The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 15, No. 5, pp.61-68, Oct. 31, 2015. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2015.15.5.61
JIIBC 2015-5-7

빠른 계수 정렬법의 제안

Proposal of Fast Counting Sort

이상운*

Sang-Un, Lee*

요 약 데이터를 비교 정렬하는 방법들 중 $O(n\log n)$보다 빠른 방법은 알려져 있지 않고 있으며, 가장 빠른 퀵 정렬법은 최적과 평균의 경우 $O(n\log n)$, 최악의 경우 $O(n^2)$ 수행 복잡도를 갖고 있다. 본 논문은 비교 정렬법이 아닌 $O(n+k),(k=$최대치$)$의 계수 정렬법을 보다 빠르게 수행하는 $O(n+k)$의 단순 계수정렬법과 데이터의 자리 수 l의 숫자별 빈도수를 계수하여 해당 가상 버킷에 저장하는 $O(ln)$의 기수 계수 정렬법을 제안하였다. 6개의 실험 데이터에 제안된 알고리즘을 적용한 결과, 퀵 정렬의 $O(n\log n)$ 또는 $O(n^2)$을 $O(n+k)$ 또는 $O(ln)$으로 단순화 시킬 수 있었다. 결론적으로 제안된 방법은 계수정렬법과 퀵 정렬법에 비해 보다 빠른 방법이다.

Abstract Among comparison sorts, no algorithm excels a current set lower bound of $O(n\log n)$ in operation. Quicksort, the fastest of its kind, has a complexity of $O(n\log n)$ at its best and on average and $O(n^2)$ at worst. This paper thus presents two methods: first is an $O(n+k)$ simple counting sort which operates much more speedily than an $O(n+k)$, $(k=\max imum\,value)$counting sort, and second is an $O(ln)$ radix counting sort which counts the frequency of numbers in the digit l of a data and saves it in a corresponding virtual bucket in an array, only to virtually divide the array into radix digit numbers. For the 6 experimental data, the proposed algorithm makes $O(n\log n)$ or $O(n^2)$ of Quicksort simple into $O(n+k)$ or $O(ln)$. After all, the proposed sorting algorithm has proved to be much faster than the counting sort and Quicksort.

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

Ⅰ. 서 론

정보검색에서 원하는 데이터를 빠른 속도로 탐색(search)하기 위해서는 저장된 데이터가 정렬 (sort)되어 있어야만 한다. 다양한 데이터 정렬 방법들 중에서 퀵정렬 (Quicksort)이 가장 빠른 방법으로 알려져 있다. 퀵정렬은 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]

평균적인 수행 복잡도가 $O(n\log n)$인 비교 정렬 (comparison sorting) 알고리즘으로는 퀵정렬 (Quicksort), 병합정렬 (Merge sort), 힙정렬 (Heap sort), Timsort, 이진트리정렬 (Binary tree sort)과 Smoothsort가 있다.[3] 비교정렬법에 있어서 수행 복잡도 $O(n\log n)$보다 빠른 방법은 일반적으로 존재하지 않는 것으로 알려져 있다.[4]

비교정렬법이 아닌 방법에는 비둘기집 정렬 (Pigeonhole

*정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 : 2015년 3월 24일, 수정완료 : 2015년 10월 2일
게재확정일자 : 2015년 10월 9일

Received: 24 March, 2015 / Revised: 2 October, 2015 / Accepted: 9 October, 2015
*Corresponding Author: sulee@gwnu.ac.kr
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

sort), 버킷정렬 (Bucket sort), 계수정렬 (Counting sort), 기수정렬 (Radix sort), Spread sort 등이 있으며, 수행 복잡도가 $O(n+k)$인 버킷정렬과 계수정렬법이 가장 빠른 것으로 알려져 있다. 계수정렬은 리스트의 최대치를 k, 길이를 n이라 할 때 계수 리스트 $C[1,2,\cdots,k]$를 준비하기 때문에 $A[i]=0$인 데이터를 처리하지 못하는 단점이 있다. 또한, $k \le n$인 경우에 한정되어 적용되며, $k \gg n$인 경우 효율성이 크게 저하된다.

