Sorting Algorithms

Part 1: Internal Sort Algorithms

Topics

- Need for sorting
- Internal and External Sorting
- Bubble sort
- Selection sort
- Quick sort
- Merge sort

Motivation

- Random file access is very slow in pile files (exhaustive search) but almost fast in sorted sequential files (Binary or Interpolation search)
- Example: T_F in the hospital pile file with 16,667 blocks needs 7000 msec but 326 msec if sorted

Sort Algorithm Types

- Internal Sorting Algorithms: All data is in memory
- External Sorting Algorithms: Only a part of data is in memory
- External sorting algorithms are more suitable for sorting large files

Slow and Fast Algorithms

- Simple/Slow algorithms: The time needed by these algorithms is of order $O(n^2)$.

  (n is the number of data items)

  Example:

  With 1,000,000 data items (records), about 1,000,000,000,000 instructions are run.

Slow and Fast Algorithms

- Fast algorithms need $n\log_2 n$ instructions for sorting $O(n\log_2 n)$. These algorithms are more complex.

- Example: with 1,000,000 data items 20,000,000 instructions are run.
- For this example, fast algorithms are 50,000 times faster
- Slow algorithms are suitable for small data sets

Internal Slow Algorithms

- Bubble Sort
- Selection Sort

Slow Algorithm 1: Bubble Sort

- Compare each data item with its next neighbor, if larger, then swap them

- Repeat until no swap happens in a pass

Bubble Sort

First Pass

(**5 1** 4 2 8) → (**1 5** 4 2 8)
(1 **5 4** 2 8) → (1 **4 5** 2 8)
(1 4 **5 2** 8) → (1 4 **2 5** 8)
(1 4 2 **5 8**) → (1 4 2 **5 8**)

Second Pass:

(**1 4** 2 5 8) → (**1 4** 2 5 8)
(1 **4 2** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

Third Pass:

(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

Bubble Sort

```
int data[MAX], i;
bool swapped;
do
{
   swapped = false;
   for ( i = 0 ; i<MAX – 1; i++ )
     if( data[ i ] > data[ i + 1 ] )
     {
       swap( A[ i ], A[ i + 1 ] );
       swapped = true;
     }
  }
  while( swapped==true);
```

Selection Sort

- Find the smallest value in the set and swap it with the first element.

- Put aside the first item, repeat the above steps with the remaining items

Selection Sort

First Pass

Find the smallest value in the set

Smallest = 5

(5 **1** 4 2 8) 1 < Smallest? Yes Smallest = 1

(5 1 **4** 2 8) 4 < Smallest? No

(5 1 4 **2** 8) 2 < Smallest? No

(5 1 4 2 **8**) 8 < Smallest? No

Swap(first element and smallest)

(**5** **1** 4 2 8) (**1** **5** 4 2 8)

Selection Sort

Second Pass

Find the smallest value in the set

Smallest = 5

(1 5 **4** 2 8) 4 < Smallest? Yes Smallest = 4

(1 5 4 **2** 8) 2 < Smallest? Yes Smallest = 2

(1 5 4 2 **8**) 8 < Smallest? No

Swap(second element and smallest)

(1 **5** 4 **2** 8) (1 **2** 4 **5** 8)

Selection Sort

Third Pass

Smallest = 4

(1 2 4 **5** 8)   5 < Smallest? No

(1 2 4 5 **8**)   8 < Smallest? No

No Swap

Fourth Pass

Smallest = 5

(1 2 4 5 **8**)   8 < Smallest? No

No Swap

Selection Sort

```
int data[MAX], i;
for( j=0; j<Max -1 ; j++ )
{
    smallest = data[j];
    small_index = j
    for ( i = j+1 ; i<MAX ; i++ )
    if( data[ i ] < smallest )
    {
     Smallest=data[i];
     Small_index = i;
    }
   Swap(data[j], data[small_index]);
 }
```

