬 데이터에 일부 추가된 데이터의 VRCB 정렬
Fig. 8. VRCB Sort for Sorted & Partial Added Data-14

제안된 방법은 버킷정렬 (Bucket sort)과 같이 여분의 버킷을 준비하는 것이 아니라 리스트 상에서 기수정렬법 (Radix sort)의 각 자리수 숫자 $i=0,1,\cdots,9$에 대한 버킷 $B[i]$는 $S[i]\leq B[i]<S[i+1]$ 범위를 갖도록 가상적인 가변 길이를 가진 것으로 간주한다. 각 버킷의 길이는 계수정렬법 (Counting sort)의 빈도수 계산법을 적용하였다. 다음 자리수 d_k에 대해서는 이전 자리수까지가 동일한 값 $d_jd_{j-1}\cdots d_{k-1}$의 버킷 범위에 한정되어 수행되기 때

문에 가상적인 분할개념이 도입되어 분할정복 개념을 도입한 퀵정렬 개념과 유사한 방법이다. 이 가상 버킷에 해당 숫자들을 저장하는 방법은 추가적인 버킷을 준비하지 않고 리스트 상에서 시작 위치까지 되돌아오는 사이클이 형성될 때까지 순차적으로 데이터들을 이동시키는 방법을 적용하였다.

제안된 알고리즘은 해당 숫자 범위의 정확한 위치를 빠르게 찾기 위해 $F[i]$ 값을 하나씩 감소시키면서 $A[S[i+1]-F[i]]$를 찾는 방법을 적용하였다. 이 방법은 이미 해당 숫자 범위에 값들이 다수 배정된 이후 시작점 $S[i]$부터 $A[i] \notin S[i]$까지 순차적으로 찾아가는 경우의 시간을 단축시킬 수 있었다. 따라서 수행 복잡도는 $O(kn)$이다.

제안된 방법이 퀵정렬, 버킷정렬, 계수정렬 (Counting sort)과 기수정렬과의 차이점은 다음과 같다.

(1) 퀵정렬은 주어진 리스트 길이 n을 $2^{l-1} < n < 2^l$인 l회의 레벨로 분할한다. 반면에 제안된 방법은 주어진 데이터의 최대 값의 자리수 k회의 레벨로 분할된다.

(2) 버킷정렬은 여분의 버켓수 k개를 별도로 준비하여 각 버킷 범위에 속한 수들로 분류하고, 각 버킷 내의 데이터를 다시 정렬하는 방법이다. 반면에, 제안된 방법은 각 자리수에 나타나는 수들 만큼의 가상 버킷에 대해 각 버킷의 가변적인 범위를 리스트 상에서 결정하여 수행한다.

(3) 계수정렬은 리스트상에서 별개 (distinct)의 값들에 대한 여분의 리스트를 최대값 k로 준비하여 해당 킷값 번지수에 빈도수를 계산하여 각 수들의 범위를 결정하고, 출력 리스트의 해당 위치로 이동시키는 방법이다. 이 방법은 $k \gg n$이면 효율성이 떨어져 $k \leq n$인 경우 적용된다. 반면에, 제안된 방법은 수백만 건의 데이터가 있다고 하더라도 각 자리수에서 실제 나타나는 값들에 대한 계수 버킷으로 숫자의 경우, 최대 10개, 부호는 2개, 영문자는 26개가 소요되며, 출력 리스트를 별도로 필요로 하지 않는다.

(4) 기수정렬은 각 자리수에 대해 LSD 우선 (least-significant-digit first) 정렬법으로 큐 (queue)와 버킷을 적용한다. 반면에, 제안된 방법은 MSD 우선 (most-significant-digit first)으로 각 자리수의 해당 가상 버킷에 해당되는 리스트의 범위로 값들을 이동시키는 방법이다.

제안된 방법은 주어진 데이터의 최대값 자리수를 k라 할 때 수행복잡도는 $O(kn)$이다.