본 논문에서는 계수정렬법의 "0"의 값을 처리하지 못하는 문제점을 해결하고, $k \gg n$인 경우에도 효율적으로 적용할 수 있는 정렬 알고리즘을 제안한다. 2장에서는 계수정렬을 고찰해 본다. 3장에서는 계수정렬을 빠르게 수행할 수 있는 $O(n+k)$의 빠른 계수정렬법 (Fast Counting sort)과 자리 수 (기수) l별로 계수정렬시키는 $O(ln)$의 기수계수정렬법 (Radix Counting sort)을 제안한다. 4장에서는 실험 데이터에 대해 제안된 정렬법의 효율성을 분석하여 본다.

II. 계수정렬법

본 장에서는 계수정렬법을 고찰해 본다. 계수정렬법은 그림 1과 같이 입력 리스트 $A[1,2,\cdots,n]$에 대해 최대치 $A[k]$를 찾아 계수 리스트 $C[1,2,\cdots,k]$와 출력 리스트 $B[1,2,\cdots,n]$를 준비한다. $A[1,2,\cdots,n]$의 각 키 값 i에 해당되는 $C[i]$에 빈도수를 계산한다. $C[1,2,\cdots,k]$의 각 i에 대해 종료 셀값으로 변경한다. $A[1,2,\cdots,n]$를 역으로 이동하면서 $A[i]$ 값을 $C[i]$의 위치인 $B[i]$에 저장하고, $C[i]$ 값을 1 감소시킨다. 이 알고리즘은 $O(k)+O(n)+O(k)+O(n)$을 수행하여 수행 복잡도는 $O(n+k)$이다.

```
입력 리스트 : A[1,2,···,n] , 계수 리스트 : C[1,2,···,k]
출력 리스트 : B[1,2,···,n]
for   i←1 to k /* k: 최대치
      do C[i]←0
for   j←i to n /* n: 리스트 길이
      do C[A[j]]←C[A[j]]+1
for   i←2 to k
      do C[i]←C[i]+C[i−1]
for   j←n downto 1
      do x←A[j]
         B[C[x]]←x
         C[x]−−
```

그림 1. 계수정렬
Fig. 1. Counting sort

그림 2는 계수정렬시 계수 리스트 C의 길이 k가 입력 리스트 A의 길이 n에 비해 길어 효율성이 저하되는 경우와 $A[i]=0$의 값을 갖는 경우 계수정렬법을 적용하지 못하는 사례를 제시하였다. (a)는 입력 리스트 A의 길이 $n=4$에 비해 최대치가 15로 계수 리스트 길이 $k=15$인 $C[1,2,\cdots,15]$를 준비해야 한다. (b)는 $n=3$에 비해 최대치가 750으로 계수 리스트 길이 $k=750$을, (c)는 $n=14$에 비해 $k=98$을 준비해야 한다. 지금까지 계수정렬은 데이터 값 $A[i]>0$에 대해서만 적용할 수 있었다. 만약, (d), (e)와 같이 $A[i]=0$이 존재하는 경우 계수정렬을 적용할 수 없다.

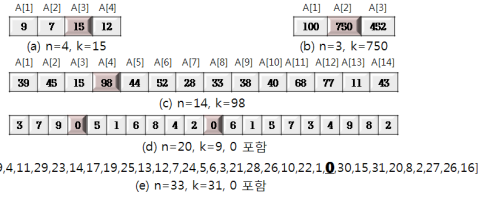


그림 2. 계수정렬의 효율성 저하와 적용 불가능 사례
Fig. 2. Low Efficiency and Inapplicable Counting sort Examples

길이 $n=5$인 리스트 $A=[4,1,3,4,5]$에 대해 계수정렬법을 적용한 결과는 그림 3과 같다.[5]

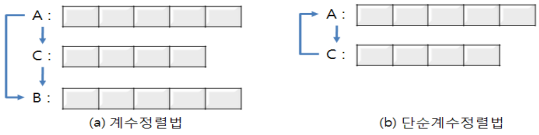
그림 3. 데이터-5에 대한 계수정렬
Fig. 3. Counting Sort for Data-5

III. 빠른 계수정렬법

본 장에서는 기수계수정렬법 (Radix Counting Sort)과 단순 계수정렬법 (Simple Counting Sort)을 제안한다.

