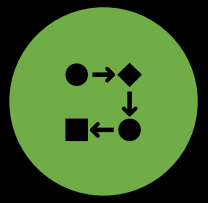
Sorting Algorithms

Why sorting!!

- Searching is a most common activity which we do in our daily life like: searching a shirt in our almirah, searching a book in the library, searching a word in the dictionary, searching info about weather, etc
- We have lots of data stored on our computers and we are always trying to extract a piece of information relevant to us by searching.
- Sorting comes into picture due to searching
- It is easy to search an ordered data, in place of an unordered data

What is sorting?



Sorting is the process of arranging items in a specific order, such as ascending or descending.

Arrange a messy pile of books

Organize student roll list by their last name.

Organize playing cards

Sort bank customers data based on their account numbers

Sort trains data by train number

Sort a phone list

Sort student answer sheets

Understanding Sorting and records

- A file of size n is a sequence of items r[0], r[1]…r[n-1].
- Each item in the file is called a record.
- A key k[i] is associated with each record and usually it is a subfield of the entire record.
- The file is ***said to be sorted*** on key if i<j implies ***k[i] precedes k[j]***
- ***k[i] precedes k[j]*** means ***k[i] is less than k[j]***
- A sort is classified as internal sort if the records sorted are in the main memory.
- If the records sorted are in some auxiliary storage, then it is called external sorting.
- A sort takes place either on records themselves or on auxiliary table of pointers

Why more than one sorting algo???Efficiency….

- There are many ways in which a data set can be sorted.
- Can we just identify one method that can be called the best method??
- What considerations needs to be taken into account to compare different sorting methods?

1. Amount of machine time necessary for running the algo (time complexity)
2. Amount of space necessary for the program
3. Length of time taken by programmer to code the algo

Sorting efficiency.. More issues

- Does Sorting efficiency depends upon the arrangement of input data in addition to the sorting algorithm used??
- Types of input data:
- Input data may be **already sorted**, **nearly sorted**, **completely random**, in **reverse order** or **nearly reverse**

Stable Sort

- A sorting technique is called stable if for all records i and j such that k[i] equals k[j], if r[i] precedes r[j] in the original file, then r[i] precedes r[j] in the sorted file too.
- The stable sort keeps the records with the same key in the same relative order that they were before the sort.
- Every record is identified by a unique key and we sort the whole file with respect to that key.
- While performing the sort, if we swap two records
- A sort can take place either on the records themselves or just key with pointers to records

Selection Sort

- Given an array of integer key values (INPUT)
- Generate an array of integer key values which are in ascending or descending order (OUTPUT)

Algorithm

Initialize i =0;

1. First find the smallest element of the array from ith element to (n-1)th element and put it at ith position.
2. Increment i by 1
3. Repeat the 1st instruction till i becomes equal to n-1

Selection sort

The important computation is number of comparisons!!

- The loop will run n-1 times
- First time, no. of comparisons will be n-1..
- Then, second time n-2…… 1
- Total no. = n(n-1)/2
- Complexity = $O(n^2)$
- Easy to understand
- Easy to code but not stable

Selection sort Code

```
selectionSort(int arr[], int n)
{ int i, j, min_index;
  for (i = 0; i < n-1; i++)
{
    min_index = i;
 for (j = i+1; j < n; j++)
if (arr[j] < arr[min_index])
min_index = j;
int temp = arr[min_index];
 arr[min_index] = arr[i];
 arr[i] = temp; }
}
```

Bubble sort

- Given an array of integer key values (INPUT)
- Generate an array of integer key values which are in ascending or descending order (OUTPUT)
- ***Basic idea of Bubble sort is to pass through the file /array sequentially several times such that in one pass the largest element of the array is placed at the end of array, and during this process, some of the elements keep moving closer to their correct position***
- Each pass consists of comparing each element in the array with its successor (x(i) with x(i+1)) and interchanging the two elements if they are not in proper order.

Bubble sort performance

- Maximum total no. of passes: n-1
- Total no. of comparisons: (n-1) +(n-2) +….1 = n(n-1)/2
- Can we modify this??
- Yes!! It is possible that the array is sorted only after x<n-1 passes, as in every pass elements are getting closer to their correct position
- How to find whether the array is sorted??
- If in any pass, if there are no swaps, this means array is sorted.
- To implement this, we can keep a flag to show whether any swap has taken place or not during a pass.

Bubble sort…

- So, if we stop the passes as soon as the array gets sorted then what will be the complexity of bubble sort??
- If after k passes, array gets sorted then number of comparisons will be:

(n-1) +(n-2)…..(n-k) which is equal to $(2kn-k^2-k)/2$

- But, it is still of the order of (n^2)
- Hence:
- We can say that ***best case*** is when array is already sorted, and the complexity is O(n)
- We can say that ***worst case*** is when array is in reverse order, and the complexity is $O(n^2)$

Code

- include <stdio.h>
- void bubbleSort(int arr[], int n)
- {
- for (int i = 0; i < n-1; i++)
- { for (int j = 0; j < n-i-1; j++)
- {
- if (arr[j] > arr[j+1])
- { int temp = arr[j];
- arr[j] = arr[j+1];
- arr[j+1] = temp; } } } }

Conclusion

- Selection sort is always $O(n^2)$
- Bubble sort is also $O(n^2)$….in worst case
- But..
- It is stable
- If the data is sorted or nearly sorted the complexity is closer to O(n)
- Best case: Data sorted
- Worst case: Data in reverse order
- ***Preferred sorting algo: if there is some information with the use that data is nearly sorted***
- ***Works better if: only keys are used instead of long records because lots of swapping is done in each pass***