정보 블록 정렬 알고리즘에 관한 연구

송태옥†

요 약

본 논문에서는 $O(NlogN)$의 시간 복잡도와 데이터의 분포상태에 영향을 받지 않는 정보블록 정렬알고리즘(IBSA : Information Block Sort Algorithm)을 제안하고, 시뮬레이터를 이용하여 그 성능을 평가하였다. 2백만 개의 랜덤 데이터를 이용하여 IBSA의 성능을 측정해본 결과, 퀵 정렬의 22%, 개선된 퀵 정렬의 36% 정도의 비교회수만으로도 정렬할 수 있음을 보여주었다.

A Study on Information Block Sort Algorithm

Tae-Ok Song†

ABSTRACT

In this paper, I proposed a sort algorithm named Information Block Sort Algorithm(IBSA) which is not influenced on distribution of data in the list and has time complexity of $O(NlogN)$. Also I evaluated the IBSA using a simulator. Performance analysis shows that, in case of sorting randomly generated two millions data, the number of actual comparisons has taken place about 36% of the number of comparisons in the improved Quick sort algorithm and 22% in Quick sort algorithm.

1. 서 론

정렬과 탐색은 기본적인 데이터처리의 방법으로서, 데이터 처리를 위한 응용 프로그램에서 많이 이용되고 있다. 그러므로 이러한 영역은 그동안 많이 연구 되어져왔으며, 성능 향상을 위해 계속 연구되고 있는 전산 과학의 한 분야이다.

본 연구는 이러한 연구의 일부분으로서, 정보블록 정렬 알고리즘(IBSA : Information Block Sort Algorithm)이라는 새로운 정렬 알고리즘을 제안하고자한다.

IBSA는 IBPA(Information Block Preprocessing Algorithm)[2]라는 전처리 과정을 거쳐, 데이터의 분포 상태를 파악하고, 데이터의 특성을 나타내는 정보블록을 생성하며, 이 정보블록의 정보를 이용하여 데이터를 재배치한 후 각각의 블록을 정렬하는 알고리즘이다.

본 연구에서 제안하는 IBSA의 성능을 일반적으로 빠른 정렬 알고리즘이라고 알려진 퀵 정렬[1,3, 5,6] 알고리즘 그리고 개선된 퀵 정렬 알고리즘[4]과 비교하고자한다.

실험에 이용될 데이터는 두 가지로써, 현재 시각을 기초로 생성되어 균등한 분포를 보이는 난수, 그리고 가우스 함수를 이용해 정규 분포를 보이는 난수를 이용한다. 이와 같이 데이터를 두 가지로 선정한 이유는 실험 데이터가 특정 범위에 데이터가 집중되는 정규 분포의 경우, 알고리즘의 성능 저하를 초래할 수 있기 때문이다.

아래에 나올 내용은 먼저 IBSA의 알고리즘을 분석한 후, 성능 비교를 위한 실험 결과를 그래

† 종 신 회 원: 관동대학교 컴퓨터교육과 조교수
논문접수: 2003년 4월 30일, 심사완료: 2003년 6월 10일

프로 제시하였다. 그리고 결론과 향후 연구 과제를 기술하였다.

2. 비교 알고리즘

IBSA와 비교할 알고리즘은 퀵 정렬 알고리즘과 개선된 퀵 정렬 알고리즘 두 가지로서, 알고리즘과 시간 복잡도를 중심으로 살펴보았다.

2.1 퀵 정렬 알고리즘

퀵 정렬 알고리즘[1,3,5,6]은 데이터 리스트를 피봇(pivot)을 기준으로 두 부분으로 분할한 다음, 각 분할된 부분을 재귀적으로 정렬하는 알고리즘이다.