Internal Fast Algorithms

- Quick Sort
- Merge Sort

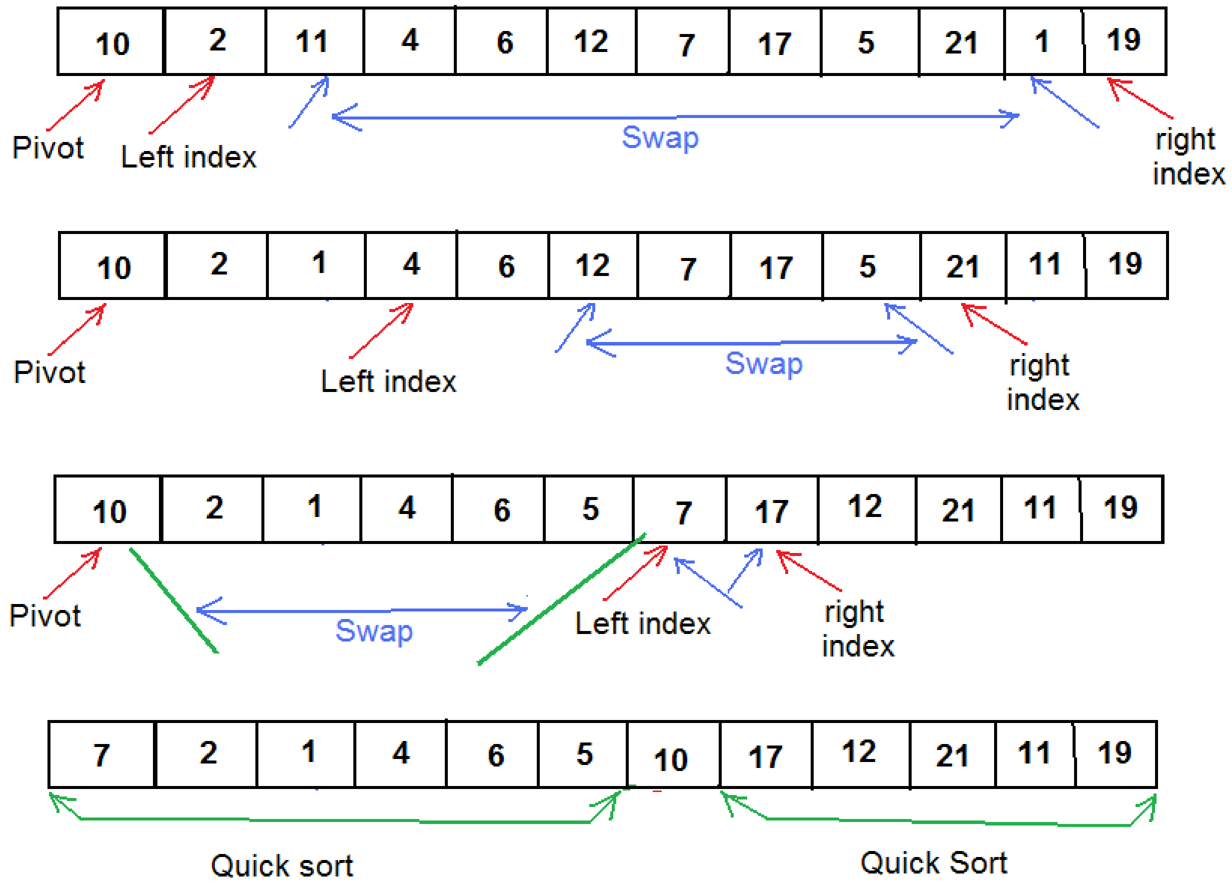
Quick Sort

- Take the first element as pivot
- Use two indexes, one starting from left the other from right
- Move the left index to right until a data item greater than the pivot is found
- Move the right index to the left until a data item smaller than the pivot is found
- Swap the items shown by the indexes

Quick Sort

- Repeat the above steps until indexes pass each other
- Swap pivot with the data shown by right index
- Call quick sort for left side of the pivot
- Call quick sort for the right side of the pivot

Quick Sort



10 2 11 4 6 12 7 17 5 21 1 19
Pivot
Left index
Swap
right index
10 2 1 4 6 12 7 17 5 21 11 19
Pivot
Left index
Swap
right index
10 2 1 4 6 5 7 17 12 21 11 19
Pivot
Swap
Left index
right index
7 2 1 4 6 5 10 17 12 21 11 19
Quick sort
Quick Sort