계수정렬법과 빠른 계수정렬법의 차이점은 그림 4와 같다. 계수정렬법은 출력 리스트 B가 별도로 필요한데

반해 빠른 계수정렬법은 계수정렬법과 동일하게 A, C 리스트는 사용하지만 출력 리스트 B를 필요로 하지 않으며, "0"을 처리할 수 있는 특징을 갖고 있다.



(a) 계수정렬법 (b) 단순계수정렬법

그림 4. 계수정렬법과 단순계수정렬법 비교
Fig. 4. Counting sort vs. Simple Counting sort

main ()
입력 : $A[1,2,\cdots,n]$
for $i \leftarrow 1$ to n
리스트 길이 n, 최대치 k, 최대 자리 수 l 계산
if $n+(k+1) \le n+20l$ then
do SimpleCountingSort ()
else if $n+(k+1) > n+20l$ then
do RadixCountingSort ()

SimpleCountingSort ()
계수 : $C[1,2,\cdots,k,k+1]$
for $i \leftarrow 1$ to $k+1$
do $C[i] \leftarrow 0$
for $j \leftarrow 1$ to n /* $A[] \rightarrow C[]$
do $C[A[j]+1] \leftarrow C[A[j]+1]+1$
$j = 1$
for $i \leftarrow 1$ to $k+1$ /* $C[] \rightarrow A[]$
do $A[j,\cdots,j+C[i]-1] \leftarrow i-1,\ C[i] \neq 0$
$j \leftarrow j+C[i]$

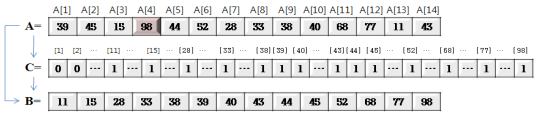
RadixCountingSort ()
Step 1. /* 수행 복잡도 : $O(l)$
리스트의 최대치 자리 수 $d_l d_{l-1} \cdots d_2 d_1, k = l, l-1, \cdots, 1$에 대해 Step 2와 3을 반복 수행한다.
Step 2. /* 수행 복잡도 : $O(n)$
동일한 $d_l d_{l-1} \cdots d_{k-1}$ 자리 숫자 범위에 대해 d_k 자리 숫자 $i = 0,1,\cdots,9$에 대한 빈도수 $F_k[i+1]$를 계산한다.
$S_k[1] = d_l d_{l-1} \cdots d_{k-1}$ 자리 숫자가 시작된 위치
$S_k[i] = A[S[i-1]+F[i-1])], i = 2,3,\cdots,9$
$F[i] \neq 0$인 $A[S[i+2]-F[i+1]], d_k \neq i$ 값을 $d_k = i$의 시작 지점이 $S[i+1]$인 가상버킷의 $A[S[i+2]-F[i+1]]$로 이동시킨다. 이동된 위치에 저장된 값 $A[]$의 $d_k = i$이면 $d_k \neq i$인 $A[S[i+2]-F[i+1]+j]$로 다시 이동되고 이동된 길이 j에 대해 $F[i+1] \leftarrow F[i+1-j]$ 로 변경시킨다.
Step 3. /* 수행 복잡도 : $O(n)$
$d_k \neq i$인 $A[S[i+2]-F[i+1]]$ 에 대해 $F[i+1] = F[i+1]-1$로 설정하고, $A[S[i+2]-F[i+1]]$에 저장된 기존의 값을 다시 $d_k = i$로 시작하는 $S[i+1]$ 가상 버킷에서 다른 버킷으로 이동시킬 값이 저장된 $A[S[i+2]-F[i+1]]$로 계속적으로 이동시킨다. 단 $A[S[i+2]-F[i+1]]$의 $d_k = i$이면 $d_k \neq i$인 $A[S[i+2]-F[i+1]+j]$ 로 다시 이동되고 이동된 길이 j에 대해 $F[i+1] \leftarrow F[i+1-j]$ 로 변경시킨다.

