

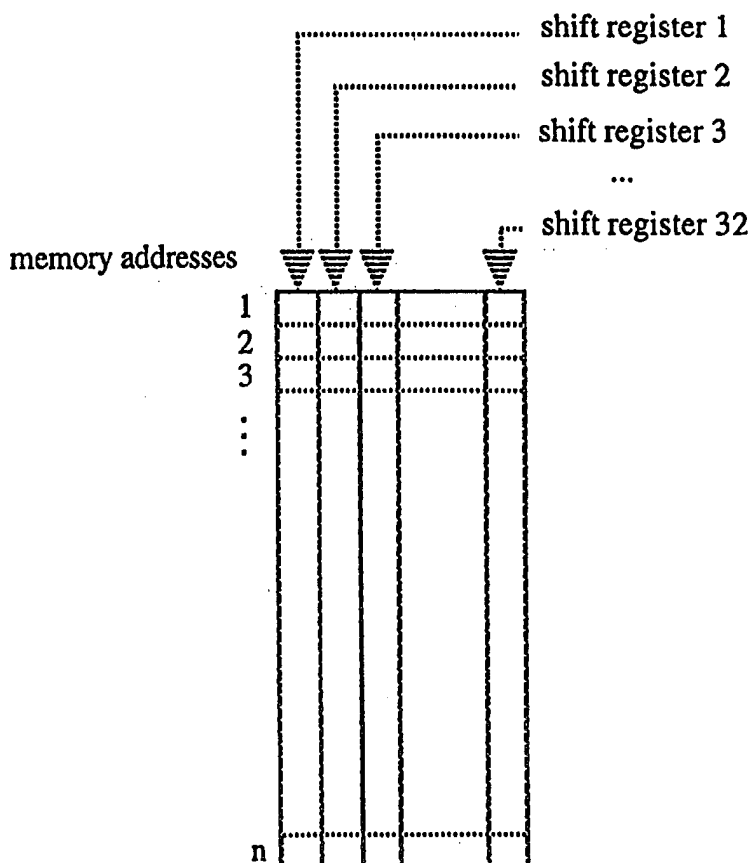


INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G11C 19/28, 7/00	A2	(11) International Publication Number: WO 98/56005 (43) International Publication Date: 10 December 1998 (10.12.98)
(21) International Application Number: PCT/SE98/01054 (22) International Filing Date: 3 June 1998 (03.06.98) (30) Priority Data: 9701874-1 3 June 1997 (03.06.97) SE (71)(72) Applicant and Inventor: BELIK, Ferenc [SE/SE]; Båtyxevägen 46, S-226 56 Lund (SE). (74) Agent: AWAPATENT AB; P.O. Box 5117, S-200 71 Malmö (SE).		(81) Designated States: CN, HU, JP, KR, RU, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE). Published <i>Without international search report and to be republished upon receipt of that report.</i>

(54) Title: METHOD AND DEVICE FOR DATA SEQUENCE MANIPULATION**(57) Abstract**

A random access memory and methods for insertion and deletion of data elements in the memory are presented. The memory comprises a number of parallel shift registers. Each memory cell consists of bits from a number of shift registers. The method for insertion or deletion of a data element into a sequence of data elements, contained in a number of consecutive memory cells comprise the steps of localizing a position at which a data element is to be inserted or deleted, moving the data elements in a controllable number of consecutive memory cells simultaneously to create space for a data element or to delete a data element and, in the case of insertion, inserting the data element into the sequence of data elements at the localized position. As a result, efficient sorting and dynamic searching are achieved, the need of pointers in linked list structures eliminated.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon	KR	Republic of Korea	PL	Poland		
CN	China	KZ	Kazakstan	PT	Portugal		
CU	Cuba	LC	Saint Lucia	RO	Romania		
CZ	Czech Republic	LI	Liechtenstein	RU	Russian Federation		
DE	Germany	LK	Sri Lanka	SD	Sudan		
DK	Denmark	LR	Liberia	SE	Sweden		
EE	Estonia			SG	Singapore		

METHOD AND DEVICE FOR DATA SEQUENCE MANIPULATIONTechnical field of the invention

This invention relates to a method of inserting data elements into and deleting data elements from a sequence of data elements, and to a random access memory designed therefor, according to the preambles of claims 1, 2, 9, and 12.