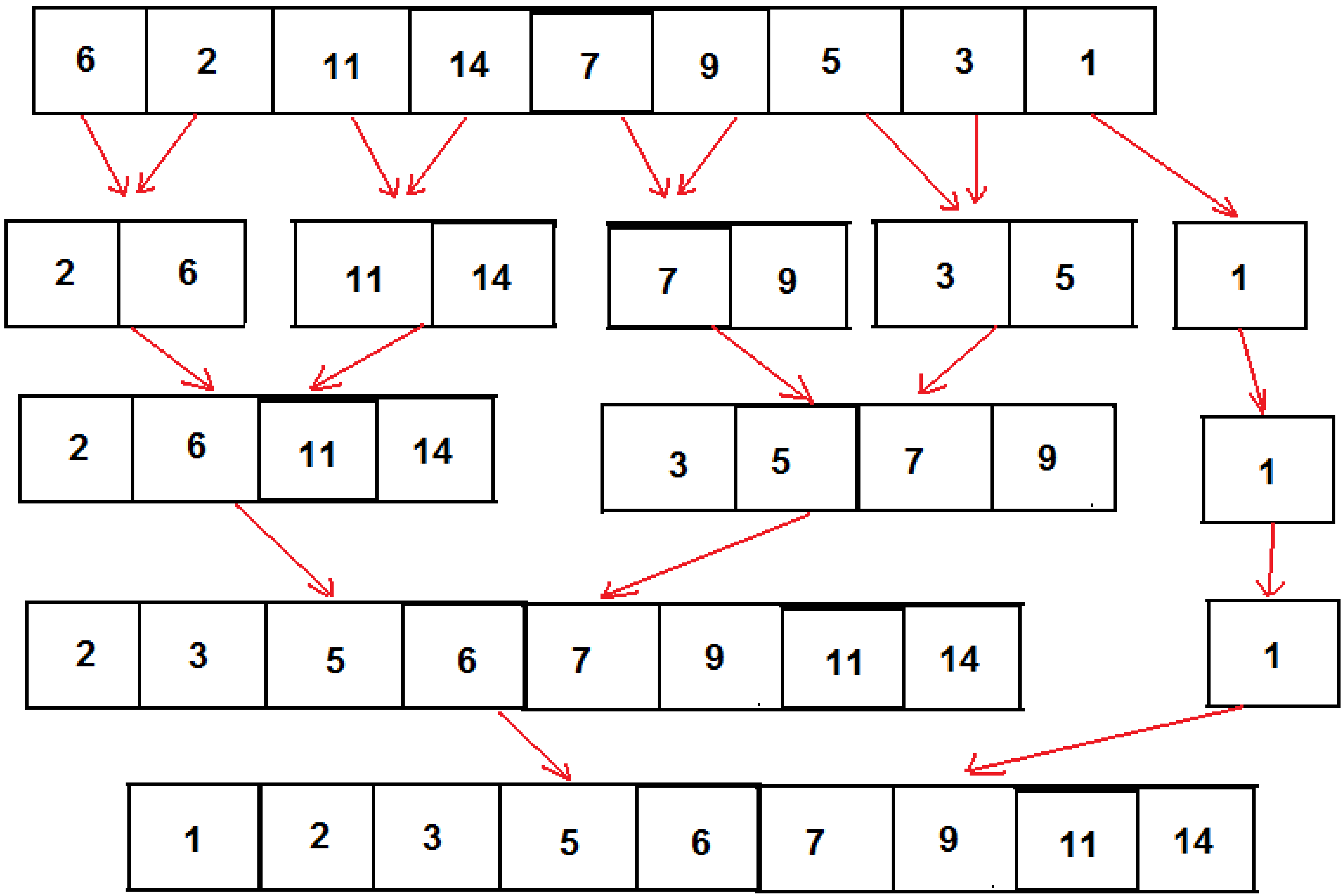
Quick Sort

```
Pivot = data[0];
Left = 1;
Right = size-1;
while( Left < Right )
{
    while( data[Left] < Pivot )
            Left ++;
    while( data[Right] > Pivot )
            Right--;
     if( Left < Right )
        Swap(data[Left] , data[Right]);
}
Swap( data[0] , data[Right] );
QuickSort( data, Right -1 );
QuickSort( &data[Left], size – Left );
```

Merge Sort

- If data is given as two sorted segments then we can merge them in a single sorted part
- Assume each data item as a sorted part
- Merge each pair of parts
- Repeat until a single list is found

Example (Merge Sort)



Merge Algorithm

- Compare the top-most elements of the two lists and pick the smaller one until the end of one of the lists is reached
- Add remaining elements from the other list

Merge

```
i = 0; j = 0; k = 0;
while( i< size1 && j < size2 )
{  if(data1[i] < data2[j] ){
        data3[k] = data1[i];
         i++; k++;
    }
    else{
        data3[k] = data2[j];
         j++; k++;
    }
}
```

Merge (Cont.)

```
if( i<size1 )
    while( i< size1 )
    {
            data3[k] = data1[i];
            i++; k++;
    }
else
    while( j< size2 )
    {
            data3[k] = data2[j];
            j++; k++;
    }
```

Merge Sort

```
void merge_sort(int m[] , int result[], int size)
{
   int left[size/2], right[size-size/2];
   if( size == 0 ) return;
   if( size == 1 )
   {
         result[0] = m[0];
         return;
   }
   merge_sort(m , left, size/2);
   merge_sort( &m[size/2] , right,  size – size/2) ;
   merge(left, right, result);
}
```

Questions?

Quiz

- Using Quick sort algorithm, sort the following data. Show only one pass.

25, 12, 3, 31, 32, 11, 15, 2, 44, 13, 65, 5, 30, 38