MULTIKEY SORTING FOR SEQUENTIAL FILES

P.Abraham Moses
BICS info Tech
Yellagiri Hills, Tirupattur, India,
Email: mosesatbics@yahoo.com

P.Radhakrishnan
Faculty of Information Science and Technology
MultiMedia university, Bukit Beruang, Melaka,75450, Malaysia
Email: radha.krishnan@mmu.edu.my

Abstract
"Multikey Sorting" is arranging a sequential file according to ascending or descending on more than one keys or one columns. This paper considers a modified version of Binary Insertion Tree to sort records on several keys with the possibility of mixing the order of keys. I.e. ascending on some keys and descending on some keys. Existing techniques consider sorting directly on all the keys either in ascending or descending order. The proposed method overcomes this limitations, and use dynamic variables to allow sorting of files with sizes comparable to the memory available to the user. It achieves an average number of comparisons comparable to the best known sorting algorithms, since inserting N records into an initially empty tree needs the same average number of comparisons between records. This paper compares its execution with Exchange Sort, Selection Sort and Quick Sort and shows a big difference in speed, execution and storage. The algorithms for multikey sorting by Exchange sort technique, Selection sort technique, Quick sort technique and the proposed method are studied and implemented in Pascal Language.

Introduction
A file is a collection of records, each record having one or more fields. The fields used to distinguish among the records are known as "Keys". Consider the problem of sorting [1] a number of keys on r different keys $(K^1 K^2 K^3 \ldots K^r)$ where, K is the least significant key. A file of records $R^1, R^2, \ldots R^n$ is said to be sorted with respect to the keys $(K^1 K^2 K^3 \ldots K^r)$ if and only if for every pair of record i,j $(K^1 K^2 K^3 \ldots K^r) <= (K^1_j, K^2_j \ldots K^r_j)$ *where* $I <= 1$.

There are two ways to accomplish the multikey sort [2,3]. Which are Sort on most significant field (MSF) and Sort on Least Significant Field (LSF).

"Most Significant Field" is to sort on the key K^1 obtaining several "plies" of records each having the same value for K^1. Then each of these piles is independently sorted on the key K^2. These subpiles are then sorted on K^3 and etc. "Least Significant Field" is to sort the records to make several "Piles" by considering all the keys. Then, place the higher value piles o the lower value piles and make bottom to top values for the sorted file.

Binary Insertion Trees

A binary tree is a tree in which every node is a record composed of data elements and two pointers, each pointing to a node or to nil. Any binary tree can be decomposed into a left subtree and a right subtree. If the binary tree is ordered, then the records to the left and below of any nodes are less than that node, while the records to the right and below of any node are greater than that node.

Representing Binary Trees in Memory

A binary tree is a tree in which every node is a record composed of data elements and two pointers, each pointing to a node or to nil. Any binary tree can be decomposed into a left subtree and a right subtree. If the binary tree is ordered, then the records to the left and below of any node are less than that node, while the records to the right and below of any node are greater than that node. Here we consider a tree with three pointers coming out of every node, namely Left, Right and Equal. The Left link is to mention the location of the smaller node, Right link is to mention the location of the bigger node and Equal link will mention the location of the node where the values are equal.

Modified Binary Tree

A binary tree is a tree in which every node is a record composed of data elements and two pointers, each pointing to a node or to nil. Any binary tree can be decomposed into a left subtree and a right subtree. If the binary tree is ordered, then the records to the left and below of the nodes are less than that node, while the records to the right and below of any node are greater than that nodes. Here we consider a tree with three pointers coming out of every node, namely Left, Right and Equal. The Left link is to mention the location of the smaller node, Right link is to mention the location of the bigger node and Equal link will mention the location of the node where the values are equal.

Multikey Sorting with Selection Sort Technique

Suppose a file F contains N records $R^1, R^2, R^3 \ldots R^n$ on r different keys $K^1 K^2 K^3 \ldots$.where K^1 is the least significant key. The selection sort for multikey [4] works as follows. Find the smallest value in the file on the first key. Make the row as the first row. Search the second smallest value in the file on the first key. Make it as the second row. Repeat the process for all the records. Now consider the second key. Search [5,6] the smallest value in the file on the second key. Compare the value with the first key. If the values are equal, make it as the first key's value. If the values are same, make it as the second row. Repeat the process for all the records and on all the keys. This process will sort the entire file on all the keys.