Background art

Sorting and searching are two of the most studied topics in computer science today. Virtually every general computer science textbook discusses a variety of sorting algorithms, search trees and priority queues. It is estimated that about 25% of all computation time in the world is spent sorting and searching. The importance of efficient sorting and searching has increased with the advent of the World Wide Web.

In almost all types of data computation the data is sequentially arranged, i.e. arranged in an ordered collection of data elements in which the position of an element plays a significant role.

Today's computers usually comprise a finite program, a finite collection of registers, each of which can store a single integer or real number, and a memory consisting of an array of memory cells of word length, each of which has a unique address and can hold data. In one step, a computer of the type presented above can perform a single arithmetic or logical operation on the contents of specified registers, place the contents of a memory cell whose address is in a register into another specified register, or store the contents of a register in a memory cell whose address is in another register.

A serious deficiency in computers of this type is that there is no "natural" way to sort data elements and handle sequences of data elements, due to the way the

CONFIRMATION COPY

memory is constructed. As a result, arrays and linked lists are used together with different sophisticated algorithms, search trees, priority queues, and similar "unnatural" solutions. Even though this minimizes the deficiencies, some problems of a serious nature remain. For example, in order to achieve quick addressing of an element in a sequence, the data sequence needs to be represented by an array in the random access memory. This leads to an addressing time that is independent of the array size, while the time for the insertion and deletion of data elements in the array is dependent on the size of the array. If, on the other hand, rapid insertion and deletion of data is required, the data sequence needs to be represented by a linked list in the random access memory. As a result, the data insertion and deletion time is independent of the size of the linked list, while the addressing time is dependent on the size of the linked list.

Summary of the invention

The purpose of the present invention is therefore to eliminate the problems mentioned above and to provide a more efficient method for combining rapid insertion and deletion with rapid addressing, and thereby provide more efficient methods for working with sequentially arranged data elements in general.

This purpose has been achieved by the methods and by a random access memory that has been designed to be used together with the methods, as set forth in the appended claims.

The methods for efficient insertion of a data element into a sequence of data elements, and for efficient deletion of a data element from a sequence of data elements, are mainly based on two ideas. The first idea is that the number of consecutive memory cells in the sequence of data elements that are simultaneously moved is controllable, and the second idea is that the number of steps the data elements in the consecutive memory cells

are simultaneously moved is controllable. With the possibility of controlling the above parameters, several advantages appear, some of which are presented here.

In a preferred embodiment of the invention, a sequence of data elements is represented as an array which is contained in a number of memory cells in a random access memory. Increased flexibility in the manipulation of data elements makes operations on arrays consisting of unsorted data elements, sorted data elements, or arrays with at least one sorted and one unsorted group of data elements equally easy. By choosing the number of words to be moved, and the number of address steps for the words to be moved, sorting of an array and dynamic searching in a sorted array is elementary to one skilled in the art. In the preferred embodiment, sorting of a sequence of data elements which is represented by an array is accomplished by a Binary Insertion Sort algorithm. By using dynamic searching, a position in a sequence of sorted data elements in an array at which a data element is to be searched, inserted, or deleted is found by binary search.

As a consequence of using arrays in which data elements can be entered in an efficient way, the need of pointers in linked lists is eliminated, which makes computer programs safer.

The increased flexibility in the manipulation of data elements have further advantages. For example, data elements do not need to be of word size, that is, the same size as the memory cells, but may instead consist of several words. Large data elements comprising much information can be manipulated almost as easily as smaller data elements.

The random access memory according to the invention preferably comprises a number of parallel shift registers. A shift register is a memory, consisting of memory cells of bit size, in which the contents of a controllable number of consecutive bit-sized memory cells can be

moved simultaneously a controllable number of steps towards higher or lower address numbers. The shift registers do not need to be physically connected in parallel, as long as the function described below is achieved. In a preferred embodiment of the random access memory, each memory cell consists of bits from a number of shift registers corresponding to the number of bits in the memory cell, the bits having the same location in the shift registers. Thus, when the same consecutive number of bits in each shift register that is comprised in consecutive memory cells is moved a certain number of steps, the contents of the same consecutive number of memory cells in the random access memory will be moved accordingly.