그림 5. 빠른 계수정렬법
Fig. 5. Fast Counting Sort

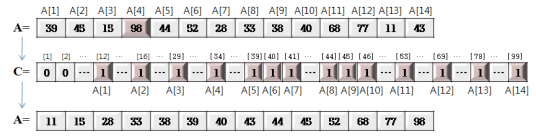
기수 계수정렬법은 기수 (자리 수)의 숫자별 빈도수 (frequency)에 기반한 가변적인 가상 버킷으로 리스트를 분할하는 방식으로 기수 (Radix), 버킷 (Bucket)과 계수 (Counting) 정렬 방식[1] 개념을 도입하였으며, 분할된 가상 버킷 범위에 대해 다음 자리 수로 다시 분할하여 정렬하는 분할정복 개념을 적용하였다. 여기서 데이터를 해당 가상 버킷으로 이동시키는 방법은 A 리스트 상에서 사이클 경로를 형성하여 다음 지점으로 시프트 시키는 방법을 적용하였다.

$A[1,2,\cdots,n]$에 대해 리스트 길이를 n, 최대 자리 수를 l, 최대치를 k라 하자. $n+(k+1)$과 $n+20l$을 계산하여 $n+(k+1) \le n+20l$이면 단순 계수정렬법을 $n+(k+1) > n+20l$이면 기수 계수정렬법을 적용한다. 제안된 빠른 계수정렬법은 그림 5에 제시되어 있다.

그림 2의 (c) 데이터-14에 대한 계수정렬, 단순 계수정렬과 기수 계수정렬법을 적용한 결과는 그림 6에 제시되어 있다. 본 데이터에 대해서는 $n+(k+1) = 113 > n+20l = 54$로 단순 계수정렬법에 비해 기수 계수정렬을 적용하는 것이 보다 좋음을 알 수 있다.



(a) 계수정렬법



(b) 단순 계수정렬법

(c) 기수 계수정렬법

그림 6. 데이터-14의 정렬
Fig. 6. Sorting for Data-14

계수정렬법은 $k=98$인 $C[1,2,\cdots,98]$을 계산하여 $A\rightarrow B$로 이동시키는데 반해, 단순 계수정렬법은 $k+1=99$인 계수 리스트 $C[1,99]$를 준비하여 "0"의 값도 처리할 수 있도록 하였다. 또한 C 리스트의 순서대로 $C[i]\neq 0$인 번지수 i의 빈도수 j에 대해 A 리스트에 순서대로 $i-1$값을 j번 반복하여 저장하는 방법을 적용하였다. 기수계수 정렬법은 10 자리 수의 숫자 $i=0,1,2,\cdots,9$에 대한 빈도수 $F[i+1]$을 계산하고, 각 숫자 i의 시작번지 $S[i+1]$를 계산한다. 따라서 10 자리 수는 $A[1]\leq 1.\leq A[2]$, $A[3]=2.$, $A[4]\leq 3.\leq A[6]$, $A[7]\leq 4.\leq A[10]$, $A[11]=5.$, $A[12]=6.$, $A[13]=7.$, $A[14]=9.$가 배치되도록 위치가 변경된다. 데이터의 10 자리 수 숫자별로 해당 가상 버킷으로 이동시키는 방법은 $F[2]=2\neq 0$인 $A[S[3]-F[2]]=A[1]$에 대해 $A[1]=45\,d_k=4$로 1x 버킷에서 4x 버킷으로 이동시켜야 하므로 $A[S[6]-F[5]]=A[7]$인 $A[7]=33, d_k=3$ 위치로 이동시킨다. 이 때 $F[5]=4-1=3$이 된다. 즉, 10 자리 수 "4x"는 4개 중에 1개는 이미 저장되어, 3개만 추가로 저장되면 됨을 의미한다. $A[7]=33, d_k=3$은 3x 버킷으로 이동시키기 위해 $A[S[5]-F[4]]=A[4]$인 $A[4]=15, d_k=1$로 이동되고 $F[4]=3-1=2$가 된다. $A[4]=15, d_k=1$은 $A[S[3]-F[2]]=A[1]$인 $A[1]=45$로 이동되고, $F[2]=2-1=1$이 된다. 따라서 $A[1]\rightarrow A[7]\rightarrow A[4]\rightarrow A[1]$의 사이클이 완료된다.