퀵 정렬의 알고리즘은 <그림 1>과 같은 코드로 구현할 수 있다.

```
int partition ( Item a[], int l, int r )
{
  int i = l-1, j = r;
  Item v = a[r];
  for ( ; ; )
  { while (less(a[++i], v)) ;
    while (less(v, a[--j]))
    if (j == l) break;
    if (i >= j) break;
    exch (a[i], a[j]);
  }
  exch (a[i], a[r]);
  return i;
}

void quicksort ( Item a[], int l, int r )
{
  int i;
  if (r <= l) return;
  i = partition (a, l, r);
  quicksort (a, l, i-1);
  quicksort (a, i+1, r);
}
```

(그림 1) 퀵 정렬 알고리즘

퀵 정렬 알고리즘의 시간 복잡도는 <표 1>과 같다.

<표 1> 퀵 정렬의 시간 복잡도

경우	시간 복잡도
최적	O(N)
평균	O(NlogN)
최악	$O(N^2)$

2.2 개선된 퀵 정렬 알고리즘

세 값의 중위(Median-Of-Three)와 삽입 정렬을 이용하는 개선된 퀵 알고리즘[4]은 1962년에 발표된 퀵 정렬보다 25~30% 가량 성능의 개선이 이루어진 알고리즘이다.

개선된 퀵 정렬의 알고리즘은 <그림 2>와 같은 코드로 구현할 수 있다.

```
#define M 10
#define key(A) (A)
#define less(A,B) (key(A) < key(B))
#define exch(A,B) {Item t=A; A=B; B=t;}
#define compexch(A,B) if (less(B,A)) exch(A, B)

void quicksort ( Item a[], int l, int r )
{
  int i;
  if (r-l <= M) return;
  exch (a[(l+r)/2], a[r-1]);
  compexch (a[l], a[r-1]);
  compexch (a[l], a[r]);
  compexch (a[r-1], a[r]);
  i = partition (a, l+1, r-1);
  quicksort (a, l, i-1);
  quicksort (a, i+1, r);
}

void insertion ( Item a[], int l, int r )
{
  int i;
  for (i = l +1; i<=r;  i++) compexch (a[l], a[i]);
  for (i = l +2; i<=r;  i++)
  { int j = i; Item v = a[i];
    while (less(v, a[j-1]))
     { a[j] = a[j-1]; j--; }
    a[j] = v;
  }
}

void sort ( Item a[], int l, int r )
{
  quicksort (a, l, r);
  insertion (a, l, r);
}
```

(그림 2) 개선된 퀵 정렬 알고리즘

이 알고리즘은 분할 원소를 선택할 때 값이 중간인 원소를 선택하여, 가장 작은 값을 a[l], 가장 큰 값을 a[r], 중간 값을 a[r-1]에 넣고, a[l+1]에서 a[r-2]까지 분할 알고리즘을 수행한다. 그리고 서브 파일(subfille)의 크기인 M이 일정 크기 이하로 작아지면 삽입 정렬 수행하는데, M 값은 보통 5에서 25의 값을 설정한다.

이 알고리즘이 가지는 세 가지 장점으로서, 첫째, 비교회수가 N^2에 비례하는 최악의 경우가 발

생할 확률을 낮추어준다는 것, 둘째 분할을 위해 키를 사용할 필요가 없다는 것, 셋째 전체 수행 시간이 기존의 퀵 정렬에 비해 20~25% 정도 빠르다는 것을 들 수 있다.

이 세 가지 장점에도 불구하고, 이 알고리즘의 시간 복잡도는 이론적으로 퀵 정렬 알고리즘과 동일하다.

3. 정보블록 정렬 알고리즘

3.1 용어의 정의

다음은 본 연구에서 사용된 용어에 대한 정의와 설명이다.

① DB(Data Block) : 데이터 블록. 데이터 리스트를 M개로 나누었을 때 나누어진 각 부분을 말한다.

② IB(Information Block) : 정보블록. 데이터 블록에 관한 정보 즉, 데이터의 개수나 데이터의 범위, 그리고 데이터의 물리적 위치 등에 관한 정보가 저장되어있다.