A random access memory of the above type can preferably be included in a processing unit, such as a computer or some other type of electronic device, where sorting, dynamic searching, list manipulation, and other manipulation of data sequences are important tasks.

Brief descriptions of the drawings

A preferred embodiment of the invention will now be described, by way of example only, with reference to the accompanying drawings.

Fig 1 shows a schematic diagram of the architecture of a Random Access Memory according to the invention.

Fig 2 shows the process of insertion of a data element into an array of data elements according to the invention.

Fig 3 shows the process of deletion of a data element from an array of data elements according to the invention.

Fig 4 shows one insertion step of Binary Insertion Sort implemented on a Random Access Memory according to the invention.

Detailed description of preferred embodiments of the invention

Referring to fig 1, a preferred embodiment of the random access memory according to the invention comprises
5 a number of parallel shift registers, each of which is m bits long, m being the highest address number in the memory. The number of shift registers corresponds to the size of a memory word or memory cell. The first bits of the parallel shift registers form the first memory word,
10 the second bits of the parallel shift registers form the second memory word, and so on. By constructing the random access memory as parallel shift registers, it is possible to move the contents of several consecutive words simultaneously.

15 Additional circuitry is provided that makes it possible to choose the number of words that are to be shifted, and to choose the number of address steps by which the words are shifted towards higher or lower address numbers. Today the random access memory according to the
20 invention can be realized in a size of about 8 Mbytes on a chip, if a controllable number of consecutive memory cells are simultaneously movable one step only. There is a large number of circuits suitable for performing the operations described above in a random access memory, and
25 the construction of a random access memory with the above characteristics can therefore be carried out by a person skilled in the art. The choice of how many words that are to be moved can be made either by the user or by the software depending on the situation. The specified addresses and numbers are stored in registers.
30

Referring now to fig 2, a method of insertion of a data element or a word x into a sequence of data elements according to a preferred embodiment of the invention will be described. In the random access memory according to
35 the invention, a sequence of data elements is implemented as an array, with one data element in each memory word. Fig 2 shows an allocated memory area of the random access

memory in which the sequence of data elements is allowed to expand or retract. For explanatory purposes two numbers i and n , referring to address positions in the memory, that is indices in the array, will be used in the following discussion. i corresponds to an address at which an element is to be inserted, and n corresponds to the address of the last word in the data sequence. Since it is possible to select the number of words that are to be moved, as well as the number of steps by which the words are to be moved, insertion of data is extremely easy and can be carried out with great efficiency. In order to insert a word at the address i in a data sequence which is implemented as an array, three steps are carried out. In step 1, the i :th position of the array is looked up. In step 2, the words between the i :th position and the n :th position in the array are simultaneously moved one step in the direction of increasing addresses. Finally, in Step 3, the new word is inserted at the address i . This method of inserting a word can obviously also be extended to inserting data elements consisting of several words.

The time complexity for each of the operations described above is $O(1)$. If a step preceding step 1 is necessary in order to retrieve the data element x from another sequence of data elements before inserting it into the new sequence, the time complexity for this step will determine the overall worst time complexity for the insertion of the data element x into the sorted array. For more information on time complexity, see computer science textbooks such as Jeffrey H Kingston: "Algorithms and Data Structures", Addison-Wesley, 1992.

Referring to fig 3, which also shows an allocated memory area of the random access memory in which the sequence of data elements is allowed to expand or retract the deletion of a data element or data word will be described. Here, i and n denote the same address positions in the array as in fig 2. The deletion of the contents at

address i is simply carried out by simultaneously moving the elements between position $i+1$ and n one step in the direction of decreasing addresses. The time complexity for this operation is $O(1)$. According to the methods
5 shown in figs 2 and 3, the invention combines rapid insertion and deletion with rapid addressing, which eliminates the need of pointers in linked lists, and makes computer programs safer and more efficient.