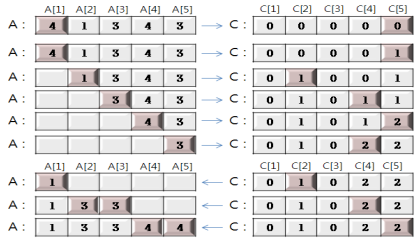
다음으로, $F[2]=1$이므로 $A[S[3]-F[3]]=A[2]$에서 출발하면, $A[2]=39, d_k\neq 1$이므로 $39, d_k=3$인 $A[2]=39\rightarrow A[5]=52\rightarrow A[11]=77\rightarrow A[13]=11\rightarrow A[2]$의 사이클 순으로 값들이 이동된다. 이 과정에서 $F[4]=1, F[6]=0, F[8]=0, F[2]=0$이 된다.

다음으로, $F[3]=1\neq 0$인 $A[S[4]-F[3]]=A[3]$인 $A[3]=98, d_k\neq 2$에서 출발하면 $A[3]=98\rightarrow A[14]=43\rightarrow A[8]=28\rightarrow A[3]$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.

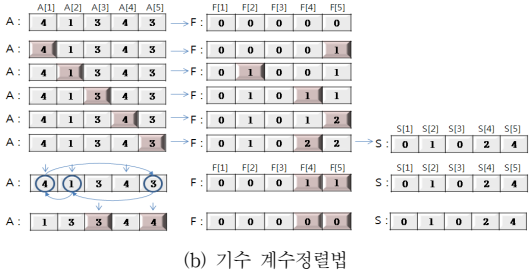
1 자리 수에 대해서도 $O(n)$의 동일한 과정을 수행하면 $A[1]=15\rightarrow A[2]=11\rightarrow A[1]$, $A[5]=39\rightarrow A[6]=38\rightarrow A[5]$, $A[7]=45\rightarrow A[10]=44\rightarrow A[9]=40\rightarrow A[7]$이 된다. 이 과정에서 리스트 길이가 1인 버킷은 수행되지 않는다.

Ⅳ. 적용 결과 및 분석

본 장에서는 $A=[4,1,3,4,3]$의 데이터-5와 그림 3의 나머지 4개 데이터에 대해 단순 계수정렬법과 기수 계수정렬법을 적용하여 성능을 분석하여 본다.



(a) 단순 계수정렬법



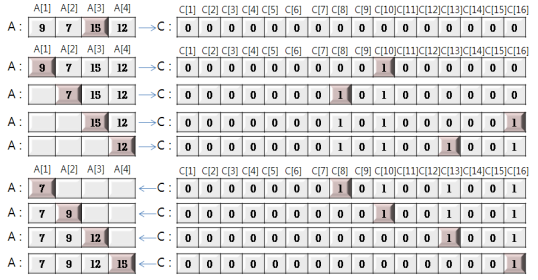
(b) 기수 계수정렬법

그림 7. 데이터-5에 대한 정렬법
Fig. 7. Sorting for Data-5

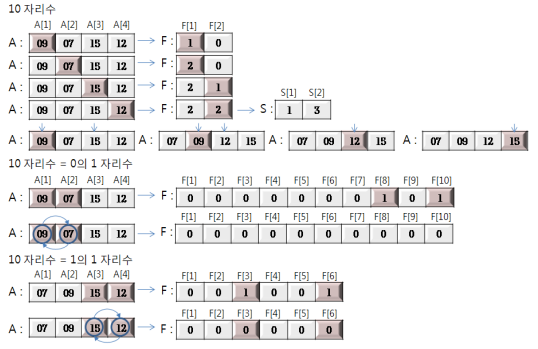
$A=[4,1,3,4,3]$의 데이터-5에 대해 제안된 단순 계수정렬과 기수 계수정렬법을 적용한 결과는 그림 7과 같다. 단순 계수정렬법은 출력 리스트 B를 사용하지 않는다. 또한, C에 대해 C'로 해당 숫자의 종료지점을 계산하지

