CHAPTER 18 (DAY 3 After noon)

Chapters and Topics page

CHAPTER – 18

SORTING OPERATIONS

INTRODUCTION

The concept of an ordered set of elements is one that has considerable impact on our daily lives. Finding a particular element in a given list is called ***search***. Searching for the element through ordered list is easier than searching through an unordered list. Elements can be ordered and kept in sorted order on the basis of numerical or character value.

A ***file*** is a holder of information, which has n items of information. Each item of information in the file is a ***record***. The file will have records numbered *r[0] …..r[n-1]*. A ***key*** is a sub-field of the record itself, a key *k[i]* is associated with a record *r[i]*. The file is said to be **sorted** on the **key** if $i < j$ implies that *k[i]* precedes *k[j]* in some ordering on the **keys**.

A ***sort*** can be classified as being **internal** if the records that it is sorting are in main memory, or ***external*** if some of the records that it is sorting are in auxiliary storage. A sort takes place either on the records themselves or on an auxiliary table of pointers.

If the amount of data stored in each of the records is large then the overhead involved in moving the actual data is prohibitive. In this case an auxiliary table of pointers may be used so that these pointers are moved instead of the actual data itself. This is called ***sorting by address***.

SELECTION SORT

```
void selectionsort (int numbers[], int array_size)
{
        int i, j;
         int min, temp;

        for (i = 0;  i < array_size-1;  i++)
        {
                min = i;

                for (j = i + 1; j < array_size; j++)
                {
                        if (numbers[j] < numbers[min])
                           min = j;
                }  /* end of inner for loop  */

                 swap (&iap[i], &iap[min]);
                 printf ("\n Secletion sort iteration: %d : \n", i);
                 printarray (iap, ASIZE);

        }   /* end of outer for loop  */

} /* end of selection sort  */
```

Selection sort is a sorting algorithm, specifically an in-place comparison sort. It has O(n2) complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. **Selection sort** is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array.

Selecting the lowest element requires scanning all n elements (this takes n - 1 comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining n - 1 elements and so on, for (n - 1) + (n - 2) + ... + 2 + 1 = n(n - 1) / 2 = Θ(n2) comparisons (see arithmetic progression). Each of these scans requires one swap for n - 1 elements (the final element is already in place). Thus, the comparisons dominate the running time, which is Θ(n2).

SAMPLE OUTPUT

```
**********************************************************************
Data items in original order
 22  92  18  42   4  68  87  35  13  49  72   7  29  39  51

Calling Selection sort
**********

Secletion sort iteration: 0 :
  4  92  18  42  22  68  87  35  13  49  72   7  29  39  51

Secletion sort iteration: 1 :
  4   7  18  42  22  68  87  35  13  49  72  92  29  39  51

Secletion sort iteration: 2 :
  4   7  13  42  22  68  87  35  18  49  72  92  29  39  51

Secletion sort iteration: 3 :
  4   7  13  18  22  68  87  35  42  49  72  92  29  39  51

Secletion sort iteration: 4 :
  4   7  13  18  22  68  87  35  42  49  72  92  29  39  51

Secletion sort iteration: 5 :
  4   7  13  18  22  29  87  35  42  49  72  92  68  39  51
```

Secletion sort iteration: 6 :
4 7 13 18 22 29 35 87 42 49 72 92 68 39 51

Secletion sort iteration: 7 :
4 7 13 18 22 29 35 39 42 49 72 92 68 87 51

Secletion sort iteration: 8 :
4 7 13 18 22 29 35 39 42 49 72 92 68 87 51

Secletion sort iteration: 9 :
4 7 13 18 22 29 35 39 42 49 72 92 68 87 51

Secletion sort iteration: 10 :
4 7 13 18 22 29 35 39 42 49 51 92 68 87 72

Secletion sort iteration: 11 :
4 7 13 18 22 29 35 39 42 49 51 68 92 87 72

Secletion sort iteration: 12 :
4 7 13 18 22 29 35 39 42 49 51 68 72 87 92

Secletion sort iteration: 13 :
4 7 13 18 22 29 35 39 42 49 51 68 72 87 92