Fig 4 shows how a one step in a Binary Insertion
10 Sort algorithm can be carried out using a random access memory according to the invention. The data elements, which are of word size, are contained in an array consisting of both a sorted group of data elements and an unsorted group of data elements. For explanatory purposes
15 two address numbers, that is indices in the array, i and k will be used. k is the address of the last sorted data element. In the insertion step shown in fig 4, the first unsorted data element, which is at address $k+1$, is to be inserted among the sorted elements. i is an address number in the sorted group of data elements, at which the
20 data element at the address $k+1$ in the unsorted group of data elements is to be inserted. In Step 1, the address i is found by using binary search. The address i and the data element at the address $k+1$ are moved to two different registers. The worst case time complexity for finding the place of the data element using binary search is
25 $O(\log n)$, where n is the number of elements to be sorted. In Step 2, consecutive elements from addresses i to k are simultaneously moved one address step in the direction of increasing addresses. The time complexity for this is
30 $O(1)$ for the worst case time. In Step 3, the data element retrieved from address $k+1$, now held in the register, is inserted at the address i . The time complexity for this is $O(1)$.

35 As can be seen, after performing n insertion steps the worst case time complexity for sorting an array consisting of sorted and unsorted data elements is $O(n \log$

n). Sorting can be achieved today in conventional random access memories by using sorting algorithms such as Merge Sort and Heap Sort with a worst case time complexity of $O(n \log n)$, and Quick Sort with an average time complexity of $O(n \log n)$. However, the asymptotic time complexity hides a constant. Improvements of Quicksort, today's most efficient sorting algorithm, such as using straight insertion sort for small subfiles, have been recommended in order to minimize this constant. For more information on improvements of Quicksort, see Robert Sedgewick: "Algorithms in C++", Addison-Wesley, 1992. In a random access memory according to the invention, the constant hidden in the asymptotic time complexity has a lower value for the Binary Insertion Sort algorithm than for any of the algorithms above, and is thus more efficient.

With a sorted array in a random access memory according to the invention, dynamic searching, that is the implementation of the operations in Table 1, is simple and efficient. The table shows a comparison between the worst case time complexity for different searching methods in a conventional random access memory and in a random access memory according to the invention. x represents a data element, n the number of data elements contained in the sorted array, and i represents an address number of a data element in the sorted array.

The operations `find(x)` and `index_of(x)` finds the data element x and its address, respectively, by binary search in the sorted array, and have a time complexity of $O(\log n)$.

The operation `insert(x)` comprises the steps of finding the position where the data element x is to be inserted using binary search, and inserting it into the sorted array. These two steps have a time complexity of $O(\log n)$ together.

The operations `delete(x)` and `delete_min` are deletion operations from a sorted array, and have an $O(1)$ time

complexity. In Table 1, it is assumed with the operation $\text{delete}(x)$ that the address of x is known.

The operations find_min , find_max , $\text{find_next}(x)$, $\text{find_previous}(x)$, and $\text{find}(i)$ correspond to finding the first data element, the last data element, the data element succeeding x , the data element preceding x , and the data element at address i in a sorted array. All these operations involve finding addresses in the random access memory, which has an $O(1)$ time complexity.

Conventional random access memory					Random access memory according to the invention
	2-3 tree	Splay-tree	Heap	Binomial queue	Sorted array
$\text{find}(x)$	$O(\log n)$	$O(\log n)$			$O(\log n)$
$\text{insert}(x)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
$\text{delete}(x)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$
delete_min	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$
find_min	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$
find_max	$O(\log n)$	$O(\log n)$			$O(1)$
$\text{find_next}(x)$	$O(\log n)$	$O(\log n)$			$O(1)$
$\text{find_previous}(x)$	$O(\log n)$	$O(\log n)$			$O(1)$
$\text{find}(i)$	$O(\log n)$	$O(\log n)$			$O(1)$
$\text{index_of}(x)$	$O(\log n)$	$O(\log n)$			$O(\log n)$

TABLE 1: Comparison of search methods (the times for Splay-tree are amortized times).