③ BDR(Block Data Range) : 하나의 데이터 블록에 포함될 수 있는 데이터의 범위를 수치화한 값. 예를 들어 최소값인 MinDV가 1이고 BDR이 50이라면 첫 번째 정보블록에 포함될 수 있는 데이터는 1에서 50까지의 값을 가지는 데이터이다. ($1 \leq BDR$)

④ TDR(Total Data Range): 데이터리스트에 있는 모든 데이터가 속하는 범위를 수치화한 값.

⑤ M : 정보블록의 개수 ($M \leq N$). M은 TDR을 BDR로 나눈 몫이 되며, 나머지가 있으면 1 증가시킨다. M이 N보다 큰 경우에는 M에 N을 대입시킴으로써 극단적으로 크거나 작은 데이터로 인한 시간 복잡도의 증가를 배제시킨다.

⑥ BDP(Blank Data Position) : 데이터가 임시기억장소로 이동하여 비어있는 위치를 의미한다. BDP는 데이터블록 내에서의 상대적인 위치가 아니라 데이터 리스트에서의 절대적인 위치를 나타낸다.

3.2 자료구조와 메모리 사용량

정보블록은 모두 M개의 단위 정보블록으로 구성되는데, 단위정보블록의 구성요소는 <표 2>와 같다. 전처리 과정에 필요한 실제 메모리는 리스트의 크기에 정보블록의 크기를 더한 값이 된다.

<표 2> 단위 정보블록의 구성요소

요소	설명
Range_start(Rs)	DB에 속할 수 있는 가장 작은 값
Range_end(Re)	DB에 속할 수 있는 가장 큰 값
Number_data(Nd)	DB에 속하는 데이터의 개수
Pointer_start(Ps)	리스트상에서 DB가 시작되는 곳의 위치
Pointer_end(Pe)	리스트상에서 DB가 끝나는 곳의 위치
Now_Pointer(Np)	이동여부를 검사해야할 데이터의 위치

IBSA에서 이용되는 자료 구조는 데이터 리스트와 정보블록 이외에 DataInfo와 CopyData라는 두 가지 자료 구조가 있다. CopyData는 데이터 블록에 있는 데이터를 그대로 복사한 것으로서, 자료구조는 데이터블록과 동일하다. DataInfo는 각각의 데이터 블록을 정렬하기 위하여 필요한 자료구조로서, 데이터블록에 대한 정보를 저장하고 있다. DataInfo의 구성요소는 <표 3>과 같다.

<표 3> DataInfo의 구성요소

구성요소	설명
Value(V)	데이터의 값
Data_Count(DC)	Value 값을 가진 데이터의 개수
Accumulated_ Position(AP)	누적 시작 위치. CopyData에서 이동되어야할 데이터의 기준위치값을 의미한다.
Sent_Count(SC)	CopyData에서 데이터블록으로 데이터가 정렬되면서 이동된 'Value'값의 데이터 개수

데이터를 정렬하기 위해서 필요한 메모리는 전처리 과정에 필요한 메모리에 DataInfo와 CopyData에 할당된 메모리를 더한 값이다.

```
필요한 메모리 = 리스트의 크기 + 정보블록의 크기
                + DataInfo의 크기 + CopyData의 크기
정보블록의 크기 = M * 단위정보블록의 크기
단위정보블록의 크기 = 6 × sizeof(요소의 데이터형)
```

3.3 알고리즘

3.3.1 논리적 구조

IBSA의 논리적 구조는 (그림 3)과 같이, 세 부분으로 구성되어 있다. 첫 번째 부분은 전처리

과정으로서 정보블록을 초기화하고 재구성하는 부분이며, 두 번째 부분은 데이터를 재배치하는 부분, 세 번째 부분은 각각의 데이터 블록을 정렬하는 부분이다.

```
Part 1. IB 초기화 및 재구성
  Step 1. TDR, BDR, M 설정 및 IB 할당
  Step 2. IB의 초기화
  Step 3. 데이터의 분포 파악
  Step 4. IB의 정보 재구성

Part 2. 데이터 재배치
  Step 5. 블록별 데이터 이동

Part 3. 정렬 : M만큼 반복
  Step 6. TDataInfo 메모리 할당 및 초기화
  Step 7. TCopyData 메모리 할당 및 복사
  Step 8. 원본인 Data 배열로의 데이터 이동
  Step 9. 동적 메모리 할당 해제
```

(그림 3) 정보블록 정렬 알고리즘

3.3.2 시간 복잡도

Part.1의 비교회수는 다음과 같으므로 시간 복잡도는 $O(N)$과 같다.