Approach in Steps

1. Find the location of the smallest value in the list of N records $R^1, R^2, R^3 \ldots R^n$ on the first key and interchange with the first now.
2. Find the next smallest value of the list of (N-1) records on the first key and interchange with the second record.
3. Repeat the process for all the records so that all the records on the first key are arranged into order.
4. Consider the next key.
5. Find the location of the smallest value in the file on the current key.
6. Compare the smallest value to the same row and the previous key.

7. If the values are equal, then interchange the records. Otherwise move to the next record.
8. If the values are equal, then interchange the records. Otherwise move to the next record.
9. Repeat the process for all the records on the current key.
10. Repeat Steps (5) through Steps (8) until all the keys are processed.

Algorithm
Interchange(R,K,J)

1. *X = R[j]*
2. *R[j] = R[c]*
3. *R[c]=X*
4. *Return*

Minimum (R,K,N,LOC)

1. *SET Min = R[K]*
2. *SET LOC = IK*
3. *Repeat Steps (4) for J = k+1,k+2,…..N*
4. *SET MIN = R[j]*
5. *SET LOC = R[j]*
6. *Return*

Selection Sort(R,N)

1. *Repeat Steps(2) for K = 1,2,…N-1*
2. *Repeat steps for I = 1 to maximum number of keys*
3. *CALL Minimum(R,K,N,LOC)*
4. *If Minimum > $R^k[J]$ then*
5. *If K <> 1 then*
6. *If $R^k[j] = R^{k-1}[j]$ then*
7. *CALL Interchange(R,K,J)*
8. *Exit*

Efficiency of Algorithm (Selection Sort)

First note that the number *F(n)* of comparisons in the selection sort algorithm is independent of the original order of the elements. Observe the MINIMUM (A,K,M,LOC) requires *N-K* comparisons during the first stage to find the smallest element on the first key. There are *(N-2)* comparisons for the finding of the second value for the first key. So, for sorting the file on the first key, the order of algorithm is $F(n) = (n-1)+(n-2)+(n-3) + \ldots\ldots\ldots + 2+1 = n(n-1)/2 = O(n^2)$
In this algorithm, the following are the efficiency of algorithm. On worst Case it is $O(n^2)$
On Average Case the efficiency is $O(n^2)$,On best Case it is $O(n^2)$

Multikey Sorting with Bubble Sort Technique

Suppose a file has N records $R_1, R_2, R_3 \ldots R_N$ with R keys. Sort the file with R keys, the bubble sort works as follows. Consider the first key. Compare the first two records on the first key. Arrange them in the desired order. Consider the second and third and first key. Arrange according to the desired order. Repeat this for N-1 times so that the largest or the least values moved to the very last record.

Consider the second key. Compare the first and the second keys. If the records are not arranged, change the corresponding values in the first key. If they are equal, then interchange otherwise move to the next pair. Repeat the process for all the records. Then, do the above process for all the keys. The file will be sorted on all the keys.

Approach in steps

- Consider the first key
- Compare *R(1)* and *R(2)* and arrange them in the desired order.
- Compare the next pair of records and arrange them in the desired order.
- Repeat Step (3) until all the pairs of records are compared.
- Consider the next key
- Compare *R(1)* and *R(2)* on the current key. If it needs to be arranged in the desired order, Compare $R^{K-1}[J]$ with $R^{K}[J]$. If the values are equal, then arrange them in the desired order.
- Repeat Step (5) and Step (6) until all the keys are considered.

Algorithm

1. *Repeat Step (2) for K1 to R keys*
2. *Repeat Step(3) & Step (4) for I = 1 to N records*
3. *Set PRT = 1*
4. *Repeat Step while PTR <= N-1*
5. *If $R^{K}[PTR]$ > Data[PTR+1] then*
6. *Call INTERCHANGE(R,K,PTR,PTR+1)*
7. *Set PTR = PTR + 1*
8. *Return*

Interchange (Bubble Sort)

1. *Interchange (R,K,PTR,PTR+1)*
2. *Set PTEMP = $R^{K}[PTR+1]$*
3. *Set $R^{K}[PTR] = R^{K}[PTR+1]$*
4. *Set $R^{K}[PTR+1]$ = PTEMP*
5. *Return*

Efficiency of Algorithm

The number *f(n)* of comparisons in the bubble sort is computed as follows. Specifically, there are *(n-1)* comparisons during the first phase, which places the largest element in the last position. There are*(n-2)* comparisons in the second step. Which places the second largest element in the next to last position and so on. Thus $F(n) = (n-1)+(n-2)+(n-3) + \ldots\ldots\ldots + 2+1 = n(n-1)/2 = O(n^2)$

In other words, the time required to execute the bubble sort algorithm in proportional to n^2 where n is the number of input items.