CLAIMS

1. A method for insertion of a data element into a sequence of data elements, contained in a plurality of consecutive memory cells in a random access memory, comprising the steps of:

- localizing the position at which the data element is to be inserted into the sequence of data elements;

- creating space at the localized position in the sequence of data elements by moving the data elements in a number of consecutive memory cells simultaneously;

- inserting the data element into the sequence of data elements at the localized position;

characterized in

- that the number of consecutive memory cells whose data elements are simultaneously moved is controllable.

2. A method for deletion of a data element from a sequence of data elements, contained in a plurality of consecutive memory cells in a random access memory, comprising the steps of:

- localizing the position in the sequence of data elements from which the data element is to be deleted;

- moving the data elements in a number of consecutive memory cells simultaneously, so that the data element at the localized position is replaced;

characterized in

- that the number of consecutive memory whose data elements are simultaneously moved is controllable.

3. A method according to claim 1 or 2, wherein the number of steps that the data elements in the consecutive memory cells are simultaneously moved is controllable.

4. A method according to any of the preceding claims, wherein the insertion and deletion respectively of data elements is performed on an unsorted sequence of data elements.

5. A method according to any of the claims 1-3, wherein the sequence of data elements is a sorted sequence of data elements.

6. A method according to any of the claims 1-3, wherein the insertion and deletion respectively of data elements is performed on a sequence of data elements in a Random Access Memory, the sequence of data elements consisting of at least one sorted and one unsorted group of data elements.

7. A method according to claim 5 or 6, wherein the step of localizing the position in the sequence of sorted data elements is performed by binary search.

8. A method according to claim 1 or 2, wherein the insertion or deletion of a data element in the sequence of data elements is performed by a Binary Insertion Sort algorithm.

9. A random access memory comprising a number of memory cells for storing data elements, each memory cell consisting of a certain number of bits
c h a r a c t e r i z e d i n

that the random access memory comprises circuits enabling simultaneous moving of the data elements in a controllable number of consecutive memory cells.

10. A random access memory according to claim 9, the random access memory further comprising circuits enabling simultaneous moving of the data elements in a controllable number of memory cells a controllable number of steps.

11. A random access memory according to any of the claims 9-10, wherein the random access memory further comprises a number of parallel shift registers, each memory cell consisting of bits from a number of shift registers corresponding to the number of bits in the memory cell, the bits having the same bit location in each shift register.