Part. 1의 비교회수
= Step.1에서 Step.4까지의 비교회수
= $C_1N+C_2M+C_3N+C_4M$ $(1\leq M\leq N)$
$\in \Theta(N)$

Part.2는 데이터를 재배치하는 부분으로서 상세한 알고리즘은 다음의 (그림 4)와 같다.

```
for I ←1  To  M ···①
do for j ←i번째 Np to I번째 Pe···②
  do Temp ←data[j]
     BDP ←j
     while 블록순회가 계속인 동안···③
     do ▷ 해당 블록 찾기 ···④
        ▷ 데이터 이동
        For k ←해당 블록의 Np
            to  해당 블록의 Pe···⑤
        do if BDP = k ···⑥
           then data[k] ← Temp
                해당 블록의 Np 1증가
                ⑤와 ③ 루프 탈출, ②로 복귀
           if data[k]가 해당 블록에 속하는가?
           then 해당 블록의 Np 1증가···⑦
           else Temp와 data[k]의 교환···⑧
                해당 블록의 Np 1증가
                ⑤ 루프 탈출, ③으로 복귀
```

(그림 4) 데이터 재배치 알고리즘

Im + Gm + Em은 N으로서, Part.2의 시간 복잡도가 $O(N)$이라는 것을 의미한다. Im은 ⑥에서처럼 BDP가 가리키는 곳으로 데이터의 삽입이 이루어지는 회수이고, Gm은 Am번째 DB에서 ⑦로 분기하는 회수이며, Em은 Am번째 DB에서 ⑧에서 데이터 교환이 발생하는 회수를 나타낸다. 가장 많은 비교가 소요되는 Em이 N만큼 발생한 경우 C_8N만큼 비교되며, 가장 적은 비교가 소요되는 Gm의 경우 C_7N만큼 비교된다. 따라서 Part.2의 비교회수는 다음과 같이 시간 복잡도가 $O(N)$이 된다.

Part 2의 비교회수
$= (C_5+C_6I_1+C_7G_1+C_8E_1)$
$+ \cdots$
$+ (C_5+C_6I_m+C_7G_m+C_8E_m)$
$= C_5M+(C_6\sum_{i=1}^{m}I_i+C_7\sum_{i=1}^{m}G_i+C_8\sum_{i=1}^{m}E_i)$
$= C_5M+(C_9N)$
$\in \Theta(N)$ (I+G+E=N)

Part.3의 비교회수는 아래와 같이 나타낼 수 있으며, 시간 복잡도는 $O(N)$과 같다.

Part.3의 비교회수
= Step.1에서 Step.5까지의 비교회수
= $C_1M+C_2M+C_3N+C_4$
$(1\leq M\leq N)$
$\in \Theta(N)$

위의 결과를 종합해보면, IBSA의 시간 복잡도를 <표 4>와 같이 나타낼 수 있다.

<표 4> IBSA의 시간 복잡도

경우	시간 복잡도	설 명
최적 (Best)	O(N)	데이터 블록 간의 데이터 이동이 거의 없는 경우
평균 (Average)	O(NlogN)	O(N)+O((N-G-I)log(N)+N)+O(N) ∈ O(N + NlogN + N + N) ∈ O(NlogN)
최악 (Worst)		

IBSA가 최적일 경우는 데이터 블록 간의 데이터 이동이 거의 없는 경우이다. 나머지 경우, IBSA는 데이터의 분포상태에 영향을 받지 않으므로 O(NlogN)의 시간 복잡도를 가진다.