않는 단순한 방법이다. 기수 계수정렬법은 1 자리 수 각 숫자 $i=0,1,\cdots,9$에 대해 빈도수 $C[i+1]$를 계산하며, 이로부터 각 숫자의 시작지점 $S[i+1]$을 계산하여 $A[j]$부터 $F[i+1]$의 개수만큼 $i+1$을 저장하는 방법이다. 본 데이터에 대해서는 $n+(k+1)=10\leq n+20l=25$로 기수 계수정렬법에 비해 단순계수정렬을 적용하는 것이 보다 간단함을 알 수 있다.

그림 3의 (a) 데이터-4, (b) 데이터-3, (d) 데이터-20, (e) 데이터-33에 대한 단순 계수정렬과 기수 계수정렬법을 적용한 결과는 그림 8 ~ 그림 11에 제시되어 있다.

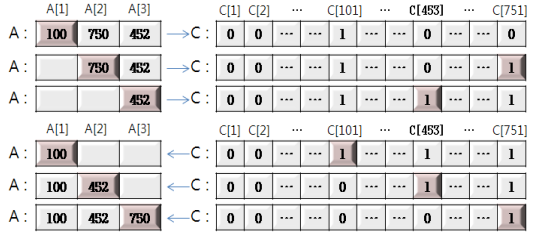


(a) 단순 계수정렬법

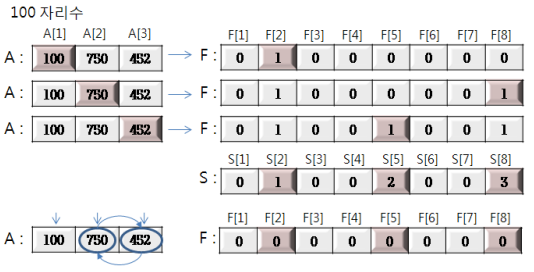


(b) 기수 계수정렬법

그림 8. 데이터-4의 정렬
Fig. 8. Sorting for Data-4

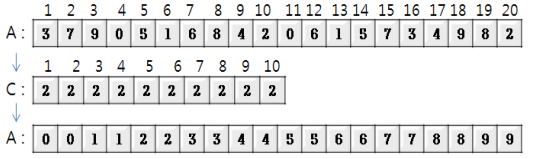


(a) 단순 계수정렬법



(b) 기수 계수정렬법

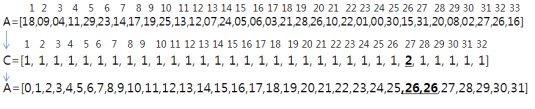
그림 9. 데이터-3의 정렬
Fig. 9. Sorting for Data-3



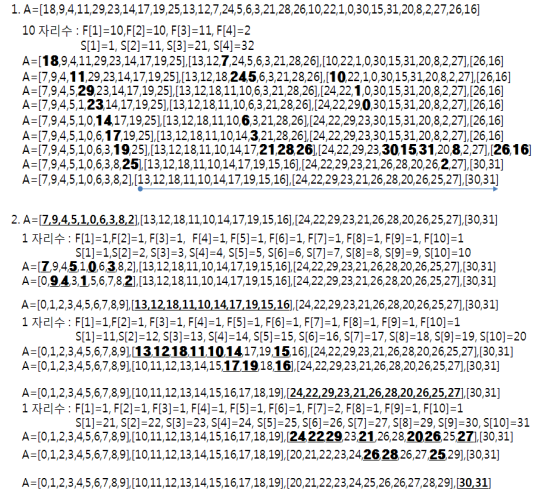
(a) 단순 계수정렬

(b) 기수 계수정렬

그림 10. 데이터-20의 정렬 방법 비교
Fig. 10. Compare of Sorting methods for Data-20



(a) 단순 계수정렬



(b) 기수 계수정렬

그림 11. 데이터-33의 정렬 방법 비교
Fig. 11. Compare of Sorting methods for Data-33

본 논문에서 적용된 6개의 데이터 사례에 대해 단순 또는 기수 계수정렬법을 선택하는 기준은 표 1과 같다. 표 1에 의하면 기수 계수정렬법이 2개, 단순 계수정렬법이 4개로 선정되었다.