Multikey Sorting with Quick Sort Technique

Consider a file with N records $R^1, R^2, R^3 \ldots R^n$ on r different keys $K^1 K^2 \ldots K^r$. As Quick sort is different from all other sorting techniques, the key K^i controls the process the right spot with respect to

the whole file. Thus if the key K^i is placed in a sub file *S(1)*, then $K^i <= KS(i)$ for $I < S(i)$. Hence after this positioning completed, the original file is partitioned into two sub files, one consisting of records $R^1 ... R^{s(1)-1}$ and the other records $R^{s(1)+1} ... R^N$, since in the sorted sequence all the records in the first sub file may appear to the left of *S(i)*, and all the second sub file to the right *S(i)*. These two sub files may be sorted independently. Again, the second key is considered. If the key K^i is placed in the sub file *S(i)* on the second key, then $K^1 > KS(i)$ for $j > S(i)$ only if $K^1 >= KS(i)$ is true on the previous key. After positioning is made, the original file is partitioned into two sub files. Once consisting of $R^1 .. Rs^{(i)-1}$ and the other records $R^{s(i)+1} ... R^N$ on the second key. The first sub file may appear to the left of *S(i)*.

Approach in steps

1. Consider the first and last record on the first key.
2. Move to the next record until the first record value is greater than the last record value.
3. Consider the last record on the first key
4. Move to the previous record until the current record's value is greater than the first record's value.
5. Interchange the lastly positioned records after processing Step(2) and
6. Split the file into two sub files according to the positioning of the file after completing the execution of Step 2 and Step 4.
7. Process the above statements independently to two sub files.
8. Stop the processing when both the sub files are empty.
9. Consider the next key
10. Consider the first and last records of the file on the second key.
11. Move to the next record until the current record's value is greater than the first record's value as well as first records value where the first key is equal to the second key's first record.
12. Move to the previous record until the current record's value is greater than the last record's value, as well as the current record's value on the first key and is equal to the current record's value on the second key.
13. Interchange the row ranges into two sub files and process the step (9) through step 14.
14. Split the two ranges into two sub files and process the step 9 through step 14 repeatedly until both sub files are empty.
15. Repeat Step 9 through Step 14 for all the keys

Interchange (R[1],R[j])

1. *T = R[I]*
2. *R[I] = R[j]*
3. *R[j] = T*
4. *Return*

Quick Sort (R,M,N)

1. *If M < N then*
2. *Set I = M*
3. *Set J = N+1*
4. *K = R[M]*
5. *Repeat Step 6 until I >= j*
6. *Repeat Step 7 until R[j] >= K*
7. *Repeat Step 8 until R[j] <= K*

8. *If I < J then interchange (R[I],R[j])*
9. *Goto Step 5*
10. *Interchange (R[M],R[j])*
11. *Call Quick Sort(R,M,j-1)*
12. *Call Quick Sort(R,J+1,N)*
13. *Return*

Efficiency of Algorithm

The worst case behavior of this algorithm is to compare all the records and move the pointer right through the last record. So, it needs the following number of Comparisons.

$$F(n) = (n-1)+(n-2)+(n-3) + \ldots\ldots\ldots + 2+1 = n(n-1)/2 = O(n^2)$$

The average case of behavior of this algorithm provides to make atleast two sub files on the first attempt and the number of comparisons is reduced to half of the previous comparisons. The time required to position the record in a file of size *n* is *O(N).* If *T(N)* is the time taken to sort a file of *N* records, then when the file splits roughly into two equal parts, each time a record is positioned correctly, then we have

$T(N) <= CN + 2T(N/2)$ *for some constant C*

$<= 2CN + 4T(N/4) <= CnLog_2N + NT(1) = O(N \log_2 N)$

So, on the average case, the order of the quick sort algorithm is $O(N \log_2 N)$