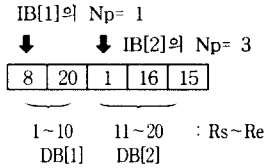
3.3.3 예제

정렬하고자하는 데이터는 8, 20, 1, 16, 15이며, BDR은 10이라고 가정하자.

(1) 전처리 과정

이 때, N은 5, MinDV가 1, MaxDV가 20, TDR은 20-1+1 즉, 20임을 알 수 있다. 그러므로 M은 TDR을 BDR로 나눈 몫으로 2가 된다. 모든 데이터의 분포를 파악해보면 1~10사이에 있는 데이터가 2개, 11~20사이에 3개가 분포한다. 따라서 IB[1]의 Nd는 2, IB[2]의 Nd는 3으로 설정된다. 또한 IB[1]의 Ps와 Pe는 각각 1과 2, IB[2]의 Ps와 Pe는 각각 3과 5로 설정된다. IB[1]의 Rs는 1, Re는 10이 되며, IB[2]의 Rs는 11, Re는 20이 된다.

1) 리스트의 초기 상태는 다음과 같으며, BDP(▲)는 Temp로 이동한 데이터가 없으므로 어떤 위치도 가리키지 않는다.

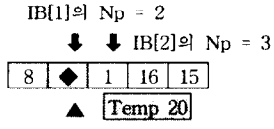


2) IB[1]의 Np만 증가

현재 검사중인 DB[1]의 Np가 가리키는 데이터 '8'은 DB[1]에 있어야 할 데이터이므로, IB[1]의 Np만 1증가시켜 2로 설정한다. 이처럼 단지 Np만 증가되는 경우에는 데이터 이동이 발생하지 않으므로, 이런 경우의 발생회수가 많을수록 실행시간과 비교회수는 감소하게 된다. 만약 모든 데이터가 데이터 블록을 벗어나지 않는다면 시간복잡도는 최적으로서 O(N)을 가지게 된다.

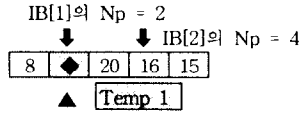
3) DB[1]의 '20'을 Temp로 이동시키기

DB[1]의 Np가 가리키는 데이터인 '20'은 IB[1]의 Rs와 Re값 사이에 속하지 않으므로 다른 DB로 옮겨야 한다. '20'을 임시기억장소인 Temp로 옮긴 후 '20'이 있던 위치인 2를 BDP에 넣는다.



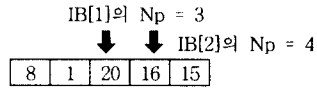
4) '20'과 '1'의 데이터 교환

'1'이 위치해야할 DB는 DB[2]이다. IB[2]의 Np가 가리키는 데이터 '1'은 DB[1]로 옮겨야 할 데이터이므로 Temp와 교환한다. 알맞은 DB에 위치한 데이터 '20'은 차후에 다시 검사할 필요가 없기 때문에 IB[2]의 Np는 1증가시켜 4로 설정한다.



5) '1' 삽입

Temp에 있는 데이터 '1'이 위치해야할 DB는 DB[1]이다. IB[1]의 Np가 가리키는 곳은 BDP가 가리키는 곳이므로 비교할 필요 없이 '1'을 이동시킨다. IB[1]의 Np는 1증가되어 3이 된다. 현재 Temp에 보관된 데이터는 이동 후 비어있으므로 BDP 역시 값이 없다.

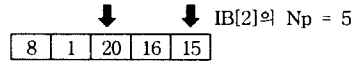


6) DB[1] 재배치 완료

IB[1]의 Np인 3은 Re보다 크므로 DB[1]의 재배치작업은 끝나고 다음 DB 즉, DB[2]의 재배치작업이 시작된다.