표 1. 정렬법 선정 기준
Table 1. Selection Criteria for Sorting Method

데이터	n	k	l	선택기준		선택
				$n+(k+1)$	$n+20l$	
Data-3	3	750	3	754	63	기수 계수정렬
Data-4	4	15	2	20	44	단순 계수정렬
Data-5	5	4	1	10	25	단순 계수정렬
Data-14	14	98	2	113	54	기수 계수정렬
Data-20	20	9	1	30	40	단순 계수정렬
Data-33	33	31	2	65	73	단순 계수정렬

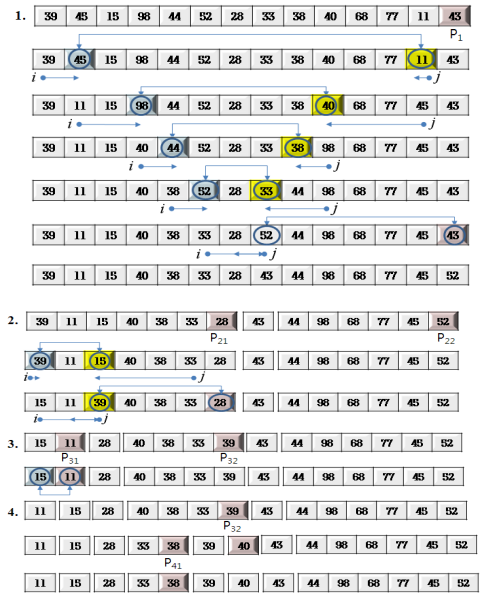
데이터 관리를 하는 2가지 경우를 고려하여 보자. 첫 번째는 자리 수 7 ($0{,}000{,}000 \le A[i] \le 9{,}999{,}999$)인 1,000,000건의 랜덤한 데이터를 신규 생성하여 정렬하는 경우이며, 두 번째는 기존의 정렬된 1,000,000건의 데이터에 신규로 1,000건이 추가되어 재정렬을 하는 경우이다. 데이터관리에 있어서는 두 번째 경우가 보다 빈번히 발생하여 현실성이 있는 경우이다.

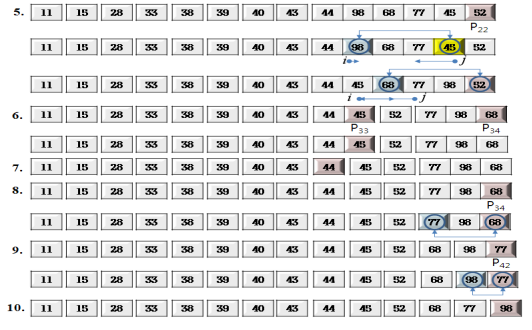
첫 번째 경우에 대해서는 $n+(k+1)=11{,}000{,}000 > n+20l=1{,}000{,}140$으로 단순 계수정렬법에 비해 기수 계수정렬법이 보다 빠름을 알 수 있다. 기수 계수정렬법은 데이터 자리 수에만 영향을 받아 $O(l)=7$로 7,000,000회가 수행된다. 그러나 퀵 정렬[6,7]은 데이터 개수 n에 영향을 받아 균형된 분할이 수행될 경우 $2^{20} < 1{,}000{,}000 < 2^{21}$로 21개 레벨로 분할되어 $O(\log n)=21$로 21,000,000회가 수행된다. 따라서 $O(l)$이 $O(\log n)$에 비해 3배 정도 효율적임을 알 수 있다.

