International Journal Of Engineering Research & Management Technology ISSN: 2348-4039

July- 2014 Volume 1, Issue-4

Email: editor@ijermt.org www.ijermt.org

A Study on Different Types of Sorting Techniques

Priyanka Gera
Department of Information and Technology
Dronacharya College of Engineering
Gurgaon, Haryana

Ashish Gill
Department of Information andTechnology
Dronacharya College of Engineering
Gurgaon, Haryana

1. ABSTRACT

Our research paper aims at "Sorting Techniques". Sorting is the process of placing elements from a collection in some kind of order. A list of words could be sorted alphabetically or by length. We are going to discuss all types of sorting techniques in our paper so that we can study most of the sorting techniques through our paper. We had not only discussed types of sorting but have also explained each one of them followed with a small piece of program which clearly indicates how each technique is going to work when executed. Sorting really makes our work easy. Without sorting it would be really difficult for us to search for the desired element in the list. But with the help of sorting we can do this in just a few minutes and that too with efficiency. So sorting helps us in our day to day life also.

Keywords: External sorting, Internal sorting, in place, Sorting, Stable, Swapping

2. INTRODUCTION

The efficiency of data handling can often be substantially increased if the data is sorted according to some criteria of order. For example it would be really difficult to find a name in the telephone directory if the names are not alphabetically arranged. Same case goes with the dictionaries, book indexes, payrolls, bank accounts, student lists and other alphabetically organized materials. It is very convenient for us if our data is in sorted order. So retrieval of information is made easier when it is stored in some predefined order. We have many sorting techniques available to us. We will discuss all types of sorting techniques and will also compare them according to their features and runtime. So sorting is really a important and interesting area to study in. Before getting into specific algorithms, we should think about the operations that can be used to analyze a sorting process. First, it will be necessary to compare two values to see which is smaller (or larger). In order to sort a collection, it will be necessary to have some systematic way to compare values to see if they are out of order. The total number of comparisons will be the most common way to measure a sort procedure. Second, when values are not in the correct position with respect to one another, it may be necessary to exchange them. This exchange is a costly operation and the total number of exchanges will also be important for evaluating the overall efficiency of the algorithm.

3. PROPERTIES OF SORTING TECHNIQUES

3.1 Stable

If the sorting algorithm preserves the relative order of any two equal elements, then the sorting order is stable. For example when the cards are sorted by rank with a stable sort, the two 5s must remain in the same order in the sorted output that they were originally in. When they are sorted with a non-stable sort, the 5s may end up in the opposite order in the sorted output. When sorting some kinds of data, only part of the data is examined when determining the sort order. For example, in the card sorting example to the right, the cards are being sorted by their rank, and their suit is being ignored. This allows the possibility of multiple different correctly sorted versions of the original list. Stable sorting algorithms choose one of these, according to the following rule: if two items compare as equal, like the two 5 cards, then their relative order will be preserved, so that if one came before the other in the input, it will also come before the other in the output.

3.2 In Place

If an algorithm does not require an extra memory space except for few memory units, then the algorithm is said to be in place.

4. TYPES OF SORTING

4.1 Internal sorting

Internal sorting is applied when the entire collection of data to be sorted is small enough so that the entire sorting can take place within the main memory. The time required to read or write is not considered to be significant in evaluating the performance of internal sorting methods. Internal sorting takes place in the main memory of a computer, where we can use the random access capability of the main memory to advantage in various ways. By internal sorting we are arranging the numbers within the array only which is in computer main memory.

4.2 External sorting

When large data have to be sorted they cannot be brought into the main memory at the same time. In such situations we use external sorting where the entire file may reside on some secondary storage device. At the same time based on the availability of the storage space in the main memory, some portion of the data from the secondary storage device will be brought into the main memory that will be sorted and sent back to the secondary storage device. Then the next portion will be brought into the main memory, sorted and then sent to the secondary storage device till the time when whole of the file which is there on the secondary storage device is sorted.

5. TYPES OF INTERNAL SORTING

5.1 Bubble Sort

Bubble sort, also known as exchange sort, is a simple sorting algorithm. It works by repeatedly stepping through the list to be sorted. Comparing two items at a time and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which means the list is sorted. The algorithm gets its name from the way smaller elements bubble to the top of the list via swaps. In this method we assume that the lower value items from the set of items are light and bubble on to the top. The bubble sort is generally considered to be the most efficient sorting algorithm in common usage.