Multikey Sorting with Modified Binary Insertion Trees Technique

Considered the problem for sorting N records $(R_1 R_2 R_3 \ldots R_N)$ on *R* different keys $K^1 K^2 \ldots K^R$ Where K^1 is the most significant key and *KR* is the least significant key. A file can be represented by a binary tree in, which every node is a record composed of data elements and tow pointers each pointing to a node or to *N. J* any binary tree can be decomposed into a left subtree and tight subtree. If the binary subtree is ordered, then the records to the left and below of any node are less than that node, which the records to the right and below of any node are greater than that node. This technique considers a tree with three pointers coming out of every node namely left (for smaller), tight (for greater) and equal. The binary insertion technique has to be modified accommodate for their change and to allow for sorting on R keys. The implementation of this is the recursive procedure INSERT, which uses the procedure COMPARE to compare the appropriate keys and returns and integer variable result. The returned value result will be

$$Result = 0 \text{ if } K_{inew} = K_{l\ current}$$
$$1 \text{ if } K_{inew} = K_{1\ current}$$
$$-1 \text{ if } K_{inew} = K_{1\ current}$$

A function CREATE-NEW-NODE-PTR created a new code, assigns values to the components and returns a pointer to the node. Procedure PRINT displays the key components of the nodes and a sequence number, which gives the rank of the node in the tree. The advantage of this implementation is that the file can be sorted on the keys $K^1 K^2 \ldots K^R$ such that ordering might be ascending on one key while descending on another. For example, records might be sorted in ascending order on K^1, within K^1 , In descending order of *K*, and within K^2 in ascending order of K^3 and so on. This requires changes in the values assigned to result for key = 2 as follows.

$$Result = 0 \text{ if } K_{inew} = K_{l\ current}$$

1 *if* $K_{inew} = K_{1\ current}$

-1 *if* $K_{inew} = K_{1\ current}$

If the key is two then result = - result, and process it recursively so that the ordering of K_2 can be descending on k_1. The same process can be repeated for $K_2 K_3$ and so on.

Approach (Modified Binary Tree)

Step 1 : If root node does not exist, then set root = new and go to Step (7). Else, start from the root node by setting Current = root and set l to 1 where l is for the key number.

Step 2 : If $K_{inew} < K_{1\ current}$ then set result = - 1, go to Step (3) if $K_{inew} > K_{1\ current}$, then go to Step (4). If $K_{inew} = K_{1\ current}$, then set result = 0 and go to step (5).

Step 3 : If left subtree is not empty then set current to left link and goto Step (2). Otherwise go to Step(6).

Step 4 : If right subtree is not empty then set current to right link and goto Setp (2). Otherwise go to Step (6).

Step 5 : If equal subtree is not empty then set current to right links and increment i by one to perform subsequent comparisons needed to insert this record in the tree on the next key *Ki*, and go to Step (2). Otherwise make a new subtree with root node equal to current node, connect equal link of current node to the new subtree and go to step (5).

Step 6 : If result = -1, set left link to new, go to step (7). If result = 1 set right link to new , go to step(7).

Step 7 : If other nodes are to be inserted to Step (1). Otherwise Stop.

Algorithm (Modified Binary Tree)

1. *If new-node-ptr = current then*
   Set result = 0 else
2. *If new-node-ptr > i then*
   Set result + = -1
3. *Set result = - 1*
4. *Return*

Create Node()

1. *New (Node)*
2. *Set Right link of New to Nil*
3. *Set Left link of New to Nil*
4. *Set Equal link of New to Nil*
5. *Return.*

Insert (key, root, ptr, New-node-ptr)

1. *If key = 1 then*
2. *If Root = Nil then*
   Set Root = New-node-ptr
3. *Else Set Current = root*
4. *else If ptr.equal = nil then*
   set current = ptr. Equal
5. *Repeat Steps (6) while Compare = 0*
6. *Result = Compare (Key, current, new-node-ptr)*
7. *If result = 0 then*

8. *Key = Key + 1*
9. *Insert (Key, root, current, new-node-ptr)*
10. *If result = - 1 then*
11. *Current. Left = new.node-ptr*
12. *If result = 1 then set current.right = new-node-ptr*
13. *If result = 1 then set current.right = new-node-ptr*
14. *Return*

Interchange (R_1, R_2)