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

참고로, 데이터-14와 데이터-33에 대한 퀵정렬을 수행한 결과는 그림 12와 같다. 데이터-14의 경우 $2^3 < n < 2^4$으로 레벨은 4까지 분할되어야 하지만 일부 불균형 분할이 발생하여 레벨 5까지 수행되었다.

만약, 테라바이트의 대용량 빅 데이터의 경우에는 두 번째 경우에 대한 기수 계수정렬법의 효율성은 퀵정렬에 비해 획기적으로 향상시킬 수 있을 것이다. 결국, 제안된 기수 계수정렬법의 수행 복잡도 $O(ln)$은 데이터의 자리수가 크지 않고 대규모의 데이터에 대해서는 퀵정렬의 $O(n\log n)$보다 성능이 우수함을 알 수 있다.





(a) 데이터-14

(b) 데이터-33

그림 12. 퀵정렬
Fig. 12. Quick sort

데이터-14의 경우 $2^3 < n < 2^4$으로 레벨은 4까지 분할되어야 하지만 일부 불균형 분할이 발생하여 레벨 5까지 수행되었다.

만약, 테라바이트의 대용량 빅 데이터의 경우에는 두 번째 경우에 대한 기수 계수정렬법의 효율성은 퀵정렬에 비해 획기적으로 향상시킬 수 있을 것이다. 결국, 제안된 기수 계수정렬법의 수행 복잡도 $O(ln)$은 데이터의 자리수가 크지 않고 대규모의 데이터에 대해서는 퀵정렬의 $O(n\log n)$보다 성능이 우수함을 알 수 있다.

V. 결 론

본 논문은 정렬방법에서 가장 빠른 방법으로 알려진 퀵정렬의 수행 복잡도 $O(n\log n)$에 비해 보다 빠른 방법인 $O(k+n)$의 단순 계수정렬법과 $O(kn)$인 기수 계수정렬법을 제안하였다. 여기서 k는 데이터의 최대치이며, $\log n$은 $l, (2^{l-1} < n < 2^l)$의 전이진트리의 레벨 수이다.

단순 계수정렬법은 계수정렬법의 $A \rightarrow C, (A, C) \rightarrow B$ 형태에서 출력 리스트 B를 사용하지 않고 $A \leftrightarrow B$로 단순화 시켰다. 기수 계수정렬법은 기수정렬법에 따라 주어진 데이터의 각 자리 수 숫자 $i = 0, 1, \cdots, 9$별 가상 버킷 범위에 해당 자리 수의 값을 순차적으로 이동시켜 첫 번째 위치에 도달할 때까지 사이클을 형성하는 방법을 적용하였다. 이 경우 해당 가상 버킷의 정확한 위치를 찾기 위해 빈도수 F와 가상버킷의 숫자별 시작위치 S의 메모리를 활용하였다.

$k+1 \leq n$인 경우 $O(k+n)$의 단순 계수정렬법을, $k+1 > n$인 경우 $O(kn)$의 기수 계수정렬법을 적용함으로 인해 비교정렬법 중에서 가장 빠른 퀵정렬의 $O(n\log n)$에 비해 수행시간을 단축시킬 수 있었다.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms, MIT Press, ISBN: 9780262033848, 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-13: 978-0201314526, 1998.

[4] S. Nilson, "The Fastest Sorting Algorithm?", Dr. Dobb's Journal, Vol. 311, pp. 38-45, Apr. 2000.

[5] P. Indyk and C. Wenk, "CS445: Introduction to Algorithms, Sorting in Linear Time", Dept. of Computer Science, The University of Arizona, 2007.

[6] H. W. Lang, "Sequential and Parallel Sorting Algorithms: Quicksort," FH Flensberg, 2011.

[7] 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 ~ 2007년 2월 : 국립 원주대학 여성교양과 조교수
- 2007년 3월 ~ 2015년 3월 : 강릉원주대학교 멀티미디어공학과 부교수
- 2015년 4월 ~ 현재 : 강릉원주대학교 멀티미디어공학과 정교수

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

- E-Mail : sulee@gwnu.ac.kr