7) IB[2]의 Np만 증가

IB[2]의 Ps는 3이지만 IB[2]의 Np는 '4'이므로 Np가 가리키는 데이터인 '16'부터 재배치작업이 시작된다. '16'은 DB[2]에 있어야 할 이동할 필요가 없는 데이터이므로, IB[2]의 Np만 1증가시켜 5로 설정한다.



8) DB[2]의 재배치 완료

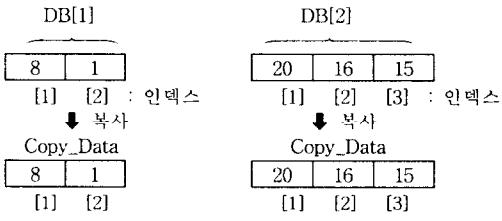
현재 검사 중인 IB[2]의 Np가 가리키는 데이터인 '15'는 DB[2]에 있어야 할 데이터이므로, IB[2]의 Np만 1증가시켜 6으로 설정한다. IB[2]의 Np는 6이고 Pe는 5이므로 더 이상 재배치해야할 데이터는 없다. 이로써 전처리 과정은 모두 끝나게 된다.

(2) 각 블록 정렬

전처리에 이어 각 블록을 정렬한다.

1) 정보 블록의 데이터 복사

DB[1]에 있던 데이터가 동적으로 할당된 Copy_Data로 복사되었다.



2) DataInfo 초기화

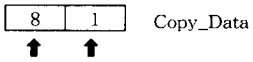
동적 할당된 DataInfo의 구성요소가 초기화된 상태는 다음과 같다.

V	DC	AP	SC
1	1	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	1	0	0
9	0	0	0
10	0	0	0

V	DC	AP	SC
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	1	0	0
16	1	0	0
17	0	0	0
18	1	0	0
19	0	0	0
20	1	0	0

3) 누적 위치 계산

SC는 데이터 리스트로 이동된 데이터가 없으므로 0으로 초기화된다.



다음은 누적 시작 위치 정보가 입력된 DataInfo의 정보를 나타낸 것이다.

V	DC	AP	SC
1	1	1	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	1	2	0
9	0	0	0
10	0	0	0

V	DC	AP	SC
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	1	3	0
16	1	4	0
17	0	0	0
18	1	0	0
19	0	0	0
20	1	5	0

4) 데이터 블록으로 이동

이동될 위치는 누적시작위치에 삽입된 개수를 더한 값이다. 데이터의 이동이 끝난 후의 DataInfo는 다음 그림과 같으며, 데이터 이동이 완료되면 각 블록은 정렬이 완료된 상태가 된다.

V	DC	AP	SC
1	1	1	1
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	1	2	1
9	0	0	0
10	0	0	0

V	DC	AP	SC
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	1	3	1
16	1	4	1
17	0	0	0
18	1	0	0
19	0	0	0
20	1	5	1

4. 성능 평가

4.1 실험 조건

IBSA의 성능을 측정하기 위한 실험조건은 다음과 같다.

① 실험 시스템의 사양은 <표 5>와 같다.

<표 5> 실험 시스템 사양

구분	장 치	사 양
H/W	CPU	Intel P-IV 1.7G
	RAM	512MB
S/W	NOS	Windows 2K Server
	DBMS	SQL Server 7.0
	Dev Tool	Delphi 7.0

② 실험데이터는 시각을 기초로 생성되어 균일 분포를 보이는 난수 그리고 가우스 함수를 이용해 정규분포를 보이는 난수, 이 두 가지 데이터를 이용한다.

③ 비교알고리즘은 퀵 정렬과 개선된 퀵 정렬 알고리즘을 이용한다.

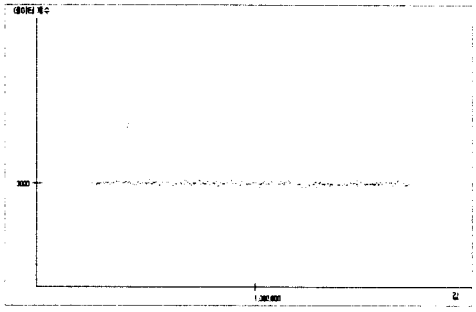
④ 비교회수에 있어서, 'if'와 'while' 그리고 'for'와 같은 명령문에서도 실제로는 비교가 발생하므로 정확한 비교회수를 측정하기 위해 이러한 명령문 역시 비교회수에 누적시켰다.