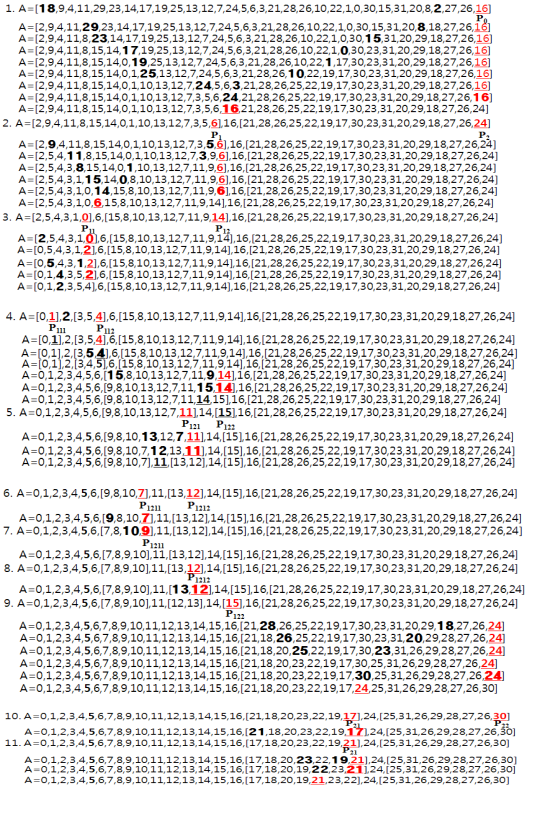
Ⅳ. 적용 및 결과 분석

본 장에서는 랜덤하게 저장된 33개 데이터인 그림 9에 대해 최우측 피벗 퀵정렬과 기수 계수가상버킷 정렬을 수행하여 수행복잡도를 비교하여 본다.

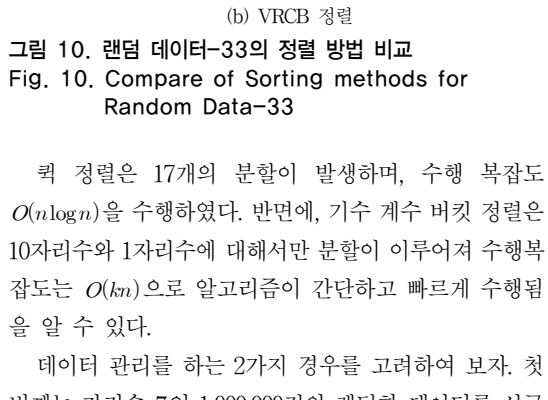
A=[18,9,4,11,29,23,14,17,19,25,13,12,7,24,5,6,3,21,28,26,10,22,1,0,30,15,31,20,8,2,27,26,16]

그림 9. 실험 데이터
Fig. 9. Experimental Data

그림 9의 33개 데이터에 대해 최우측 피벗 퀵정렬과 기수 계수 정렬법을 적용한 결과는 그림 10에 제시되어 있다.



(a) 최우측 피벗 퀵정렬



(b) VRCB 정렬

그림 10. 랜덤 데이터-33의 정렬 방법 비교
Fig. 10. Compare of Sorting methods for Random Data-33

퀵 정렬은 17개의 분할이 발생하며, 수행 복잡도 $O(n\log n)$을 수행하였다. 반면에, 기수 계수 버킷 정렬은 10자리수와 1자리수에 대해서만 분할이 이루어져 수행복잡도는 $O(kn)$으로 알고리즘이 간단하고 빠르게 수행됨을 알 수 있다.

데이터 관리를 하는 2가지 경우를 고려하여 보자. 첫 번째는 자리수 7인 1,000,000건의 랜덤한 데이터를 신규 생성하여 정렬하는 경우이며, 두 번째는 기존의 정렬된 1,000,000건의 데이터에 신규로 1,000건이 추가되어 재정렬을 하는 경우이다. 데이터관리에 있어서는 두 번째 경우가 보다 빈번히 발생하여 현실성이 있는 경우이다.