1. *Set z = 1*
2. *Repeat steps (3), Steps (4) until >< Ki*
3. *If $R_1^z = R_2^z$ then $R = R_1$, $R_1 = R_2$, $R_2 = R_1$*
4. *Z = z + 1*
5. *Return*

Print (tree i)

1. *If tree <> Nil then*
2. *Print (Tree . Let, I)*
3. *If tree. Equal = Nil then*
4. *i = i + 1*
5. *else (3) Print (tree.Equal, i)*
6. *else (1) Print (tree. Right. i)*
7. *Return*

Efficiency of Algorithm (Modified Binary Tree)[Sort on one key]

Worst case : Since there are N records, and sorting on the first key compares n-1 times the equation forms as follows for the worst key. $f(n) = (n-1) + (n-2) + \ldots\ldots 2+1 = n(n+1)/2 = O(n^2)$

Average case : Suppose $T(N)$ is the time taken to search N records, then

$T(N) \leq (N+2T (N / 2)$ for some constant C

$\leq (N + 2) (N / 2 + 2. T (N / 4) \leq (N \log_2 N + NTC1) = O (N \log N)$

Modified Binary Tree[Sort on two keys : k^1 and k^2]

Assume m different values for k^1

i) Worst Case:

Starting from root, the largest subtree will be length m. For any tree with equal K^1, the largest length of such tree will be m / n . Therefore, searching for one record will take $m = m / n$ comparisons. Searching for N records required $O (N + M / N)$ Comparisons.

$f(n) = O (N (M+M / N) = O (N^2 / M) = O (N^2)$

ii)Average Case:

Starting from root, the length of the subtree will be $2 \log m$ and for any tree with equal $K1$, if balanced is $2 \log (N/M)$. Therefore, searching for one record will take an average $2 \log M + 2 \log N / M$. ie,

$f(N) = 2 \log N / M = 2 \log N$ Searching of N records requires

$= O (N(2\log N)) = O (N \log N)$

Sort on R keys (Modified Binary Tree)
Assume m different values for N^{i} $\prod_{i=1}^{r} m_i = N$

- Worst Case:If $m_I = 1$ for all $I = j$, then $m_1 = N$ which needs $O(N(m_1 + m_2 + \ldots + m_r)) = O(N(N+r-1)) = O(N^2)$ Comparisons
- Average Case: $O(N(2 \log m\,1 + 2 \log m2 + \ldots 2 \log m_1) = O(N \log N)$
  Statistical Analysis of Modified Binary Insertion tree method (25 keys)

Conclusion

A modified version of Binary Insertion Tree has been discussed to sort records on several keys with the possibility of mixing the order of keys. (ie) ascending on some keys and descending on some keys. Existing techniques consider sorting directly on all the keys either in ascending or descending order. It overcomes this limitations, and use dynamic variables to allow sorting of files with sizes comparable to the memory available to the user. It achieves an average number of comparisons [Table.1] comparable to the best known sorting algorithms, since inserting N records into an initially empty tree needs the same average number of comparisons between records. It is better than Exchange Sort, Selection Sort and Quick Sort and shows a big difference in speed, execution and storage.

Table 1. Comparison of Multikey Sorting

Method	**25 Records**	**50 Records**	**75 Records**	**100 Records**
Modified Binary Insertion Tree	164	640	1453	2704
Bubble Sort	6694	33889	94868	203691
Quick Sort	902	2760	5492	8920
Selection Sort	15000	76250	215000	462500

References

1. Stephen Alstrup, gerth Strlting Brodel, Theis Rauhe, "New Data Structures for orthogonal range searching" IEEE symposium on foundations of computer Science (2000)
2. C. C. Chang, D. F. Leu: Multikey Sorting as a File Organization Scheme When Queries Are Not Equally Likely. DASFAA 1989: 219-224
3. AbdelAziz Fellah , Distributed Database for Multikey Sorting, International Journal of Parallel and Distributed System and Networks, 1999.
4. Mahmoud F. Abaza , Multilevel Sorting in Distributed Systems, PACRIM, 1997.
5. Jon L.Bentley, Robert Sedgewick, "Fast Algorithms for sorting and searching strings" 1998.
6. Ehrlich, G. 1981. Searching and sorting real numbers. Journal of Algorithms 2, 1, 1-12. Kingston, J. H. 1990. Algorithms and Data Structures: Design, Correctness, Analysis.