⑤ 실험 데이터의 개수는 2백만개 데이터까지 실험하였으며, IBSA의 BDR은 초기값으로 50으로 설정하였다.

4.2 실험 결과

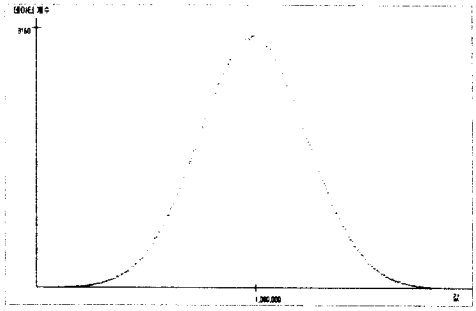
4.2.1 데이터 분포

첫 번째 실험 데이터는 균등한 분포를 띄는 데이터로서, 이 데이터를 5000이라는 범위로 그룹을 묶어 데이터의 분포를 (그림 5)에 나타내었다.



(그림 5) 데이터의 균등 분포

두 번째 실험 데이터는 가우스 함수를 이용해 생성된 정규 분포를 나타내는 데이터로서, 이 데이터를 5000이라는 범위로 그룹을 묶어 데이터의 분포를 (그림 6)에 나타내었다.



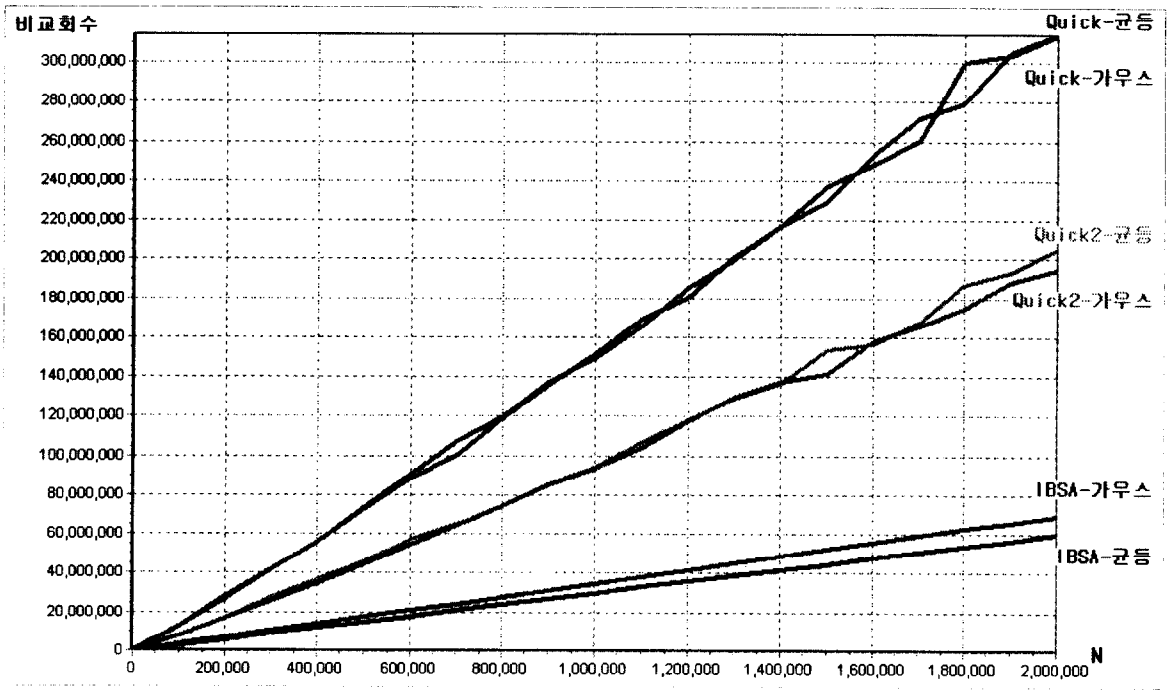
(그림 6) 데이터의 정규 분포

4.2.2 실험 결과

실험결과는 실제 비교회수를 10만개 단위로 (그림 7)에 나타내었다.