첫 번째 경우에 대해서는 제안된 알고리즘은 데이터 자리수에만 영향을 받아 $O(k)=7$로 ,000,000회가 수행된다. 그러나 퀵 정렬은 데이터 개수 n에 영향을 받아 균형된 분할이 수행될 경우 $2^{20}<1,000,000<2^{21}$로 21개 레벨로 분할되어 $O(\log n)=21$로 1,000,000회가 수행된다. 따라서 $O(k)$는 $O(\log n)$에 비해 3배 정도 효율적임을 알 수 있다.

두 번째 경우에 대해서는 제안된 알고리즘은 7,007,000회를 수행하지만 퀵 정렬은 이미 정렬된 데이터에 대해서는 1개와 $n-1$개씩으로 불균형적으로 분할되어 최악의 경우인 $O(n^2)$이 수행되어 1,000,001,000,00909회가 수행된다. 이 경우 제안된 알고리즘이 7.007×10^{-6}배 정도 효율적임을 알 수 있다.

만약, 테라바이트의 대용량 빅 데이터의 경우에는 두 번째 경우에 대한 제안된 알고리즘의 효율성을 퀵정렬에 비해 획기적인 효율성이 있다고 할 수 있다. 결국, 제안된 $O(kn)$ 알고리즘은 데이터의 자리수가 크지 않고 대규모의 데이터에 대해서는 퀵정렬의 $O(n\log n)$보다 성능이 우수함을 알 수 있다.

V. 결론

본 논문은 정렬방법에서 가장 빠른 방법으로 알려진 퀵 정렬의 수행복잡도 $O(n\log n)$에 비해 $O(kn)$의 알고리즘을 제안하였다.

제안된 방법은 기수정렬법에 따라 주어진 데이터의 각 자리수의 숫자 $i=0,1,\cdots,9$별 범위에 해당 자리수의 값을 이동시키고 리스트를 가상적으로 분할하고, 각 가상적으로 분할된 리스트들에 대해 다음 자리수에 대해 동일한 방법으로 정렬하는 방법으로 자리수 k에 대해 $O(n)$이 수행된다. 따라서 제안된 알고리즘은 $O(kn)$의 수행 복잡도를 갖고 있으며, 단지 추가적으로 필요한 메모리는 길이가 10인 F와 S이다.

References

[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, ISBN: 978-0262033848, 2005.

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

[3] R. Sedgewick, "Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching", 3rd Ed., Addison-Wesley, ISBN:

978-0201314526, 1998.

[4] S. Nilson, "The Fastest Sorting Algorithm?", http://drdobs.com/architecture-and-design/184404062, Dr. Dobb's Journal, Vol. 311, pp. 38-45, 2000.

[5] H. W. Lang, "Sequential and Parallel Sorting Algorithms, Quicksort", FH Flensberg, http://www.nf.fh-flensburg.de/lang/algorithmen/sortieren/quick/quicken.htm, 2011.

[6] C. A. R. Hoare, "Quicksort," The Computer Journal, Vol. 5, No. 1, pp. 10-16, doi: 10.1093/comjnl/5.1.10, 1962.

저자 소개

이 상 운(정회원)

- 1987년 : 한국항공대학교 항공전자공학과 (학사)
- 1997년 : 경상대학교 컴퓨터과학과 (석사)
- 2001년 : 경상대학교 컴퓨터과학과 (박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년 ~ 2007.2 : 국립 원주대학 여성교양과 조교수
- 2007.3 ~ 2015.3 : 강릉원주대학교 멀티미디어공학과 부교수
- 2015.4 ~ 현재 : 강릉원주대학교 멀티미디어공학과 정교수

<관심분야 : 소프트웨어 프로젝트 관리, 개발 방법론, 분석과 설계 방법론, 시험 및 품질보증, 소프트웨어 신뢰성, 그래프 알고리즘>

- e-mail : sulee@gwnu.ac.kr