12. A processing unit characterized in that it comprises a random access memory according to any of the claims 9-11.

Fig 1

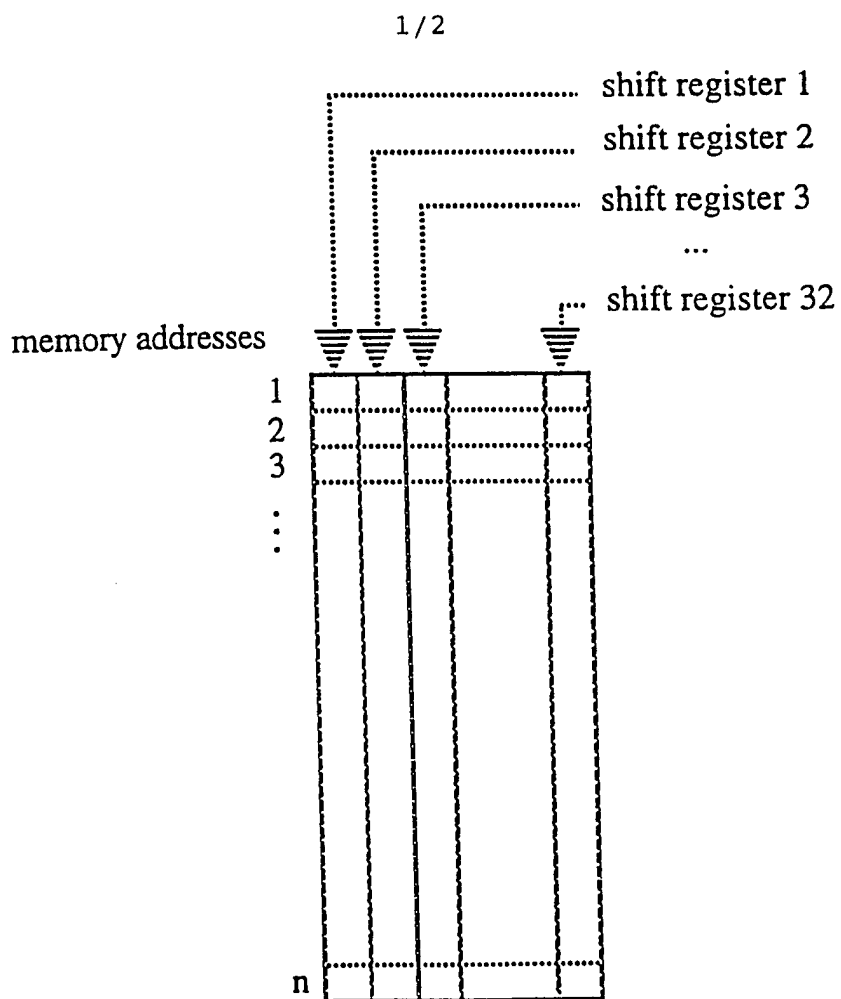
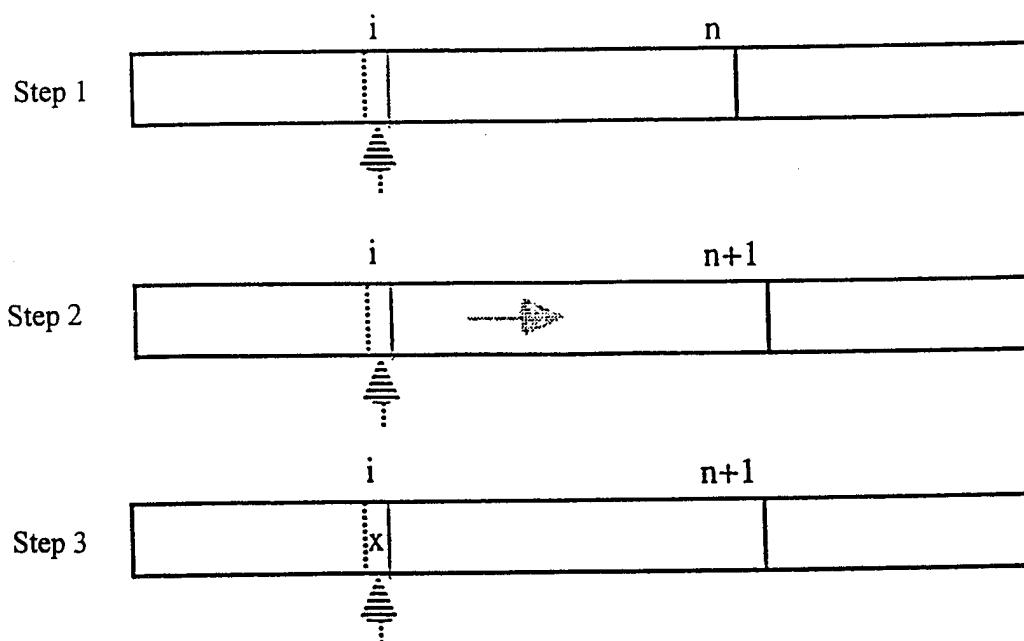


Fig 2



2/2

Fig 3

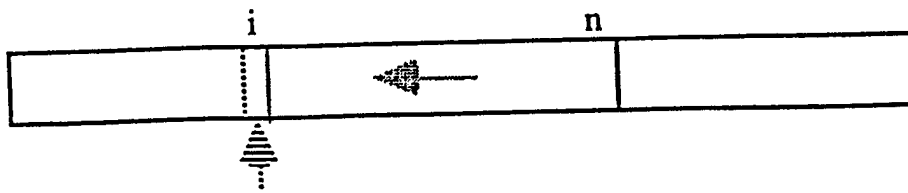


Fig 4