실험 결과를 분석해보면 다음과 같다.

첫째, IBSA는 정규 분포를 보이는 데이터의 정렬뿐만 아니라 균등 분포를 보이는 데이터의 정렬에서도 퀵 정렬과 개선된 퀵 정렬 알고리즘



(그림 7) 비교 회수

에 비해 상당히 효과적인 알고리즘이라는 점이다. 비교회수 면에서 IBSA는 퀵 정렬의 22%, 개선된 퀵 정렬의 36% 정도의 비교회수만으로 정렬을 완료할 수 있었다. 그리고 개선된 퀵 정렬 알고리즘은 퀵 정렬 알고리즘에 비해 40% 가량 성능이 우수한 것으로 나타났다.

둘째, 데이터가 특정 범위에 밀집되는 가우스 분포를 보이는 경우에도 IBSA의 성능 저하는 없다는 점이다. 이 점은 IBSA의 최악과 평균 시간 복잡도를 대변해주는 것으로서, 정렬의 안정성을 보여주고 있다.

5. 결론 및 향후 연구과제

본 연구에서는 $O(NlogN)$의 시간 복잡도와 데이터의 분포상태에 영향을 받지 않는 정보블록 정렬알고리즘을 제안하고, 그 성능을 평가하였다.

IBSA의 성능을 측정해본 결과, 2백만 개의 랜덤데이터를 정렬한 경우, 퀵 정렬의 약 20% 정도의 비교회수만으로도 정렬할 수 있음을 보여주었다.

앞으로 연구해야할 과제라고 할 수 있는 것은 크게 두 가지로 볼 수 있다.

첫째로는, IBSA의 성능을 높이기 위하여 알고리즘의 핵심요소인 블록 데이터 범위(BDR)와 정보 블록의 개수(M)의 최적화에 대한 연구이며, 둘째로는, IBSA 역시 분할 정렬 알고리즘이기 때문에 외부 정렬 기법과의 연동 그리고 멀티프로세서를 이용하는 병렬처리시스템의 정렬에서도 유용할 것으로 생각되므로 이에 관한 연구를 하는 것이다.

참 고 문 헌

[1] 도경구 역(1999). 알고리즘. 사이텍 미디어.

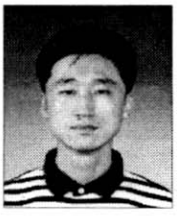
[2] 송태옥, 송기상(2000). 정렬알고리즘의 성능 향상을 위한 정보블록 전처리 알고리즘. 한국정보과학회 2000년도 추계학술논문집.

[3] Hoare, C.A.R(1962). Quick Sort. Computer Journal 5(1), pp.10-15.

[4] Robert Sedgewick(1998). Algorithms in C, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching, Third Edition. Addison Wesley.

[5] Thomas A.S(1994). Data Structures, Algorithms & Software Principles in C. Addison Wesley.

[6] Thomas H.C. et al(1989). Introduction to Algorithms. The MIT Press.



송 태 옥

1991 부산교육대학교 교육학과 (교육학학사)

1998 한국교원대학교 컴퓨터교육과 (교육학석사)

2001. 2 한국교원대학교 컴퓨터교육과 (교육학박사)

2001. 3~ 2002. 2 한국교원대학교 컴퓨터교육과 Post-Doc 및 BK21 연구교수

2002. 3~현재 관동대학교 컴퓨터교육과 조교수

관심분야: 컴퓨터교육, 인성교육 및 정보통신윤리교육, 알고리즘, 네트웍 프로그래밍.

E-Mail: kinggem@kwandong.ac.kr