5.1.1 Code for bubble sort

```
Void bubble ( int x[ ], int n)
{
int  i, j, tmp;
for(i=0; i<n; i++)
{
        for(j=0; j<n-1; j++)
        {
                if(x[j] > x[j+1])
                {
                        tmp=x[j];
                        x[j] = x[j+1];
                        x[j+1] = tmp;
                }
          }
}
}
```

5.2 Selection Sort

The idea of selection sort is rather simple; we repeatedly find the next largest or smallest element in the array and move it to its final position in the sorted array. For example if we need to sort the array in increasing order we will begin by selecting the largest element and moving it to the highest index position. We can do this by swapping the element at the highest index and the largest element. We then reduce the effective size of the array by one element and repeat the process on the smaller sub array. The process stops when the effective size of the array becomes 1. Thus, the selection sort works by selecting the smallest unsorted item remaining in the list, and then swapping it with the item in the next position to be filled. This is the simple method of sorting.

5.2.1 Code for selection sort

```
Void selection ( int x[ ], int n)
{
int  i, j, larg, pos, c=0;
for ( i = n-1; i > 0; i--)
      {
      larg = x[0];
      pos=0;
      for(j = 1; j<=i; j++)
      c++;
      if ( x[j] > larg)
            {
             larg = x[j];
             pos = j;
            }
    }
    x[pos] = x[i];
    x[i] = larg;
    }
    printf("\n  number  of  comparisons  required = %d",c);
}
```

5.3 Insertion Sort

The insertion sort works just like its name suggests it- it inserts each item in its proper place in the final list. The simplest implementation of this requires two list structures – the source list and the list into which sorted items are inserted. To save memory , most implementations use an in place sort that works by moving the current item past the already sorted items and repeatedly swapping it with the preceding item until it is in place. This method is widely used by card players.

6.3.1 Code for insertion sort

```
Void insertion ( int x[ ] , int n)
{
int  i , j, tmp;
for( i=0; i<n; i++)
{
       tmp = x[i];
for(  j= i-1; j>=0; j--)
       {
        if( temp < x[j])
               x[j+1] = x[j];
        else
                break;
       }
x[j+1] = temp;
 }
 }
```

5.4 Shell Sort

Shell sort is the modification of the insertion sort. Shell sort improves on the efficiency of insertion sort by quickly values to their destination. In this instead of sorting the entire array at once the array is first divided into smaller segments. Then these segments are sorted separately using the insertion sort.

5.4.1 Code for shell sort

```
#define MAX 15
Void main ( )
{
int arr[MAX], i , j , k, n, incr;
printf(" enter no of elements");
scanf("%d", &n);
for ( i=0; i<n; i++)
{
printf(" enter element %d", i+1);
scanf("%d", &arr[i]);
}
printf(" unsorted list is");
for( i=0; i<n; i++)
printf("%d", arr[i]);
printf( " enter  maximum  increment (odd value)");
scanf("%d", &incr);
/* shell sort algorithm applied */
while( incr >=1)
{
for( j=incr; j<n; j++)
{
k=arr[ i];
for( i = j- incr; i>=0 && k<arr[i]; i = i-incr)
arr[ i+incr]= arr[i];
arr[ i +incr]=k;
}
printf(" increment = %d", incr);
for ( i=0; i<n; i++)
printf(" %d", arr[i]);
incr= incr-2;
}
printf{"sorted list is")
for(i=0; i<n; i++)
printf("%d", arr[i]);
}
```

5.5 Merge Sort

Merge sort is a algorithm that uses the idea of divide and conquer approach.

5.5.1 Code for merge sort through recursion

```
# define MAX 20
int array [MAX];
Void merge ( int low, int mid, int high)
{
int temp[MAX};
int i= low;
int j= mid+1;
int k= low;
While (( i<= mid) && (j <= high))
{
        if ( array[i]<=array[j])
        temp [k++]= array[i++];
        else
         temp [k++]= array[j++];
}
While ( i<=mid)
        temp[k++]= array[i++];
while (j<= high)
        temp[k++]= array[j++];
while (j<=high)
       temp[k++]= array[j++];
for (i=low; i<= high; i++)
        Array[i]= temp[i];
}
Void merge_ sort( int low, int high)
{
int mid;
if (low != high)
{
       mid=(low+ high)/2;
       merge_ sort(low,mid);
       merge_ sort(mid+1, high);
       merge(low, mid, high );
}
}
```

6. CONCLUSION

In our day to day life we come across many such instances where we literally need sorting. Without sorting it would be very difficult to search a particular thing in the list. At the same time it would be really time consuming as it would take a lot of time if we are going to search a particular thing in our list one by one. So sorting is the best option in that case in which we can arrange elements according to our wish. Sorting makes our task really convenient, easy and efficient. Manually there are chances that we may commit a mistake but if we are following the code of any type of sorting algorithm then there are the minimal chances of any kind of mistake. So sorting holds a very important place in computer science.

7. REFERENCES

1. Search engines like Google, Wikipedia
2. ACM transactions on algorithms
3. Various books on Data Structures
5. The sorting techniques: a tutorial paper on card sorts, picture sorts and item sort